INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, VOL. 26, 2671–2685 (1988)

A STEEPEST EDGE ACTIVE SET ALGORITHM FOR SOLVING SPARSE LINEAR PROGRAMMING PROBLEMS

S. W. SLOAN

Department of Civil Engineering and Surveying, The University of Newcastle, N.S.W., 2308, Australia

SUMMARY

A steepest edge active set algorithm is described which is suitable for solving linear programming problems where the constraint matrix is sparse and has more rows than columns. The algorithm uses a steepest edge criterion for selecting the search direction at each iteration and recurrence relations are derived which enable it to execute efficiently. The canonical form for the active set method is convenient for many applications and may be exploited to devise a simple crash procedure which is employed prior to phase one. A complete two-phase algorithm which incorporates the crash procedure is outlined. Only one artificial variable is needed to determine if the linear programming problem has a feasible solution in phase one. Some computational results are given to illustrate the effectiveness of the algorithm for a range of sparse linear programming problems. Comparisons between the steepest edge criterion and the traditional Dantzig criterion suggest that the former usually requires fewer iterations and often leads to substantial savings for large problems.

INTRODUCTION

Linear programming problems are traditionally solved using the revised simplex algorithm. Recently, Best and Ritter[1] have published an alternative procedure, known as the active set algorithm, which has the following canonical form:

$$\begin{aligned} &\text{Minimize} && \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ &\text{Subject to} && \mathbf{A}_1\mathbf{x} \leqslant \mathbf{b}_1 \\ & && \mathbf{A}_2\mathbf{x} = \mathbf{b}_2 \end{aligned}$$

where $\mathbf{c}$ is a vector of objective function coefficients of length n, $\mathbf{A}_1$ is an $m \times n$ matrix of inequality constraint coefficients, $\mathbf{A}_2$ is an $r \times n$ matrix of equality constraint coefficients, $\mathbf{b}_1$ is a vector of length m, $\mathbf{b}_2$ is a vector of length r and $\mathbf{x}$ is an unknown vector of length n which is to be determined. The active set algorithm has a very simple geometric interpretation, works with an active constraint matrix of dimension $n \times n$ and is ideally suited to problems where $n < m + r$.

This paper describes an efficient implementation of the active set procedure which employs a steepest edge heuristic for choosing the search direction at each iteration. The steepest edge scheme is similar to that developed by Goldfarb and Reid[2] for the revised simplex method and is relatively simple to implement. We also describe a complete two-phase algorithm for the active set method which incorporates a crash procedure and uses one artificial variable to determine if the linear programming problem has a feasible solution in phase one. The effectiveness of the proposed algorithm is illustrated by applying it to a variety of sparse linear programming problems that arise in the application of classical plasticity theory.

0029–5981/88/122671–15$07.50

Received 20 August 1987
Revised 21 April 1988

THE ACTIVE SET ALGORITHM

A comprehensive description of the active set algorithm may be found in Best and Ritter[1] and a detailed justification of the method will not be repeated here. The notion of active set procedures is also discussed in a more general context by Gill *et al.*[3] Each iteration of the algorithm, in a similar manner to the revised simplex technique, is comprised of three distinct steps. These are:

1. the determination of the search direction and test for optimality;
2. the determination of the maximum feasible step size;
3. updating of the solution and active constraint matrix.

At each iteration the objective function decreases (or in the case of a zero maximum feasible step size remains the same) and the algorithm moves along an edge of the feasible region from one vertex to another. Iteration is halted once it is no longer possible to decrease the objective function further by moving to an adjacent vertex. The canonical form required by the proposed active set algorithm, which is a slight variation of that used by Best and Ritter,[1] is

$$\begin{array}{lll} \text{Minimize} & \mathbf{c}^{\mathrm{T}}\mathbf{x} & \\ \text{Subject to} & \mathbf{a}_i^{\mathrm{T}}\mathbf{x} = b_i; & i = 1, \ldots, r \\ & \mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leqslant b_i; & i = r+1, \ldots, r+m \end{array}$$

In order to describe the algorithm succinctly we define an ordered index set for iteration j as $I^j = \{I_1^j, I_2^j, \ldots, I_n^j\}$, where each of the numbers I_i^j is an element of the set $\{1, 2, \ldots, r+m\}$. The index set I^j defines the constraints which are active (or binding) at iteration j and hence defines the active constraint matrix $\mathbf{D}^j$. If $I_i^j = k$, this implies that the ith row of the $n \times n$ matrix $\mathbf{D}^j$ is given by $\mathbf{a}_k^{\mathrm{T}}$ (the kth row of the constraint matrix). In order to start the algorithm with an arbitrary feasible point, it is also convenient to introduce artificial rows which are denoted by $I_i^j = 0$. If $I_i^j = 0$, this implies that the ith row of $\mathbf{D}^j$ is equal to the ith row of the identity matrix. Following Best and Ritter[1] and adopting the above notation, the active set algorithm may be described as follows.

Algorithm 1 (*Basic active set algorithm*)

Step 1.0—(Initialization)

Set $j = 0$. Start with an initial feasible point $\mathbf{x}^0$, an initial index set $I^0 = \{I_1^0, I_2^0, \ldots, I_n^0\}$ and $[\mathbf{D}^0]^{-1}$, where $[\mathbf{D}^0]^{\mathrm{T}} = [\mathbf{d}_1^0, \mathbf{d}_2^0, \ldots, \mathbf{d}_n^0]$ and $\mathbf{D}^0$ is non-singular. Each column $\mathbf{d}_i^0$ corresponds to a row of the constraint matrix such that $\mathbf{d}_i^0 = \mathbf{a}_{I_i^0}$ (or an artificial row in which case $I_i^0 = 0$ and $\mathbf{d}_i^0$ is the ith column of the identity matrix) and $0 \leqslant I_i^0 \leqslant r+m$. Each equality constraint $i = 1, 2, \ldots, r$ must be in I^0 (provided it is linearly independent of all other equality constraints)

Step 1.1—(Computation of search direction)

1.1.0 Set $\boldsymbol{\lambda}^j = [\mathbf{D}^j]^{-\mathrm{T}}\mathbf{c}$

1.1.1 If $\mathbf{D}^j$ contains no artificial rows (i.e. $0 \notin I^j$) go to Step 1.1.4

1.1.2 If $\lambda_i^j = 0$ for all i with $I_i^j = 0$ go to Step 1.1.4. Else determine the smallest index k such that

$$|\lambda_k^j| = \max_{\substack{1 \leqslant i \leqslant n \\ I_i^j = 0}} \{|\lambda_i^j|\}$$

1.1.3 Set $\mathbf{s}_k^j = [\mathbf{D}^j]^{-1}\mathbf{e}_k$ if $\lambda_k^j > 0$, or $\mathbf{s}_k^j = -[\mathbf{D}^j]^{-1}\mathbf{e}_k$ if $\lambda_k^j < 0$, and omit Steps 1.1.4–1.1.5

1.1.4 If $\lambda_i^j \leqslant 0$ for all i with $I_i^j > r$, exit with optimal solution $\mathbf{x}^j$. Else determine the smallest index k such that

$$\lambda_k^j = \max_{\substack{1 \leqslant i \leqslant n \\ I_i^j > r}} \{\lambda_i^j\}$$

1.1.5 Set $\mathbf{s}_k^j = [\mathbf{D}^j]^{-1}\mathbf{e}_k$

Step 1.2—(Computation of maximum feasible step size)

If $\mathbf{a}_i^{\mathrm{T}}\mathbf{s}_k^j \geqslant 0$ for all $i = r+1, \ldots, r+m$ and $i \notin I^j$, print message that problem is unbounded from below and stop. Else compute the smallest index l and σ^j such that

$$\sigma^j = \frac{\mathbf{a}_l^{\mathrm{T}}\mathbf{x}^j - b_l}{\mathbf{a}_l^{\mathrm{T}}\mathbf{s}_k^j} = \min_{\substack{r+1 \leqslant i \leqslant r+m \\ i \notin I^j \\ \mathbf{a}_i^{\mathrm{T}}\mathbf{s}_k^j < 0}} \left\{\frac{\mathbf{a}_i^{\mathrm{T}}\mathbf{x}^j - b_i}{\mathbf{a}_i^{\mathrm{T}}\mathbf{s}_k^j}\right\}$$

Step 1.3—(Update solution and active set data)

1.3.0 Set $\mathbf{x}^{j+1} = \mathbf{x}^j - \sigma^j\mathbf{s}_k^j$

1.3.1 Obtain I^{j+1} from I^j by setting $I_k^{j+1} = l$ and $I_i^{j+1} = I_i^j$ for $i = 1, n$ with $i \neq k$

1.3.2 Obtain $[\mathbf{D}^{j+1}]^{-1}$ from $[\mathbf{D}^j]^{-1}$ by replacing row k of $\mathbf{D}^j$ with $\mathbf{a}_l^{\mathrm{T}}$ or compute $[\mathbf{D}^{j+1}]^{-1}$ afresh using I^{j+1} and the constraint matrix coefficients

1.3.3 Replace j with $j+1$ and go to Step 1.1.

In the computation of the search direction, we use $\mathbf{e}_k$ to denote the kth column of the identity matrix. The central concept of the above algorithm is the active set. If constraint i belongs to the active set at iteration j (i.e. $i \in I^j$), this implies that the constraint is binding and hence $\mathbf{a}_i^{\mathrm{T}}\mathbf{x}^j = b_i$. If $i \notin I^j$ then, in general, $\mathbf{a}_i^{\mathrm{T}}\mathbf{x}^j < b_i$ (or $\mathbf{a}_i^{\mathrm{T}}\mathbf{x}^j = b_i$ if the constraint is active but linearly dependent on one or more of the constraints in I^j). Note that an equality constraint is always active throughout the optimization iterations provided that it is linearly independent of the other equality constraints. If this is not the case, the equality constraint is redundant and may be ignored. Note also that, if the current active constraint matrix contains an artificial row (i.e. $0 \in I^j$), the current iterate $\mathbf{x}^j$ need not define a vertex of the feasible region. The algorithm will, however, still terminate with the correct optimal solution, as discussed in detail by Best and Ritter.[1] At each stage in the iteration process, Step 1.1 determines which inequality constraint (or artificial constraint) may be dropped from the active set in order to decrease the current value of the objective function and the corresponding search direction. Step 1.2 ascertains the maximum step size which may be taken along the current search direction, whilst maintaining feasibility, and the corresponding inequality to be added to the active set. Step 1.3 concludes each iteration by updating the solution, the active set and the inverse of the active constraint matrix. In Step 1.3.2 it is not necessary to recompute the inverse of the active constraint matrix afresh for each iteration. More efficient procedures may be used which modify the inverse of a matrix due to a change in a row or column (see, for example, Sherman and Morrison[4]). After a number of iterations, however, it is usually necessary to compute $[\mathbf{D}^{j+1}]^{-1}$ afresh in order to minimize the effects of accumulated round-off error.

The majority of large linear programming problems that occur in practice have constraint matrices which are extremely sparse. It is not uncommon for the density of the overall constraint matrix to be significantly less than one per cent. Many of the developments in solving large sparse problems by the revised simplex method are directly attributable to developments in sparse matrix theory. Since the revised simplex method possesses many similarities to the active set algorithm, a number of the procedures developed for the former technique are also applicable to the latter. Rather than working with the inverse of the active constraint matrix directly, it is preferable (and indeed necessary for large sparse problems) to use sparse factorization techniques such as those described by Forrest and Tomlin,[5] Bartels[6] and Reid.[7,8] These methods are based on a product form of *LU* decomposition and preserve sparsity during the initial factorization and its subsequent updates by the replacement of rows or columns. A comprehensive discussion of the various procedures that have been developed for the revised simplex method may be found in Gill *et al.*[3] The factorization scheme adopted in our implementation is due to Reid.[7,8] Reid's algorithm produces a factorization of an $n \times n$ sparse matrix $\mathbf{A}$ of the form

$$\mathbf{A} = \mathbf{LPUQ} \tag{1}$$

The inverse of the matrix $\mathbf{L}$ is held as a product according to

$$\mathbf{L}^{-1} = \mathbf{M}_k \mathbf{M}_{k-1} \ldots \mathbf{M}_1$$

where the k matrices $\mathbf{M}_i$ differ from the identity matrix in just one element. The matrix $\mathbf{U}$ is upper triangular and $\mathbf{P}$ and $\mathbf{Q}$ are permutation matrices. The suite of FORTRAN subroutines described in Reid[7,8] solves sets of equations of the form

$$\mathbf{Ax} = \mathbf{b} \tag{2}$$

$$\mathbf{A}^{\mathrm{T}}\mathbf{x} = \mathbf{b} \tag{3}$$

and also provide for the update of (1) when a column of $\mathbf{A}$ is replaced by an n-vector $\mathbf{a}$. The original factorization uses Gaussian elimination with the pivoting strategy of Markowitz[9] to preserve sparsity. The stability of the factorization may be controlled by insisting that no pivot is less than a user-specified multiple of the largest element in its row (this multiple is typically set to 0·1, with smaller values preserving sparsity at the expense of stability and larger values preserving stability at the expense of sparsity). When a column of $\mathbf{A}$ is replaced by a vector $\mathbf{a}$, the Reid scheme first checks to see if the factorization (1) can be updated by permutations alone, without any genuine arithmetic. This ingenious algorithm thus reduces both the amount of round-off error and fill-in in the updated factorization.

Since the Reid algorithm has only a column replacement facility, and not a row replacement facility, we have chosen to factorize and update the transpose of the active constraint matrix. This imposes a negligible computational overhead. In the computation of the search direction for each iteration it is necessary to determine the kth column of $[\mathbf{D}^j]^{-1}$, which is denoted as $\mathbf{s}_k^j$. This may be achieved economically by solving

$$\mathbf{D}^j \mathbf{s}_k^j = \mathbf{e}_k$$

where $\mathbf{e}_k$ is the kth column of the identity matrix. This operation is of the general form of equation (2). Solving for $\boldsymbol{\lambda}^j$ in Step 1.1.0 is of the general form of equation (3).

THE STEEPEST EDGE ACTIVE SET ALGORITHM

During each iteration of Algorithm 1, the constraint to be deleted from the active set, and the corresponding search direction, are ascertained after solving the set of linear equations

$$[\mathbf{D}^j]^{\mathrm{T}} \boldsymbol{\lambda}^j = \mathbf{c}$$

where the quantities $\boldsymbol{\lambda}^j$ correspond to Lagrange multipliers. Choosing the search direction which has the largest positive multiplier is traditional in both the active set and simplex methods (Best and Ritter,[1] Dantzig[10]) and is often known as the Dantzig criterion. If a step of size σ is taken along a search direction $\mathbf{s}_i^j$, where $\mathbf{s}_i^j$ is the ith column of $[\mathbf{D}^j]^{-1}$, it follows that the objective function is reduced by an amount $\sigma\,\mathbf{c}^{\mathrm{T}}\mathbf{s}_i^j = \sigma\lambda_i^j$. Thus the Dantzig criterion may be interpreted as choosing the search direction which gives the greatest rate of decrease of the objective function per unit σ. An alternative strategy, which has proved most successful for the simplex method, is to select the search direction which gives the greatest rate of decrease of the objective function per unit distance, where 'distance' is measured along the search direction in n-dimensional space. For a step size σ, the distance moved along a search direction $\mathbf{s}_i^j$ is $\sigma\|\mathbf{s}_i^j\|_2$. Hence a search direction which gives the greatest rate of decrease of the objective function per unit distance may be selected using the rule

$$\frac{\lambda_k^j}{\|\mathbf{s}_k^j\|_2} = \max_{1\leqslant i\leqslant n}\left\{\frac{\lambda_i^j}{\|\mathbf{s}_i^j\|_2}\right\}$$

Since each search direction $\mathbf{s}_i^j$ corresponds to an edge of the feasible region, this criterion is commonly known as the steepest edge criterion. This procedure has been implemented for the revised simplex method by Goldfarb and Reid[2] and their results indicate that it often requires significantly fewer iterations than the Dantzig scheme in order to obtain an optimal solution. Approximations to the steepest edge simplex algorithm have been implemented by Harris[11] and Crowder and Hattingh,[12] both of whom also report improved convergence. In order for the steepest edge scheme to be effective, it is necessary to be able to estimate the Euclidean norms for each column of $[\mathbf{D}^j]^{-1}$ efficiently. Since computing these values afresh each time is prohibitively expensive, we follow the idea of Goldfarb and Reid[2] and develop relations which allow the norms to be recurred from iteration to iteration.

Let each column of $[\mathbf{D}^j]^{-1}$ be denoted by $\mathbf{s}_i^j$ such that $[\mathbf{D}^j]^{-1} = [\mathbf{s}_1^j, \mathbf{s}_2^j, \ldots, \mathbf{s}_n^j]$. Similarly, let the columns of the updated inverse $[\mathbf{D}^{j+1}]^{-1}$ be denoted by $\mathbf{s}_i^{j+1}$ such that $[\mathbf{D}^{j+1}]^{-1} = [\mathbf{s}_1^{j+1}, \mathbf{s}_2^{j+1}, \ldots, \mathbf{s}_n^{j+1}]$. Noting that $[\mathbf{D}^{j+1}]^{-1}$ is obtained from $[\mathbf{D}^j]^{-1}$ by replacing row k of $\mathbf{D}^j$ by row l of the constraint matrix, which is denoted as $\mathbf{a}_l^{\mathrm{T}}$, we have by the Sherman–Morrison[4] formulae

$$\mathbf{s}_k^{j+1} = \frac{\mathbf{s}_k^j}{\mathbf{a}_l^{\mathrm{T}}\mathbf{s}_k^j}$$

$$\mathbf{s}_i^{j+1} = \mathbf{s}_i^j - \left(\frac{\mathbf{a}_l^{\mathrm{T}}\mathbf{s}_i^j}{\mathbf{a}_l^{\mathrm{T}}\mathbf{s}_k^j}\right)\mathbf{s}_k^j; \quad i \neq k$$

Hence

$$\eta_k^{j+1} = \|\mathbf{s}_k^{j+1}\|_2^2 = \frac{\eta_k^j}{p^2}$$

$$\eta_i^{j+1} = \|\mathbf{s}_i^{j+1}\|_2^2 = \eta_i^j - (2/p)[(\mathbf{s}_i^j)^{\mathrm{T}}\mathbf{a}_l(\mathbf{s}_i^j)^{\mathrm{T}}\mathbf{s}_k^j] + [(\mathbf{s}_i^j)^{\mathrm{T}}\mathbf{a}_l]^2\eta_k^{j+1}; \quad i \neq k$$

where η^j and η^{j+1} denote the square of the Euclidean norm for each column of $[\mathbf{D}^j]^{-1}$ and $[\mathbf{D}^{j+1}]^{-1}$, respectively, and $p = \mathbf{a}_l^{\mathrm{T}}\mathbf{s}_k^j$ is the pivot. Using the relation $(\mathbf{s}_i^j)^{\mathrm{T}} = \mathbf{e}_i^{\mathrm{T}}[\mathbf{D}^j]^{-\mathrm{T}}$, where $\mathbf{e}_i$ is the ith column of the identity matrix, and defining the vectors $\mathbf{u}^j$ and $\mathbf{v}^j$ according to

$$\mathbf{u}^j = \frac{1}{p}[\mathbf{D}^j]^{-\mathrm{T}}\mathbf{s}_k^j$$

$$\mathbf{v}^j = [\mathbf{D}^j]^{-\mathrm{T}}\mathbf{a}_l$$

we obtain the required recurrence relations as

$$\eta_k^{j+1} = \frac{u_k^j}{p} \tag{4}$$

$$\eta_i^{j+1} = \eta_i^j + v_i^j(v_i^j \eta_k^{j+1} - 2u_i^j); \quad i \neq k \tag{5}$$

Equations (4) and (5) provide the basis for an efficient implementation of the steepest edge criterion, since it is necessary only to determine the vectors $\mathbf{u}^j$ and $\mathbf{v}^j$ ($\mathbf{s}_k^j$ is already known) in order to compute the squares of the norms for each column of $[\mathbf{D}^{j+1}]^{-1}$. Note that, in equation (4), η_k^{j+1} is computed afresh so as to reduce the effect of accumulated round-off error. Further economies can be achieved by storing and updating the Lagrange multipliers for each iteration, thus avoiding the need to solve for $\boldsymbol{\lambda}^j$ when computing the search direction. Using the Sherman–Morrison formulae, the updated Lagrange multipliers are given by

$$\lambda_k^{j+1} = \frac{\mathbf{c}^{\mathrm{T}} \mathbf{s}_k^j}{p} \tag{6}$$

$$\lambda_i^{j+1} = \lambda_i^j - v_i^j \lambda_k^{j+1}; \quad i \neq k \tag{7}$$

where again λ_k^{j+1} is computed afresh for each iteration. Using the recurrences (4), (5), (6) and (7) the steepest edge active set algorithm may be implemented as follows.

Algorithm 2 (Active set algorithm with steepest edge search)

Step 2.0—(Initialization)

2.0.0 Same as Step 1.0 but also enter with $\boldsymbol{\eta}^0$ where $\eta_i^0 = \|[\mathbf{D}^0]^{-1}\mathbf{e}_i\|_2^2$ for $i = 1, n$

2.0.1 Set $\boldsymbol{\lambda}^0 = [\mathbf{D}^0]^{-\mathrm{T}}\mathbf{c}$

Step 2.1—(Computation of search direction)

2.1.0 If $\mathbf{D}^j$ contains no artificial rows $(0 \notin I^j)$ go to Step 2.1.3

2.1.1 If $\lambda_i^j = 0$ for all i with $I_i^j = 0$ go to Step 2.1.3. Else determine the smallest index k such that

$$\frac{|\lambda_k^j|^2}{\eta_k^j} = \max_{\substack{1 \leq i \leq n \\ I_i^j = 0}} \left\{ \frac{|\lambda_i^j|^2}{\eta_i^j} \right\}$$

2.1.2 Set $\mathbf{s}_k^j = [\mathbf{D}^j]^{-1}\mathbf{e}_k$ if $\lambda_k^j > 0$, or $\mathbf{s}_k^j = -[\mathbf{D}^j]^{-1}\mathbf{e}_k$ if $\lambda_k^j < 0$ and go to Step 2.2

2.1.3 If $\lambda_i^j \leq 0$ for all i with $I_i^j > r$, exit with optimal solution $\mathbf{x}^j$. Else determine the smallest index k such that

$$\frac{(\lambda_k^j)^2}{\eta_k^j} = \max_{\substack{1 \leq i \leq n \\ I_i^j > r \\ \lambda_i^j > 0}} \left\{ \frac{(\lambda_i^j)^2}{\eta_i^j} \right\}$$

2.1.4 Set $\mathbf{s}_k^j = [\mathbf{D}^j]^{-1}\mathbf{e}_k$

Step 2.2—(Computation of maximum feasible step size)

Same as Step 1.2, but also store the pivot $p = \mathbf{a}_l^{\mathrm{T}}\mathbf{s}_k^j$

Step 2.3—(Update solution and active set data)

2.3.0 Step 1.3.0

2.3.1 Overwrite $\mathbf{s}_k^j$ according to $\mathbf{s}_k^j = \frac{1}{p}\mathbf{s}_k^j$

2.3.2 If $\lambda_k^j < 0$ replace p with $-p$

2.3.3 Set $\lambda_k^{j+1} = \mathbf{c}^T \mathbf{s}_k^j$

2.3.4 Overwrite $\mathbf{s}_k^j$ according to $\mathbf{s}_k^j = [\mathbf{D}^j]^{-T} \mathbf{s}_k^j$ and set $\eta_k^{j+1} = \max\{s_k^j/p, \varepsilon\}$, where ε is a small positive number

2.3.5 Set $\mathbf{v}^j = [\mathbf{D}^j]^{-T} \mathbf{a}_l$ and then compute

$$\eta_i^{j+1} = \max\{\eta_i^j + v_i^j(v_i^j \eta_k^{j+1} - 2s_i^j), \varepsilon\}$$
$$\lambda_i^{j+1} = \lambda_i^j - v_i^j \lambda_k^{j+1}$$

for $i = 1, n$ and $i \neq k$

2.3.6 Step 1.3.1

2.3.7 Step 1.3.2

2.3.8 Go to Step 2.1.

When computing the search direction for each iteration in Step 2.1, we choose to work with squared quantities to avoid taking square roots. In Steps 2.3.4 and 2.3.5, we ensure that the squared norm for each column of $[\mathbf{D}^{j+1}]^{-1}$ is always positive by insisting that $\eta_i^{j+1} \geqslant \varepsilon$ for $i = 1$, n, where ε is a small positive number. This precaution guards against the effects of accumulated round-off error in the recurred values for $\boldsymbol{\eta}$. In our implementation, all of the computations are conducted in double precision for a machine with a 32-bit word length. Provided that the constraint matrix is reasonably well-scaled and the active constraint matrix is periodically refactorized afresh, experience suggests that the accumulated round-off error is not serious (even for large problems requiring thousands of iterations) and, in general, $\eta_i^{j+1} \gg 0$ for $i = 1, n$. Nonetheless it is considered prudent to incorporate this safeguard.

To incorporate the steepest edge and Dantzig search schemes in a single computer program it is convenient to rearrange Algorithm 1 so that it has a similar form to Algorithm 2. This can done as follows.

Algorithm 3 (*Active set algorithm with Dantzig search*)

Step 3.0—(Initialization)
Same as Step 2.0 but no need to enter with $\boldsymbol{\eta}^0$

Step 3.1—(Computation of search direction)
Same as Step 1.1 but omit Step 1.1.0

Step 3.2—(Computation of maximum feasible step size)
Same as Step 2.2

Step 3.3—(Update solution and active set data)
Same as Step 2.3 but omit Steps 2.3.2, 2.3.4 and the update of $\boldsymbol{\eta}$ in Step 2.3.5.

Comparing Algorithm 2 with Algorithm 3, the main computational overhead associated with the steepest edge scheme is one additional equation solution per iteration. Algorithm 2 requires three equation solutions per iteration (one in the computation of the search direction and two in the update step) whilst Algorithm 3 requires two equation solutions per iteration (one in the computation of the search direction and one in the update step). Another additional computation associated with the steepest edge scheme is the calculation of the vector $\boldsymbol{\eta}^0$ in the initialization phase. For problems with sparse constraint matrices, the overhead associated with this step is likely to be slight. Experience indicates that it typically requires less than five per cent of the overall solution time.

TWO-PHASE ALGORITHM WITH CRASH PROCEDURE

In a similar fashion to the simplex method, the active set algorithm may be implemented in two phases. The first phase determines an initial feasible solution for the linear programming problem or reports that no feasible solution exists. The second phase uses the initial feasible solution from phase one and proceeds to compute the optimal solution, or reports that the problem is unbounded from below. Various two-phase strategies for the active set algorithm, together with FORTRAN code, may be found in Best and Ritter.[1] In this section we describe a two-phase algorithm which executes a simple crash procedure prior to phase one. The crash procedure has proved effective in reducing the number of phase one iterations, particularly if some equality constraints are present. In a similar manner to the schemes of Best and Ritter,[1] only one artificial variable is needed for phase one.

Consider the phase one problem of the form

$$\begin{aligned} &\text{Min} && \alpha \\ &\text{Subject to} && \mathbf{a}_i^{\mathrm{T}}\mathbf{x} = b_i; \quad i = 1, r \\ &&& \mathbf{a}_i^{\mathrm{T}}\mathbf{x} - \alpha \leqslant b_i; \quad i = r+1, r+m \\ &&& -\alpha \leqslant 0 \end{aligned} \tag{8}$$

which is a linear programming problem in $n+1$ variables, r equality constraints and $m+1$ inequality constraints. If an initial feasible solution can be found for (8), then the phase one problem may be solved directly using Algorithms 2 or 3 to yield an optimal solution $(\mathbf{x}^*, \alpha^*)$. The original linear programming problem has a feasible solution only if $\alpha^* = 0$. If this is the case, then an initial feasible solution for phase two may be extracted from $\mathbf{x}^*$.

An initial feasible solution for (8) is $\mathbf{x}^0$ which satisfies all of the equality constraints together with

$$\alpha^0 = \max_{r+1 \leqslant i \leqslant r+m} \{0, \mathbf{a}_i^{\mathrm{T}}\mathbf{x}^0 - b_i\}$$

Prior to solving (8) using Algorithms 2 or 3, each equality constraint $1, 2, \ldots, r$ must be in the initial index set I^0 provided it is linearly independent of all other equality constraints. Any active inequality constraint, for which $\mathbf{a}_i^{\mathrm{T}}\mathbf{x}^0 - \alpha^0 = b_i$ and $i = r+1, r+m+1$, may also be in I^0 provided that it is linearly independent of all other active constraints. Thus the initial index set is comprised of all linearly independent equalities, all linearly independent inequalities that are active at $(\mathbf{x}^0, \alpha^0)$, and $I^0 = \{I_1^0, I_2^0, \ldots I_{n+1}^0\}$, where $0 \leqslant I_i^0 \leqslant r+m+1$. The initial active constraint matrix is defined by $[\mathbf{D}^0]^{\mathrm{T}} = [\mathbf{d}_1^0, \mathbf{d}_2^0, \ldots, \mathbf{d}_{n+1}^0]$, where $\mathbf{d}_i^0$ corresponds to a row of the constraint matrix such that $\mathbf{d}_i^0 = \mathbf{a}_{I_i^0}$ (or an artificial row in which case $I_i^0 = 0$ and $\mathbf{d}_i^0$ is the ith column of the identity matrix). We now describe a complete two-phase algorithm, with a crash procedure, for solving linear programming problems with the steepest edge active set method.

Algorithm 4 (*Two-phase algorithm with crash procedure for steepest edge active set method*)

Step 4.0—(Initialize)

4.0.1 Expand constraint matrix and right hand side to $n+1$ variables and $r+m+1$ constraints. Set $a_{i,n+1} = -1$ for all inequality constraints $i = r+1, r+m$. Set $a_{r+m+1,n+1} = -1$ and $b_{r+m+1} = 0$

4.0.2 Set $I_i^0 = 0$ for $i = 1, n+1$ and $[\mathbf{D}^0]^{-1}$ equal to the $n+1 \times n+1$ identity matrix

Step 4.1—(Insert linearly independent equalities in active constraint matrix.) If $r = 0$ go to Step 4.2. Else do Steps 4.1.0 to 4.1.4 for $i = 1, r$

4.1.0 Set $\boldsymbol{\lambda} = [\mathbf{D}^0]^{-\mathrm{T}}\mathbf{a}_i$

4.1.1 Determine the smallest index k such that

$$|\lambda_k| = \max_{\substack{1 \leqslant i \leqslant n \\ I_i^0 = 0}} \{|\lambda_i|\}$$

4.1.2 If $\lambda_k = 0$, print message that equality constraint i is redundant and omit Steps 4.1.3–4.1.4

4.1.3 Update I^0 by settting $I_k^0 = i$

4.1.4 Update $[\mathbf{D}^0]^{-1}$ by replacing row k of $\mathbf{D}^0$ with $\mathbf{a}_i^{\mathrm{T}}$ or compute $[\mathbf{D}^0]^{-1}$ afresh using I^0 and the constraint matrix coefficients

Step 4.2—(Complete initial feasible solution)

4.2.0 If $r = 0$, set $x_i^0 = 0$ for $i = 1, n$,

$$x_{n+1}^0 = \max_{r+1 \leqslant i \leqslant r+m} \{0, -b_i\} \text{ and go to Step 4.3}$$

4.2.1 Set $x_i^0 = 0$ for $i = 1, n+1$ if $I_i^0 = 0$
Set $x_i^0 = b_{I_i^0}$ for $i = 1, n+1$ if $I_i^0 \neq 0$
Then overwrite $\mathbf{x}^0$ according to $\mathbf{x}^0 = [\mathbf{D}^0]^{-1}\mathbf{x}^0$ (Note that $x_{n+1}^0 = 0$)

4.2.2 Set $x_{n+1}^0 = \max_{r+1 \leqslant i \leqslant r+m} \{0, \mathbf{a}_i^{\mathrm{T}}\mathbf{x}^0 - b_i\}$

Step 4.3—(Insert active linearly independent inequalities in active constraint matrix.) Do Steps 4.3.0 to 4.3.6 for $i = r+1, r+m+1$

4.3.0 If $\mathbf{a}_i^{\mathrm{T}}\mathbf{x}^0 - b_i \neq 0$ constraint is inactive. Omit Steps 4.3.1–4.3.6

4.3.1 If no artificial rows are left (i.e. $0 \notin I^0$) go to Step 4.4

4.3.2 Set $\boldsymbol{\lambda} = [\mathbf{D}^0]^{-\mathrm{T}}\mathbf{a}_i$

4.3.3 Determine the smallest index k such that

$$|\lambda_k| = \max_{\substack{1 \leqslant i \leqslant n+1 \\ I_i^0 = 0}} \{|\lambda_i|\}$$

4.3.4 If $\lambda_k = 0$ inequality constraint is active but not linearly independent of other active constraints. Omit Steps 4.3.5 and 4.3.6

4.3.5 Update I^0 by setting $I_k^0 = i$

4.3.6 Update $[\mathbf{D}^0]^{-1}$ by replacing row k of $\mathbf{D}^0$ with $\mathbf{a}_i^{\mathrm{T}}$ or compute $[\mathbf{D}^0]^{-1}$ afresh using I^0 and constraint matrix coefficients

Step 4.4—(Form phase one objective function)
Set $c_i^0 = 0$ for $i = 1, n$ and $c_{n+1}^0 = 1$

Step 4.5—(Compute square of Euclidean norms for each column of $[\mathbf{D}^0]^{-1}$)
Set $\eta_i^0 = \|[\mathbf{D}^0]^{-1}\mathbf{e}_i\|_2^2$ for $i = 1, n+1$

Step 4.6—(Solve phase one linear programming problem)
Solve phase one problem using Algorithm 2 with $\mathbf{c}^0$, $\mathbf{x}^0$, I^0, $[\mathbf{D}^0]^{-1}$ and $\boldsymbol{\eta}^0$ as initial data. This problem has $n+1$ variables, r equality constraints and $m+1$ inequality constraints. Exit with optimal solution $\mathbf{x}^*$, I^*, $[\mathbf{D}^*]^{-1}$ and $\boldsymbol{\eta}^*$

Step 4.7—(Check for feasible solution)
If $x_{n+1}^* \neq 0$, print message that linear programming problem has no feasible solution and stop

Step 4.8—(Extract initial data for phase two)

4.8.0 If constraint $r+m+1 \in I^*$, find the index k such that $I_k^* = r+m+1$ and go to Step 4.8.6

4.8.1 Set $\mathbf{v}^* = [\mathbf{D}^*]^{-\mathrm{T}} \mathbf{a}_{r+m+1}$

4.8.2 Determine the smallest index k such that

$$|v_k^*| = \max_{\substack{1 \leq i \leq n+1 \\ I_i^* > r}} \{|v_i^*|\}$$

4.8.3 Set $\mathbf{s}^* = [\mathbf{D}^*]^{-1} \mathbf{e}_k$ and store the pivot $p = -s_{n+1}^*$

4.8.4 Overwrite $\mathbf{s}^*$ according to $\mathbf{s}^* = \frac{1}{p}[\mathbf{D}^*]^{-\mathrm{T}} \mathbf{s}^*$ and set $\eta_k^* = \max\{s_k^*/p, \varepsilon\}$

4.8.5 Compute $\eta_i^* = \max\{\eta_i^* + v_i^*(v_i^* \eta_k^* - 2s_i^*), \varepsilon\}$ for $i = 1, n+1$ with $i \neq k$

4.8.6 Set $I_k^* = I_{n+1}^*$ and establish initial index set for phase two by setting $I_i^0 = I_i^*$ for $i = 1, n$

4.8.7 Set $\eta_k^* = \eta_{n+1}^*$ and establish initial $\boldsymbol{\eta}$ vector for phase two by setting $\eta_i^0 = \eta_i^*$ for $i = 1, n$

4.8.8 Establish initial solution for phase two by setting $x_i^0 = x_i^*$ for $i = 1, n$

4.8.9 Obtain original constraint matrix and right hand side by deleting all coefficients in column $n+1$ and deleting row $r+m+1$

4.8.10 Establish the inverse of the active constraint matrix for phase two by computing the $n \times n$ matrix $[\mathbf{D}^0]^{-1}$ afresh using I^0 and the constraint matrix coefficients

Step 4.9—(Solve phase two linear programming problem)

Solve original problem using Algorithm 2 with $\mathbf{c}, \mathbf{x}^0, I^0, [\mathbf{D}^0]^{-1}$ and $\boldsymbol{\eta}^0$ as initial data. This problem has n variables, r equality constraints and m inequality constraints. Exit with optimal solution $\mathbf{x}^*, I^*, [\mathbf{D}^*]^{-1}$ and $\boldsymbol{\eta}^*$.

The above two-phase algorithm can be modified for use with the Dantzig search by omitting Steps 4.5, 4.8.3, 4.8.4, 4.8.5, 4.8.7 and employing Algorithm 3 in Steps 4.6 and 4.9.

To determine the linearly independent equality constraints in the two-phase procedure, we first set $I_i^0 = 0$ for $i = 1, n+1$ and initialize $[\mathbf{D}^0]^{-1}$ to the $(n+1) \times (n+1)$ identity matrix. We then solve the set of linear equations $\boldsymbol{\lambda} = [\mathbf{D}^0]^{-\mathrm{T}} \mathbf{a}_i$ for each equality $i = 1, r$, where $[\mathbf{D}^0]^{-1}$ is updated at each stage if a constraint is linearly independent. An equality is linearly independent only if $\lambda_k \neq 0$ where

$$|\lambda_k| = \max_{\substack{1 \leq i \leq n \\ I_i^0 = 0}} \{|\lambda_i|\}$$

and we note that $\lambda_{n+1} = 0$ for all equalities (since they have no entries in column $n+1$). The above test uses the result that if $\mathbf{d}_1, \ldots, \mathbf{d}_n$ are linearly independent n-vectors, and $\mathbf{d} = \lambda_1 \mathbf{d}_1 + \lambda_2 \mathbf{d}_2 + \ldots \lambda_n \mathbf{d}_n$, where $\mathbf{d}$ is also an n-vector, then $\mathbf{d}_1, \ldots, \mathbf{d}_{n-1}, \mathbf{d}$ are linearly independent only if $\lambda_n \neq 0$. If an equality is linearly independent, then we update $[\mathbf{D}^0]^{-1}$ by replacing row k of $\mathbf{D}^0$ (which is an artificial row) with $\mathbf{a}_i^{\mathrm{T}}$. If an equality is not linearly independent then it is redundant and may be deleted from the constraint matrix. A study of Algorithms 2 and 3, however, indicates that it is sufficient to ignore redundant equalities by not inserting them into the active set prior to phase one, since no equalities are considered as candidates to enter the active set in the optimization iterations. Once all the linearly independent equalities have been inserted in the active constraint matrix, a feasible solution for the phase one problem, defined by equation (8), is obtained by solving

$$\mathbf{x}^0 = [\mathbf{D}^0]^{-1}\mathbf{b}^0$$

where $b_i^0 = 0$ if $I_i^0 = 0$, $b_i^0 = b_{I_i^0}$ if $I_i^0 \neq 0$, and,

$$x_{n+1}^0 = \max_{\substack{r+1 \leqslant i \leqslant r+m \\ x_{n+1}^0 = 0}} \{0, \mathbf{a}_i^{\mathrm{T}}\mathbf{x}^0 - b_i\}$$

Once $\mathbf{x}^0$ has been computed, each active inequality (for which $\mathbf{a}_i^{\mathrm{T}}\mathbf{x}^0 - b_i = 0$ and $i = r+1$, $r+m+1$) is inserted into the active constraint matrix provided it is linearly independent of all other active constraints. The test for linear independence is identical to that used for the equality constraints. This completes the crash procedure.

After completing phase one and checking that a feasible solution exists for the original linear programming problem, we extract the initial data for phase two. Let $\mathbf{x}^*$, I^* and $\boldsymbol{\eta}^*$ be the optimal values generated by the solution of the phase one problem. If the constraint $r+m+1 \in I^*$, we set $I_k^* = I_{n+1}^*$ and $\eta_k^* = \eta_{n+1}^*$, where k is the index such that $I_k^* = r+m+1$, before obtaining the initial data for phase two by setting $x_i^0 = x_i^*$, $I_i^0 = I_i^*$ and $\eta_i^0 = \eta_i^*$ for $i = 1$ to n. This is equivalent to modifying $[\mathbf{D}^*]^{-1}$ by first swapping column k with column $n+1$ and then deleting the $(n+1)$th row and $(n+1)$th column. Since the $(n+1)$th row of $[\mathbf{D}^*]^{-1}$ is equal to $-\mathbf{e}_{n+1}^{\mathrm{T}}$ after the swap, deleting the last row and column must produce a new $n \times n$ inverse, $[\mathbf{D}^0]^{-1}$, which is non-singular. Moreover, the norms η_i^* for $i = 1$ to n are unchanged. If one or more of the constraints that are active at the optimal solution for phase one are linearly dependent, then the constraint $r+m+1$ may not be in the index set I^*. In this case, we find a row k of $\mathbf{D}^*$ which can be replaced by the constraint $r+m+1$ without making the active constraint matrix singular. Since none of the equality constraints has an entry in column $n+1$ of the overall constraint matrix, there must be at least one inequality which can be replaced. Once the row k is found, we update the $\boldsymbol{\eta}^*$ vector by replacing row k with the constraint $r+m+1$. We then proceed as before by setting $I_k^* = I_{n+1}^*$ and $\eta_k^* = \eta_{n+1}^*$ and extract the initial data for phase two by setting $x_i^0 = x_i^*$, $I_i^0 = I_i^*$ and $\eta_i^0 = \eta_i^*$ for $i = 1$ to n. After deleting the coefficients that were added to the constraint matrix for phase one, the $n \times n$ matrix $[\mathbf{D}^0]^{-1}$ is computed afresh using the index set I^0 and the constraint matrix coefficients. The original linear programming problem is then solved using Algorithm 2 with $\mathbf{x}^0$, I^0, $[\mathbf{D}^0]^{-1}$ and $\boldsymbol{\eta}^0$ as initial data.

As mentioned in an earlier section, we have chosen the sparse LU decomposition algorithm described by Reid[7,8] to factorize and update the active constraint matrix. Since Reid's procedure does not have a row replacement facility, we factorize the transpose of the active constraint matrix and perform updates by replacing columns instead of rows. This imposes a negligible computational overhead. In the update steps of Algorithms 2 and 3, and in the crash procedure prior to phase one of Algorithm 4, it is necessary to periodically refactorize the active constraint matrix afresh using the current index set and the constraint matrix coefficients. After a number of updates of the original factorization, round-off error may accumulate and cause the factorization to become inaccurate. Provided that the constraint matrix is reasonably well-scaled, experience suggests that problems due to round-off error are rare when Reid's algorithm is used with the pivot selection parameter set to 0·1 (see Reid[8]). It is, however, necessary to periodically compute the factorization afresh in any case due to the steady build up of non-zeros in the product form of the LU decomposition. Various strategies for choosing when to refactorize with the simplex method are discussed by Reid.[7] With reference to equation (1), let N be the number of equations, LENU be the number of non-zeros held in $\mathbf{U}$ and LENL be the number of non-zeros held in $\mathbf{L}$. Computational experiments indicate that the time required for a single solution of equations (2) or

(3) is roughly proportional to $N + \text{LENU} + \text{LENL}$. In our implementation of Algorithms 2, 3 and 4, we choose to refactorize when

$$N + \text{LENU}^j + \text{LENL}^j > 1{\cdot}5(N + \text{LENU}^0 + \text{LENL}^0)$$

where $(\;)^j$ denotes values for the current iteration and $(\;)^0$ denotes values after the last refactorization. Thus we refactorize when the estimated time for a solution of equations (2) or (3) increases by roughly 50 per cent since the last refactorization. The overall performance of the algorithms is not especially sensitive to the exact refactorization point and the factor of 1·5 may be replaced by a value anywhere in the range 1·4–2. It may occasionally be necessary to refactorize earlier than the above rule dictates if the factorization has become inaccurate due to accumulated round-off error. This is uncommon, but various tests for checking the accuracy of the factorization are discussed in Reid.[8] In our implementation, we refactorize immediately if the magnitude of the largest entry in $\mathbf{U}$ exceeds 10^8.

DEGENERACY

Many of the linear programming problems that occur in practice have feasible regions with degenerate vertices. A degenerate vertex is one where the number of active constraints is greater than the number of variables, and implies that some of the active constraints are not linearly independent. Such a vertex is indicated in the active set algorithm when $I_k^j \neq 0$ and $\sigma^j = 0$ prior to the update step, and results in no change in the solution or the objective function for an iteration. It is theoretically possible for the index set I^j, and hence the active constraint matrix $\mathbf{D}^j$, to be repeated after a number of consecutive degenerate iterations. This phenomenon, which is known as cycling, results in the algorithm becoming trapped at a vertex and finite termination is no longer assured. A complete theoretical discussion of cycling in the active set method may be found in Best and Ritter,[1] who also suggest a technique for its resolution using the rules of Bland.[13] Although cycling is rarely observed in practice, due to the effect of round-off error, multiply degenerate problems may result in a large number of wasted iterations and it is sometimes beneficial to resolve degeneracy using a heuristic scheme. One technique for resolving degeneracy, which has proved successful for the simplex method, is due to Harris.[11] The basic idea behind the Harris algorithm is to broaden the selection of potential pivots by relaxing each of the inequality constraints by a specified feasibility tolerance. Of all the inequalities not in the active set which satisfy the relaxed feasibility tolerance, the one which has the largest pivot is chosen to enter the active set. The Harris scheme may be implemented in Algorithm 2/3 by modifying Step 2.2/3.2 as follows.

(*Computation of maximum feasible step size with Harris pivot selection*)

If $\mathbf{a}_i^{\mathrm{T}}\mathbf{s}_k^j \geqslant 0$ for all $i = r+1, \ldots, r+m$ and $i \notin I^j$, print message that the problem is unbounded from below and stop. Else compute the limit L such that

$$L = \min_{\substack{r+1 \leqslant i \leqslant r+m \\ i \notin I^j \\ \mathbf{a}_i^{\mathrm{T}}\mathbf{s}_k^j < 0 \\ \mathbf{a}_i^{\mathrm{T}}\mathbf{x}^j - b_i \leqslant \delta}} \left\{ \frac{\mathbf{a}_i^{\mathrm{T}}\mathbf{x}^j - b_i - \delta}{\mathbf{a}_i^{\mathrm{T}}\mathbf{s}_k^j} \right\}$$

where δ is a specified feasibility tolerance. Then compute the smallest index l and σ^j according to

$$\sigma^j = \frac{\mathbf{a}_l^{\mathrm{T}}\mathbf{x}^j - b_l}{\mathbf{a}_l^{\mathrm{T}}\mathbf{s}_k^j}$$

where $\sigma^j \leqslant L$ and

$$\mathbf{a}_l^T \mathbf{s}_k^j = \min_{\substack{r+1 \leqslant i \leqslant r+m \\ i \notin I^j \\ \mathbf{a}_i^T \mathbf{s}_k^j < 0 \\ \mathbf{a}_i^T \mathbf{x}^j - b_i \leqslant \delta}} \{\mathbf{a}_i^T \mathbf{s}_k^j\}$$

With careful coding, it is possible to implement the above scheme with only a single sweep through the inequalities for each iteration. Computational experience with the scheme suggests that it often reduces the number of iterations substantially for problems which contain many degenerate vertices. We have chosen not to implement it, however, since the saving in iterations is usually counteracted by the increased computation time required. This conclusion is, of course, problem dependent and the topic is worthy of further investigation.

APPLICATIONS

In order to compare the performance of the steepest edge and Dantzig search criteria, Algorithms 2 and 3 have been implemented in a single computer program and applied to a number of sparse linear programming problems. Both Algorithms were used with the two-phase procedure of Algorithm 4. The problems themselves arise from the application of classical plasticity theory to compute upper and lower bounds on the limit loads for two-dimensional continua (Bottero *et al.*,[14] Sloan[15]). A summary of the problems considered is shown in Table I. They range in size from medium to large, and all have very sparse constraint matrices which are reasonably well scaled.

The performance of the proposed active set algorithms on the test problems is summarized in Table II. For all of the test problems considered, the steepest edge scheme required fewer iterations than the Dantzig scheme. The average reduction in total iterations is 53 per cent, with the largest reduction of 68 per cent occurring for problem 6 and the smallest reduction, of 33 per cent, occurring for problem 1. Owing to the computational overhead associated with the steepest edge algorithm, the savings in CPU time are less dramatic, but nonetheless substantial. The average saving in CPU time is 36 per cent, with the largest saving occurring for problem 4 (55 per cent) and the smallest saving occurring for problem 1 (6 per cent). In general, the savings in both iterations and CPU time are more pronounced for the larger problems (4, 5 and 6), thus emphasizing the superiority of the steepest edge search heuristic over the Dantzig search heuristic.

The crash procedure incorporated in Algorithm 4 appears to be particularly effective in minimizing the number of phase one iterations for both the steepest edge and Dantzig algorithms. The average proportion of the total iterations required by phase one for the Dantzig scheme is approximately 5 per cent, with the largest proportion occurring for problem 2 (13 per cent) and

Table I. Test problems

Problem	Number of variables n	Number of equalities r	Number of inequalities m	Total number of constraints $r+m$	Non-zeros in constraint matrix
1	330	128	1157	1285	3897
2	330	128	2309	2437	7363
3	531	456	4248	4704	14196
4	820	302	6056	6358	19240
5	1669	600	6348	6948	21241
6	1152	1026	9216	10242	30870

Table II. Results for test problems

Problem	Dantzig search (Algorithm 3) Phase one iterations	Phase two iterations	Total iterations	Time (sec)	Steepest edge search (Algorithm 2) Phase one iterations	Phase two iterations	Total iterations	Time (sec)
1	21	460	481	19·2	14	310	324	18·0
2	252	1617	1869	108	283	747	1030	77
3	65	1205	1270	176	53	632	685	123
4	129	5001	5130	730	25	1823	1848	327
5	103	6703	6806	1493	75	2552	2627	813
6	0	1554	1554	497	0	496	496	255

Notes: All times are in CPU-seconds for VAX 8550 operating under VMS with the optimizing FORTRAN 77 compiler
All problems solved incore with negligible page faulting

the smallest proportion occurring for problem 6 (no phase one iterations required). Similar trends are evident in the results for the steepest edge algorithm where the average proportion is approximately 7 per cent, with the largest proportion of 28 per cent occurring for problem 2. The proportion of iterations required by phase one is, of course, dependent on the type of problem. Computational experience with Algorithm 4, however, suggests that it is rare for phase one to require more than 30 per cent of the total iterations when large sparse problems are considered.

CONCLUSIONS

Overall, it would appear that the active set algorithm, with a steepest edge search, is an effective means of solving large linear programming problems where the constraint matrix is sparse and has more rows than columns. The steepest edge search is substantially more efficient than the Dantzig search, particularly for large problems, and is simple to implement. The active set algorithm can, of course, be applied to problems which have constraint matrices with more columns than rows by invoking duality theory. It would be interesting to compare the steepest edge active set and steepest edge simplex methods by applying both algorithms to the same set of test problems (and employing duality theory were appropriate).

ACKNOWLEDGEMENTS

The author would like to thank Dr. B. J. Williams and Dr. G. Kuczera for helpful discussions on linear programming strategies. He would also like to thank Ms Cherie Hook for typing the manuscript.

REFERENCES

1. M. J. Best and K. Ritter, *Linear Programming: Active Set Analysis and Computer Programs*, Prentice-Hall, Englewood Cliffs, New Jersey, 1985.
2. D. Goldfarb and J. K. Reid, 'A practicable steepest-edge simplex algorithm', *Math. Programming*, **12,** 361–371 (1977).
3. P. E. Gill, W. Murray and M. H. Wright, *Practical Optimisation*, Academic Press, New York, 1981.
4. J. Sherman and W. J. Morrison, 'Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row', *Annals Math. Statistics*, **20,** 621 (1949).
5. J. J. H. Forrest and J. A. Tomlin, 'Updating triangular factors of the basis to maintain sparsity in the product form simplex method', *Math. Programming*, **2,** 263–278 (1972).
6. R. H. Bartels, 'A stabilization of the simplex method', *Numer. Mathematik*, **16,** 414–434 (1971).

7. J. K. Reid, 'A sparsity exploiting variant of the Bartels–Golub decomposition for linear programming bases', *Math. Programming*, **24**, 55–69 (1982).
8. J. K. Reid, 'Fortran subroutines for handling sparse linear programming bases', *Harwell Report No. R8269*, England, 1976.
9. H. M. Markowitz, 'The elimination form of the inverse and its applications to linear programming', *Manage. Sci.*, **3**, 255–269 (1957).
10. G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey, 1963.
11. P. M. J. Harris, 'Pivot selection methods of the Devex LP code', *Math. Programming*, **5**, 1–28 (1973).
12. H. Crowder and J. M. Hattingh, 'Partially normalised pivot selection methods in linear programming', *Mathematical Programming Study 4*, 1975, pp. 12–25.
13. R. G. Bland, 'New finite pivoting rules for the Simplex method', *Math. Operations Res.*, **2**, 103–107 (1977).
14. A. Bottero, R. Negre, J. Pastor and S. Turgeman, 'Finite element method and limit analysis theory for soil mechanics problems', *Comp. Methods Appl. Mech. Eng.*, **22**, 131–149 (1980).
15. S. W. Sloan, 'Lower bound limit analysis using finite elements and linear programming', *Int. j. numer. anal. methods geomech.*, **12**, 61–77 (1988).