SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305-4022

A Strictly Improving Linear Programming Algorithm Based on a Series of Phase I Problems

by

S. A. Leichner, G. B. Dantzig and J. W. Davis

TECHNICAL REPORT SOL 92-2

April 1992

Research and reproduction of this report were partially supported by the National Science Foundation Grants ECS-8906260, DMS-8913089; the Department of Energy Grant DE-FG03-92ER25116; Office of Naval Research Grant N00014-89-J-1659; the Electric Power Research Institute Contracts RP 8010-09 at Stanford University, and by Hewlett Packard Laboratories.

A Strictly Improving Linear Programming Algorithm Based on a Series of Phase I Problems

S. A. Leichner G. B. Dantzig J. W. Davis

3 April 1992

Abstract

When used on degenerate problems, the simplex method often takes a number of degenerate steps at a particular vertex before moving to the next. In theory (although rarely in practice), the simplex method can actually cycle at such a degenerate point. Instead of trying to modify the simplex method to avoid degenerate steps, we have developed a new linear programming algorithm that is completely impervious to degeneracy.

This new method solves the Phase II problem of finding an optimal solution by solving a series of Phase I feasibility problems. Strict improvement is attained at each iteration in the Phase I algorithm, and the Phase II sequence of feasibility problems has linear convergence in the number of Phase I problems.

When tested on the 30 smallest *NETLIB* linear programming test problems, the computational results for the new Phase II algorithm (using the series of feasibility problems) were over 15% faster than the simplex method; on some problems, it was almost two times faster, and on one problem it was four times faster.

1 Introduction

On highly degenerate problems, the simplex method often "stalls", performing a number of iterations at a degenerate point before producing any improvement in the objective value. Examples have been constructed by Hoffman [8] and Beale [1] to show that it is theoretically possible that the iterative steps can repeat and thus cycle forever, although this phenomenon is quite rare in practice. Instead of trying to make the simplex more efficient by trying to avoid stalls due to degeneracy (or near degeneracy), we develop a new linear programming algorithm that is completely impervious to degeneracy.

This new method is based on the solution of a series of Phase I feasibility problems, where the feasibility problems are solved by a Phase I algorithm involving the use of least-squares subproblems in column selection. This least-squares Phase I algorithm has the property of strict improvement at each iteration, even if every basic solution (in the simplex method sense) is degenerate. Like the simplex method, the new Phase I algorithm terminates in a finite number of steps, and in practice, this number is often quite low compared to the simplex method. In addition, the sequence of Phase I problems solved in Phase II is shown to have linear convergence to the optimal solution.

Section 2 gives an overview and pseudocode for the least-squares Phase I algorithm. More details on this algorithm can be found in a companion paper [10].

Section 3 extends the algorithm presented in Section 2 to produce a Phase II algorithm to solve a linear program. This Phase II algorithm solves a series of augmented feasibility problems, each solved by the least-squares Phase I algorithm from Section 2. The Phase II algorithm relies heavily on the theorem of the alternative in order to generate the sequence of Phase I problems. Convergence to the optimal solution is proved, and it is shown that a minor variation of the algorithm has linear convergence in the number of Phase I problems solved. Finally, there is a discussion of some practical implementation issues.

Section 4 presents computational results for the algorithms developed in Section 3. As noted, the Phase II algorithm has good performance, with run times 15% times faster than LSSOL's implementation of the simplex method [6]. On some problems, the Phase II algorithm was over 2 times faster.

Section 5 summarizes the work presented here and attempts to draw some conclusions. Suggestions for future work appear at the end of this section.

2 The Least-Squares Phase I Algorithm

2.1 The Problem

The least-squares Phase I algorithm indirectly solves the problem

$$\begin{aligned} Ax &= b, \\ x &\geq 0, \end{aligned} \tag{2-1}$$

by solving the problem

$$\begin{aligned} Bx_B &= b, \\ x_B &> 0, \end{aligned} \tag{2-2}$$

where $x \in \Re^n$, $b \in \Re^m$, $A_j \neq 0 \; \forall \; j$, $A \in \Re^{m \times n}$, and B is a linearly independent subset of the columns of A. We assume that $b \neq 0$, or else $x = 0$ is a trivial solution. We also assume that b has been rescaled so that $\| b \|_2 = 1$, and similarly the columns of A have been rescaled so that $\| A_j \|_2 = 1 \; \forall \; j$, where A_j denotes the jth column of the matrix A. Once we have a solution x_B to (2-2), it is a trivial matter to construct an x from x_B such that x is a solution to (2-1).

2.2 The Algorithm

The following material is intended to be an overview of the least-squares Phase I algorithm. For more details, see [10].

The main objective of the least-squares Phase I algorithm to be presented here is to build up such a matrix B from linearly independent columns of A, and to find positive weightings of these columns, x_B, such that B and x_B comprise a solution to (2-2). This will in turn give us a solution to (2-1). In the remainder of this discussion, we will refer to this matrix B as a *basis*. Note that this notion of a basis differs from that of the simplex method, as our basis B may contain less than m columns and that these columns may not be sufficient in themselves to span the column space of A.

At the start of an iteration, we are given a basis B, composed of a linearly independent subset of the columns of A, with the property that the least-squares weighting of these columns which yields the closest approximation to the right-hand side b is a positive weighting. The iterative step tests whether the current approximation is the closest approximation with positive a weighting, and if not, selects an incoming column to augment the basis. This yields a positive combination of the previous approximation and the incoming column which is a strictly better approximation to the right-hand side. There exists, however, a (possibly proper) subset of the columns of the augmented basis such that the least-squares weighting is positive, and also a better approximation to the right-hand side than the positive combination. This subset of the augmented set of basic columns can be used to start the next iteration. We will use the following notation:

B: a basis made up of linearly independent columns of A.

$x_B > 0$: positive weightings of the columns of B.

$v = Bx_B$: a basic weighting and approximation to the right-hand side b.

$u = b - v$: the current residual.

In the first step, when no columns are yet in B, we have $v = 0$, and the least-squares column-selection subproblem degenerates to finding

$$\min_j \left(\min_{\beta > 0} \| b - \beta A_j \|^2 \right).$$

This is equivalent to selecting the entering column to be A_s where

$$s = \operatorname*{argmax}_{j : b^T A_j > 0} b^T A_j .$$

If we cannot find an incoming column at this point, we know that the problem is infeasible, and that $v = Ax$ where $x = 0$ is the closest approximation to b using non-negative weightings of the columns of A. However, if (2-1) is feasible, we know that we can find such an A_s. Thus our first approximation to b is $v = (b^T A_s)A_s$, and it is as close to b as is possible using a positive weighting of a single column of A.

In the general step, the approximation v is not 0, so we minimize the residual of the following two-variable least-squares subproblem:

$$\min_{(\alpha,\beta)>0} \| b - \alpha v - \beta A_j \|^2.$$

That is, the entering column is A_s, where A_s minimizes the above expression for all $A_j \notin B$. Thus we are selecting that column A_s whose nonnegative combination with v brings us closer to b than any other A_j, $j \neq s$. The index s is determined by the expression

$$s = \operatorname*{argmax}_{j:A_j^T u>0} \left(\frac{A_j^T u}{\left(v^T v - (A_j^T v)^2\right)^{1/2}} \right). \tag{2-3}$$

If we cannot find an incoming column at this point, we know that the problem is infeasible, and that $v = Bx_B$ is the closest approximation to b using nonnegative weightings of the columns of A. However as before, we know that we can find such an A_s if (2-1) is feasible. Once we have added A_s to B, we need find the (possibly proper) subset of the columns of the augmented basis (that we mentioned earlier) so that it can be used to start the next iteration.

Let B be the basis matrix before adding A_s, and let x_B correspond to B. Let $\tilde{B} = (\,B \quad A_s\,)$ and let $\tilde{x}_B$ be the updated x_B corresponding to $\tilde{B}$. To find $\tilde{x}_B$, we first need to solve the least-squares problem $\min \| b - \tilde{B}y \|$, and to set $c = (\,x_B^T \quad 0)^T$.

Case 1: $y > 0$
Let $\tilde{x}_B = y$.

Case 2: $y \geq 0$
Remove all columns from $\tilde{B}$ that correspond to zero elements in y. Let $\tilde{x}_B$ consist of all the nonzero elements of y.

Case 3: We do not have $y > 0$ or $y \geq 0$.
In this case, at least one component of y is negative. We form the convex combination $c = \lambda c + (1 - \lambda)y$ where

$$\lambda = \max_{y_i<0} \frac{-y_i}{c_i - y_i}.$$

This gives us $c \geq 0$. We remove all columns from $\tilde{B}$ and elements from x_B that correspond to zero elements in c. Then we go back and solve $\min \| b - \tilde{B}y \|$ again to get a new y. We repeat this until we find a $y \geq 0$.

Thus at each iteration, we choose a column from A to enter B. Then we solve the least-squares problem $\min \| b - \tilde{B}y \|$. We find a new B and x_B using y as just described, possibly dropping one or more columns from $\tilde{B}$. Each new approximation $\tilde{v} = \tilde{B}\tilde{x}_B$ is strictly closer to b than the previous $v = Bx_B$. We are done as soon as $Bx_B = b$ in some iteration.

The least-squares Phase I algorithm has the following desirable properties:

1. The approximation $v = Bx_B$ is strictly closer to b at each iteration. This means that degeneracy does not cause problems; cycling cannot occur, and the process terminates in a finite number of steps.

2. If (2-1) is infeasible, this will be detected when the current approximation $v = Bx_B$ is closer to b than is any other nonnegative weighting of the columns of A.

3. Whenever a new column is chosen from A to enter B, it is linearly independent of the columns currently in B. Thus the solution $\min \| b - \tilde{B}y \|$ is always unique, so numerical difficulties aside, rank deficient least-squares problems are never encountered.

This algorithm is quite similar to a number of existing algorithms, including one to solve the bounded least-squares problem (found in [3]), and another to solve the non-negative least-squares problem (found in [9]). In addition, our algorithm is also closely related to the algorithm developed by Dantzig [4] and by Van de Panne & Whinston [12].

Pseudocode

Notation and Variables:

A: The original matrix in (2-1).
b: The original right-hand side in (2-1).
B: The matrix of currently basic columns.
x_B: The current positive weightings of the columns of B.
v: The current approximation Bx_B.
u: The current residual $b - v$.
y: The current least-squares solution of $\min \| b - By \|^2$.
X: The vector to hold the solution to (2-1).

0. Initialization {no columns initially in basis}
   $B := \emptyset;\quad x_B := \emptyset;\quad u := b;\quad v := 0;\quad ;X := 0;$
1. Startup {find the first column to place in B}
   If $b^T A_j \leq 0 \;\; \forall\, j = 1, \ldots, n$ then go to 4. {the problem is infeasible}
   $s := \operatorname{argmax}_{j : b^T A_j > 0} b^T A_j$
   Place A_s in B; let $y = b^T A_s$.
2. Main Loop {Add columns to B until $u = 0$ or infeasibility discovered.}
   $x_B := y;\quad v := Bx_B$ {set current approximation}
   $u := b - v$ {set current residual}

If $u = 0$ then go to 4. {solution found}
If $u^T A_j \leq 0 \ \forall j$ then go to 4. {infeasibility discovered}

$$s = \operatorname*{argmax}_{\substack{j \notin BI \\ j : A_j^T u > 0}} \frac{A_j^T u}{\left(v^T v - (A_j^T v)^2\right)^{1/2}}$$

Place A_s in B; set $c = (x_B \ 0)^T$.

3. Least-Squares Loop
   Solve $\| By - b \|^2$ for y.
   If $y > 0$ then go to 2. {new B and x_B found}
   If $y \geq 0$ then remove all columns from B and elements
      from x_B that correspond to zero components of y.
      Go to 2. {new B and x_B found}
   $\lambda := \max_{y_i < 0}(-y_i/(c_i - y_i))$
   $c := \lambda c + (1 - \lambda)y$ {form convex combination of c and y }
      Remove all columns from B and elements from x_B and c that
      correspond to zero components of c.
   Go to 3. {repeat Least-Squares loop}
4. Report on feasibility and output the solution found.

A number of variations of the least-squares Phase I algorithm were developed in [10]. These included:

- Free variables were permitted in the problem without requiring the problem to be explicitly converted to the form of (2-1).
- The column selection rule denominator in (2-3) was replaced by "1" to reduce computation.
- A crash basis (made of columns primarily corresponding to slack variables) was used as the initial basis B.

When tested on the 30 smallest linear programming test problems available from *NETLIB* [5], the least-squares Phase I algorithm had excellent performance, with run times almost 3.5 times faster than LSSOL's implementation of the simplex method [6]. On some problems, the least-squares Phase I algorithm was over 10 times faster. See [10] for more details.

3 The Phase II Algorithm

3.1 The Problem

The Phase II algorithm addresses the complete linear programming problem. That is,

$$\begin{aligned} \min\ & c^T x \\ \text{s.t. } Ax &= b \\ x &\geq 0, \end{aligned} \tag{3-1}$$

where $x, c \in \Re^n$, $b \in \Re^m$ and $A \in \Re^{m\times n}$. As before, we assume that b has been rescaled so that $\| b \| = 1$, and similarly the columns of A have been rescaled so that $\| A_j \| = 1 \ \forall\ j$.

The simplex method solves this problem in two stages, commonly called Phase I and Phase II. In Phase I, a feasible solution is found, and in Phase II, an optimal solution is found, starting from the feasible solution found in Phase I. In contrast, we will develop an algorithm to solve this problem based on solving a sequence of Phase I problems, each solved by the least-squares Phase I algorithm presented in Section 2.

3.2 Theorem of the Alternative

The series of Phase I problems we are going to solve are of the form

$$\begin{aligned} c^T x &\leq z_k \\ Ax &= b \\ x &\geq 0, \end{aligned} \tag{3-2}$$

where z^* is the optimal objective value of (3-1), and

$$\begin{aligned} z_0 \leq z_{k+1} &\leq z^* \\ z_{k+1} &> z_k. \end{aligned}$$

We will discuss the selection of z_0 later, as well as the formulation of z_{k+1} from z_k. We will first consider some properties of the augmented primal (3-2). Farkas' Lemma states that either the augmented primal system (3-2) or the system

$$\begin{aligned} \sigma c^T + \pi^T A &\leq 0 \\ \sigma z_k + \pi^T b &> 0 \\ \sigma &\leq 0 \end{aligned} \tag{3-3}$$

has a solution.

Theorem 3.1
If we attempt to solve (3-2) using the least-squares Phase I algorithm from Section 2 and find that it is infeasible, then the residual u is not zero, and u is a feasible solution to (3-3).

Proof: We know from Section 2 that if (3-2) is infeasible, then $u \neq 0$. We will now show that u is a feasible solution to (3-3). Let $u = (u_0, \tilde{u}) = (u_1, \ldots, u_m)$

Part 1: We can rewrite the augmented primal problem as

$$\begin{aligned} \begin{pmatrix} c^T & 1 \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} &= \begin{pmatrix} z_k \\ b \end{pmatrix} \\ \begin{pmatrix} x \\ y \end{pmatrix} &\geq 0. \end{aligned}$$

When the least-squares Phase I algorithm terminates with infeasibility of the augmented primal, we know that u is orthogonal to all basic columns. That is,

$$u^T \begin{pmatrix} c^T & 1 \\ A & 0 \end{pmatrix}_i = 0 \quad \forall\, i,$$

where $i \in BI$ and BI is the index set of basic columns. For all nonbasic columns, we know

$$u^T \begin{pmatrix} c^T & 1 \\ A & 0 \end{pmatrix}_j \leq 0,$$

where $j \notin BI$. Thus

$$u^T \begin{pmatrix} c^T & 1 \\ A & 0 \end{pmatrix} \leq 0.$$

This can be rewritten as the system

$$\begin{aligned} u_0 c^T + \tilde{u}^T A &\leq 0 \\ u_0 &\leq 0. \end{aligned}$$

Part 2: Next we show

$$u_0 z_k + \tilde{u}^T b > 0. \tag{3-4}$$

We have $u \neq 0$, so we have

$$u^T u = u^T \begin{pmatrix} z_k \\ b \end{pmatrix} > 0.$$

□

Theorem 3.2

If the augmented primal problem (3-2) is found to be infeasible by the least-squares Phase I algorithm and $u_0 = 0$, then the original primal problem (3-1) is infeasible.

Proof: From Theorem 3.1 we know that $u = (u_0, \tilde{u})$ is a solution to (3-3). If $u_0 = 0$, then $\tilde{u}$ is a solution to the smaller system

$$\begin{aligned} \tilde{u}^T A &\leq 0 \\ \tilde{u}^T b &> 0, \end{aligned}$$

which implies (by Farkas' Lemma) that there is no solution to the system (2-1).
□

3.3 Iterating Towards the Minimum

If we solve the augmented primal problem and find that it is infeasible, then we use the residual $u = (u_0, \tilde{u})$ to replace z_k by z_{k+1} as follows:

$$z_{k+1} = -\frac{\tilde{u}^T b}{u_0}. \tag{3-5}$$

Note that when we solve the augmented primal problem (3-2) again using z_{k+1} instead of z_k, the new residual will still be called u, but will correspond to z_{k+1}.

Theorem 3.3

$$z_{k+1} > z_k \quad \forall\ k.$$

Proof: We assume that we earlier have solved the Phase I problem and (3-1) is feasible. This implies that when we solve (3-2), it is infeasible and $u_0 < 0$. Therefore, noting (3-4), we can write

$$z_k < \frac{-\tilde{u}^T b}{u_0} = z_{k+1}.$$

$\square$

Theorem 3.4
If (3-1) has a finite optimal objective value z^*, then $z_{k+1} \le z^*\ \forall\ k$.

Proof: Consider the primal problem and its dual.

$$\begin{array}{llll}
 & \text{Primal} & & \\
\min & c^T x & & \\
\text{s.t.} & Ax & = & b \\
 & x & \geq & 0
\end{array}
\qquad\qquad
\begin{array}{llll}
 & \text{Dual} & & \\
\max & \pi^T b & & \\
\text{s.t.} & \pi^T A & \leq & c^T
\end{array}$$

Note that if $u_0 = 0$, then (3-1) is infeasible and we do not form z_{k+1}, so we can assume that $u_0 < 0$. We will now show that $\pi = -\tilde{u}/u_0$ is feasible in the dual.

$$\begin{aligned}
u_0 c^T + \tilde{u}^T A &\leq 0 \qquad \text{from Theorem 3.1} \\
-\frac{\tilde{u}^T}{u_0} A &\leq c^T.
\end{aligned}$$

Thus π is feasible in the dual, and the objective value of the dual at π is

$$-\frac{\tilde{u}^T b}{u_0} = z_{k+1},$$

and must be less than or equal to z^*. $\square$

We have proved that $z_k < z_{k+1} \le z^*\ \forall\ k$. Assuming that we can find a $z_0 < z^*$, this defines a monotonically increasing sequence bounded above by z^*. This sequence will converge to a unique limit point, and we will prove that this limit point is z^*. In order to prove this, we need the following lemma.

Lemma 3.5
Let $\tilde{z}$ be a real number and let τ be positive. Let f be continuous on the interval $[\tilde{z}-\tau,\tilde{z}]$. If $f(z) > z \ \forall \ z \in [\tilde{z}-\tau,\tilde{z}]$, then $\exists \ y \in [\tilde{z}-\tau,\tilde{z})$, such that $f(z) > \tilde{z} \ \forall \ z \in [y,\tilde{z}]$.

Proof: This is an elementary consequence of continuity, and the fact that $f(z) > z \ \forall \ z \in [\tilde{z}-\tau,\tilde{z}]$. □

Theorem 3.6
The sequence of iterates $\{z_k\}$ produced by (3-5) converges to z^*.

Proof (by contradiction):
We know $\{z_k\}$ converges to a limit point $\tilde{z}$ where $z_0 < \tilde{z} \le z^*$. Assume $\tilde{z} < z^*$. Let $f(z)$ describe the function which finds z_{k+1} from z_k as shown in (3-5). Given this notation, we know that since $\tilde{z} < z^*$, the augmented primal problem (3-2) is infeasible and thus $f(\tilde{z}) > \tilde{z}$ by Theorem 3.3. Now we need to show that $f(z)$ is continuous near $\tilde{z}$.

We know that $f(\tilde{z})$ is a function of the residual vector u, so we will first examine the behavior of u as $\tilde{z}$ is perturbed by $\epsilon > 0$. Consider the augmented primal (3-2). This can be rewritten as

$$\begin{aligned} \tilde{A}\tilde{x} &= \tilde{b} \\ \tilde{x} &\ge 0, \end{aligned}$$

where

$$\tilde{A} = \begin{pmatrix} c^T & 1 \\ A & 0 \end{pmatrix}; \quad \tilde{x} = \begin{pmatrix} x \\ y \end{pmatrix}; \quad \tilde{b} = \begin{pmatrix} \tilde{z} \\ b \end{pmatrix}.$$

When the Phase I algorithm terminates with infeasibility on (3-2), we have $\tilde{A} = (\tilde{B} \ \ \tilde{N})$, where $\tilde{B}$ is made up of the basic columns and $\tilde{N}$ is made up of the nonbasic columns. The current solution, $\tilde{x}$, is found by solving the normal equations $\tilde{x} = (\tilde{B}^T\tilde{B})^{-1}\tilde{B}^T\tilde{b}$. We know we can form this expression, since the columns of $\tilde{B}$ are linearly independent. If $\tilde{b}$ is perturbed, the Phase I solution $\tilde{x}$ must be adjusted, and one of two cases can occur.

Case 1: The perturbation does not cause a new column to enter $\tilde{B}$.
Letting

$$\tilde{b}(\epsilon) = \begin{pmatrix} \tilde{z}-\epsilon \\ b \end{pmatrix},$$

we have

$$\tilde{x}(\epsilon) = (\tilde{B}^T\tilde{B})^{-1}\tilde{B}^T \begin{pmatrix} \tilde{z}-\epsilon \\ b \end{pmatrix},$$

in which case $\tilde{x}$ is clearly a continuous function of ϵ. Moreover, $u(\epsilon) = \tilde{b}(\epsilon) - \tilde{x}(\epsilon)$ is also a continuous function of ϵ. Therefore, $f(\tilde{z})$ is also a continuous function of ϵ in the range $[\tilde{z}-\epsilon,\tilde{z}]$, provided that (3-1) is feasible. (If (3-1) is infeasible, we could encounter $u_0 = 0$, and thus have no need to form z_{k+1}.)

Case 2: The perturbation causes a new column to enter $\tilde{B}$.

Call the entering column $\tilde{N}_s$. Thus

$$\tilde{x}(\epsilon) = \left((\tilde{B} \quad \tilde{N}_s)^T (\tilde{B} \quad \tilde{N}_s) \right)^{-1} (\tilde{B} \quad \tilde{N}_s)^T \begin{pmatrix} \tilde{z} - \epsilon \\ b \end{pmatrix}.$$

We know that at $\epsilon = 0$,

$$\tilde{x}(\epsilon) = \tilde{x}(0) = \begin{pmatrix} \tilde{x} \\ 0 \end{pmatrix},$$

where $\tilde{x}$ is as defined in Case 1. We also know that for small enough ϵ, no columns will be dropped from $\tilde{B}$ because we have $\tilde{x}$ *strictly* greater than 0. Therefore, for small enough $\epsilon > 0$, $\tilde{x}$ is a continuous function of ϵ. Thus u is a continuous function of ϵ, and $f(\tilde{z})$ is a continuous function of ϵ in $[\tilde{z} - \epsilon, \tilde{z}]$ (again provided that (3-1) is feasible).

Thus in both cases, for some $\epsilon > 0$, $f(z)$ is continuous in $[\tilde{z} - \epsilon, \tilde{z}]$. From Theorem 3.3, we have $f(z) > z \ \forall \ z \in [\tilde{z} - \epsilon, \tilde{z}]$ because $\tilde{z} < z^*$. Then from Lemma 3.5, $\exists \ y \in [\tilde{z} - \epsilon, \tilde{z})$ such that $f(z) > \tilde{z} \ \forall \ z \in [y, \tilde{z}]$. Hence $\tilde{z} < z^*$ cannot be the limit point of the sequence $\{z_k\}$. This contradiction completes the proof.
□

3.4 The Phase II Algorithm

The following algorithm solves (3-1) using a series of Phase I subproblems.

Step 1:

Solve the augmented primal (3-2) using the least-squares Phase I algorithm presented in Section 2. We will assume that z_0 is less than or equal to the optimal value z^*. (We will discuss how to select z_0 in the next section.) If we find a feasible solution, we are done. Otherwise, the augmented system is infeasible and we go to Step 2.

Step 2:

Let $u = (u_0, \tilde{u})$ be the residual vector from the infeasible augmented system as found by the Phase I algorithm in Step 1. If $u_0 = 0$, then the primal (3-1) is infeasible (by Theorem 3.2). If $u_0 \neq 0$, then find z_{k+1} using (3-5). Go back to Step 1 and solve the augmented primal problem again, replacing z_k with z_{k+1}. We know that $z_{k+1} > z_k$, and that the sequence of iterates converges to z^*, so we will iterate towards the optimal solution.

3.5 Discussion

In this section we will consider the questions of

- Transitions between augmented primal problems
- Primal unboundedness
- Finding $z_0 < z^*$

- Convergence

Transitions

After performing Step 1 of the algorithm, we have a least-squares solution to the problem

$$\min \left\| \begin{pmatrix} z_k \\ b \end{pmatrix} - \begin{pmatrix} c^T & 1 \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \right\|. \tag{3-6}$$

However, once we replace z_k with z_{k+1}, we no longer have a least-squares solution to the augmented primal system. In fact, we may need to alter the current basis in order to get a new least-squares solution.

It seems reasonable to assume that the current basis (associated with z_k) will be fairly similar to the final basis (associated with z_{k+1}). Therefore, we solve the least-squares problem (3-6), replacing z_k by z_{k+1}. If any components of the solution are negative, we form a convex combination with the previous least-squares solution (which we know is positive). This is just like the usual method for dropping columns.

We keep forming convex combinations and dropping columns until we have a (possibly proper) subset of the columns of the basis associated with z_k, only now we have a least-squares solution, and the weightings are all positive. As the bases are expected to be fairly similar, this should not be too many steps. (In practice, this is indeed the case.) Once we have the new least-squares solution and the (possibly reduced) basis, we look for an incoming column and perform the least-squares Phase I algorithm as usual.

Primal Unboundedness and Finding z_0

One way to find a $z_0 \leq z^*$ is to solve the dual feasibility problem. That is, find a solution to the system $A^T\pi \leq c$ using the least-squares Phase I algorithm variation allowing free variables. If a feasible solution is found, we can set z_0 to the objective value at this solution ($z_0 = \pi^T b \leq z^*$). If no solution is found, we can conclude that the primal problem is unbounded. At first sight, dualizing the problem in this manner appears to necessitate the solution of a problem much larger than the primal problem because $n > m$. This turns out not to be the case if we take advantage of the fact that $n - m$ (or more) of the columns will be unit vectors associated with the slack dual variables.

Another approach to finding an appropriate z_0 might be to set z_0 to an arbitrarily large negative number, such as -10^{10}. Machine precision (in our case) is 10^{-19}, so that this is still many orders of magnitude away from severe numerical problems. It is also likely that our problem will not have an optimal objective value below -10^{10} (provided it is not unbounded). If, when we solve the augmented primal problem, we discover that $z_0 = -10^{10}$ is indeed feasible, we can still go and attempt to solve the dual feasibility problem in order to find an appropriate $z_0 \leq z^*$, (or give up, declaring that there may have been some mistake in the formulation of the original problem).

Convergence

It is possible that the sequence $\{z_k\}$ could have a slow rate of convergence, especially near z^*. The following is a scheme guaranteed to have at least linear convergence, as it is based on bisection.

First solve the primal feasibility problem, that is, find a solution to (2-1). This gives us a feasible z; call it z_f. Next, solve the dual feasibility problem, that is, find a solution to $A^T\pi \leq c$. This gives us a z_0. Note that (3-2) will be infeasible as long as $z_0 \neq z^*$. In any case, we know that $z_0 \leq z^* \leq z_f$.

Now solve the augmented primal problem using z_0, and find z_1 using the formula (3-5). If

$$z_1 \geq \frac{z_0 + z_f}{2},$$

then use the usual z_1 as found using (3-5) in the next augmented primal system to solve. Otherwise, use

$$z_1 = \frac{z_0 + z_f}{2}. \tag{3-7}$$

If we use the usual z_1, we need no further discussion. However, if we use z_1 as found using (3-7), we have two cases to consider.

Case 1: We find that z_1 causes the augmented primal to be infeasible.

In this case, we know

$$z_1 = \frac{z_0 + z_f}{2} \leq z^* \leq z_f,$$

and we can therefore find the next z_{k+1} using (3-5). Note that we have eliminated

$$\left(z_0, \frac{z_0 + z_f}{2}\right)$$

from the interval which could contain z^*.

Case 2 We find that z_1 is feasible in the augmented primal.

In this case, we know

$$z_0 \leq z^* \leq z_1 = \frac{z_0 + z_f}{2}.$$

We redefine z_f to be $\frac{1}{2}(z_0 + z_f)$, and go back to find a z_1 from z_0 using (3-5) as before. This time, we have eliminated

$$\left(\frac{z_0 + z_f}{2}, z_f\right)$$

from the candidate interval for z^*. Note that in the general step, we will be finding z_{k+1} from z_k, and that we will know $z_k \leq z^* \leq z_f$.

Using the process described above, we can eliminate at least half of the remaining interval at each step. Thus we have at least linear convergence in the number of Phase I problems solved.

In practice, we really do not want to find a feasible solution for both the primal and dual systems (in order to find z_0 and z_f) as described above, as this would be extremely expensive. Therefore, in our implementation of the Phase II algorithm, we do not find an initial feasible solution to either system. We start with $z_0 = -10^{10}$ as discussed earlier, and only find a solution to $A^T\pi \leq c$ if this first z_0 proves to be feasible in the augmented primal problem.

In our first implementation of the Phase II algorithm, we did actually initialize z_f by finding a primal feasible solution. However, it was observed that the binary search was very rarely invoked—that is, z_{k+1} usually eliminated well over half of the remaining interval, and convergence was excellent. Thus we removed the initialization of z_f (and consequently one Phase I problem). However, we found that numerical problems occasionally produced a $z_{k+1} < z_k$, especially when z_k was close to z^*. To solve this problem, we set $z_{k+1} = z_k + |\, z_{k+1} - z_k \,|$, which tended to produce a feasible augmented primal. If so, z_f was set to z_{k+1}, and the binary search described earlier was used. If not, a new z_{k+1} was found in the usual manner, and the above scheme was repeated. This modified binary search scheme only guarantees linear convergence once a valid z_f is known. However, in practice it ran faster than did the original binary search method, as the Phase I problem required to initialize z_f was eliminated. This was the scheme used to produce the results in Section 4. It should be noted that for most of the test problems, this modified binary search was not actually invoked, as the situation where $z_{k+1} < z_k$ was relatively rare.

4 Computational Results

All computational results have been obtained using the 30 smallest linear programming test problems available from *NETLIB* [5]. These test problems were converted to the form of (2-1), with the addition that free variables were allowed (although nonpositive variables were not). This produced a set of equivalent problems with modified dimensions; see Table 4-1 for the original and modified dimensions. For bitmaps of the nonzero patterns of many of the original problems, see [11].

The test environment consisted of the following hardware and software:

Computer:	HP 9000/835
	32M main memory
	100M virtual memory
Operating System:	HP-UX 7.0
Language:	HP FORTRAN, 64-bit real numbers

	Original		Converted	
Problem(Name)	Rows	Columns	Rows	Columns
1(AFIRO)	28	32	28	51
2(SC50A)	51	48	51	78
3(SC50B)	51	48	51	78
4(ADLITTLE)	57	97	57	138
5(BLEND)	75	83	75	114
6(SHARE2B)	97	79	97	162
7(SC105)	106	103	106	163
8(STOCFOR1)	118	111	118	165
9(RECIPE)	92	180	212	298
10(SCAGR7)	130	140	130	185
11(BOEING2)	167	143	244	382
12(ISRAEL)	175	142	175	316
13(SHARE1B)	118	225	118	253
14(VTP.BASE)	199	203	346	475
15(SC205)	206	203	206	317
16(GROW7)	141	301	421	581
17(BEACONFD)	174	262	174	295
18(BRANDY)	221	249	221	303
19(SCSD1)	78	760	78	760
20(E226)	224	282	224	472
21(FORPLAN)	162	421	187	514
22(BORE3D)	234	315	247	346
23(AGG)	489	163	489	615
24(CAPRI)	272	353	419	613
25(SCORPION)	389	358	389	466
26(BANDM)	306	472	306	472
27(SCTAP1)	301	480	301	660
28(SCFXM1)	331	457	331	600
29(STAIR)	357	467	445	620
30(SCSD6)	148	1350	148	1350

Table 4-1: Size of Test Problems

The run times reported here include only computation time, as the amount of main memory used virtually eliminated swapping, and I/O time was negligible.

4.1 Finding an Optimal Solution

We now consider the computational results produced by the Phase II algorithm developed in Section 3, along with variations analogous to those discussed in Section 2. See Section 3.4 for pseudo code of the Phase II algorithm. Again, we only consider those variations in which free variables are allowed. In addition, all variations considered contain the refinement to implement the modified binary search scheme as described in Section 3.5. We thus consider the following four Phase II algorithms:

P2F: The least-squares Phase II algorithm, in which free variables have not been explicitly converted to pairs of nonnegative variables.

P2D1F: The same as P2F, with the exception that the denominator of the column selection rule is replaced by "1".

CBP2F: The same as P2F, using a crash basis as described in Section 2.

CBP2D1F: The same as P2D1F, using a crash basis as described in Section 2.

For the purpose of comparison and evaluation, we define two types of iterations in these Phase II algorithms. A major iteration is defined to be whenever a new z_{k+1} is calculated, and the augmented primal (3-2) is altered. A minor iteration represents an iteration of the least-squares Phase I algorithm (see [10] for an exact definition).

Note that it is the *augmented primal* (3-2) that is normalized to have columns of length 1, as all of the least-squares Phase I algorithms assume that this has been done. Since the first element of the right-hand side changes each time z_k is updated (and consequently changes its norm), it was decided to normalize the right-hand side only once, assuming for the sake of normalization that $z_k = 0$. This assumption allows us to normalize the augmented primal problem only once per Phase II problem, instead of once per major iteration.

All of the Phase II methods were implemented using dense matrix methods, and QR factorization was used to solve the embedded least-squares subproblems. The factorization was updated using Givens rotations as columns were added and dropped. See for example [7] for details on using the QR factorization to solve least-squares problems, and for details on Givens rotations (also known as plane rotations). Transitions between major iterations were handled as discussed in Section 3.5.

Finally, z_0 was set to an "arbitrarily" large negative number (in our implementation, $z_0 = -10^{10}$). Machine precision in our case is 10^{-19}, which is several orders of magnitude away from severe numerical difficulties. If, when we solve the first augmented primal problem, we discover that $z_0 = -10^{10}$ is indeed feasible, then we can go and solve the dual feasibility problem $A^T\pi \leq c$ in order to find an appropriate $z_0 \leq z^*$ where z^* is the optimal objective value. Notice that this method assumes no prior information about z^*. As it happened, it was never necessary to solve the dual feasibility problem for the problems in our test suite, as $z_0 = -10^{10}$ was always less than z^*. We also found (through experimentation) that as long as $z_0 \leq z^*$, the Phase II algorithms are fairly insensitive to the choice of z_0.

The Phase II algorithms are compared with LSSOL, an established package. LSSOL is written in Fortran 77 using dense matrix techniques, and solves a class of linearly constrained quadratic programming problems. See [6] for a more detailed description of LSSOL. In order to compare our Phase I algorithm to LSSOL, we use the "LP" option. This causes LSSOL to find an optimal

Problem(Name)	P2F	P2D1F	CBP2F	CBP2D1F	LP
1(AFIRO)	26:04	26:04	30:04	30:04	8
2(SC50A)	73:03	73:03	80:03	84:03	4
3(SC50B)	83:03	83:03	81:03	81:03	1
4(ADLITTLE)	257:05	279:05	227:06	273:31	121
5(BLEND)	218:04	220:04	222:04	224:04	86
6(SHARE2B)	409:07	451:07	423:07	464:07	110
7(SC105)	171:04	165:04	190:04	186:04	10
8(STOCFOR1)	236:03	297:03	236:03	297:03	91
9(RECIPE)	315:05	308:05	360:05	358:05	65
10(SCAGR7)	516:06	691:06	490:06	711:28	84
11(BOEING2)	593:07	590:07	574:07	575:07	257
12(ISRAEL)	731:06	739:06	715:06	719:06	559
13(SHARE1B)	325:04	344:04	342:04	355:04	306
14(VTP.BASE)	2260:13	2224:14	812:13	1239:22	1122
15(SC205)	332:04	332:04	374:04	374:04	9
16(GROW7)	905:06	911:06	893:06	892:06	332
17(BEACONFD)	205:07	209:07	171:07	175:07	100
18(BRANDY)	1002:07	1000:07	976:07	976:08	352
19(SCSD1)	165:06	192:06	165:06	192:06	67
20(E226)	1148:05	1094:05	1182:05	1137:05	735
21(FORPLAN)	764:08	771:08	732:08	736:08	583
22(BORE3D)	569:08	601:08	579:08	611:08	188
23(AGG)	1446:11	1407:11	1740:11	1791:12	231
24(CAPRI)	892:09	891:09	936:09	930:09	566
25(SCORPION)	857:08	1018:09	826:08	998:08	57
26(BANDM)	1258:05	1275:05	1334:05	1316:05	627
27(SCTAP1)	1169:10	1210:10	1034:10	1119:10	560
28(SCFXM1)	1490:06	1405:07	1502:06	1422:06	511
29(STAIR)	679:08	679:08	722:08	722:08	257
30(SCSD6)	603:07	721:07	603:07	721:07	112
TOTAL	19697	20206	18551	19708	8111

Table 4-2: Phase II Algorithm and LSSOL:LP Iteration Counts

solution to the problem submitted, using its implementation of the simplex method. LSSOL (LP option) will be denoted by LP in the following result tables, and by LSSOL:LP in the text.

Table 4-2 displays the iteration counts of the four Phase II algorithms and LSSOL:LP. As both major and minor iterations of the Phase II algorithms are of interest, their iteration counts are written as "minor:major" in this table. Notice that LSSOL:LP did not successfully solve problems SCSD1 and SCSD6, stalling at 67 and 112 iterations respectively, after performing 50 iterations with no change in the solution. Although the Phase II algorithm rarely needed more than 10 major iterations on any given problem, they all used over twice as many iterations as did LSSOL:LP.

Looking at the run times in Table 4-3, we see that iteration count is not a good indicator of the performance of the Phase II algorithms. All of the Phase II

Problem(Name)	P2F	P2D1F	CBP2F	CBP2D1F	LP
1(AFIRO)	0	0	0	0	1
2(SC50A)	1	1	2	2	1
3(SC50B)	2	2	2	2	1
4(ADLITTLE)	7	8	7	9	8
5(BLEND)	8	8	8	8	11
6(SHARE2B)	22	25	23	25	21
7(SC105)	13	12	14	14	6
8(STOCFOR1)	18	24	18	24	24
9(RECIPE)	74	71	88	86	71
10(SCAGR7)	49	62	48	69	30
11(BOEING2)	195	198	195	190	265
12(ISRAEL)	142	141	143	142	335
13(SHARE1B)	31	32	34	35	89
14(VTP.BASE)	1262	1243	440	699	2147
15(SC205)	84	84	97	96	41
16(GROW7)	900	917	1019	973	1140
17(BEACONFD)	35	36	32	32	62
18(BRANDY)	199	203	203	210	237
19(SCSD1)	34	37	35	37	14
20(E226)	253	359	407	386	584
21(FORPLAN)	203	204	198	191	489
22(BORE3D)	161	173	173	180	203
23(AGG)	1942	1914	2318	2263	1163
24(CAPRI)	910	931	1002	995	1730
25(SCORPION)	641	732	651	720	328
26(BANDM)	704	703	760	728	911
27(SCTAP1)	739	771	667	724	880
28(SCFXM1)	1026	952	1061	975	876
29(STAIR)	809	783	851	848	1048
30(SCSD6)	388	447	383	447	68
TOTAL	10852	11073	10879	11110	12784

Table 4-3: Phase II Algorithm and LSSOL:LP Run Times

algorithms performed better (by total time) than did LSSOL:LP, with the best, P2F, running 15% faster. In contrast to the least-squares Phase I algorithm results, however, the Phase II algorithm did not always perform better than did LSSOL; there were a number of problems in which LSSOL:LP was faster.

Finally we consider the issue of accuracy. The Phase II algorithms have some difficulties with accuracy, just as do the least-squares Phase I algorithms as described in [10]. Although most of the optimal objective values found by the Phase II algorithms are quite close to the true values, there are some notable exceptions. The most inaccurate values tended to be found on problems suspected of ill-conditioning, such as FORPLAN and VTP.BASE. It is possible that these difficulties with accuracy could be alleviated by using some sort of iterative refinement (see [2]), but this variation was not implemented.

5 Summary and Conclusions

Section 2 presented a Phase I algorithm using least-squares subproblems. Section 3 used the algorithm from Section 2 to produce a Phase II algorithm based on solving a sequence of augmented feasibility problems, each solved by the least-squares Phase I algorithm. Convergence to the optimal solution was proved, and with a minor alteration of the algorithm, convergence was shown to be linear in the number of augmented feasibility problems solved. Computational results for this Phase II algorithm were generally good, but not as spectacular as for the least-squares Phase I algorithm. The variations of the Phase II algorithm implemented required many more iterations than did LSSOL's implementation of the simplex method. However, run times were better, with the best Phase II algorithm variation requiring 85% of the total run time of LSSOL:LP.

Future Work

As stated in [10], although the results presented here are promising, the Phase II algorithm must be fine-tuned in order to be commercially competitive, just as the simplex method has been since its discovery. In particular, the problem of inaccuracy on unstable problems must be addressed, perhaps with the inclusion of iterative refinement of the solutions to the embedded least-squares subproblems. In addition, these algorithms should be implemented using sparse matrix methods to see how they compare to sparse matrix implementations of the simplex method.

References

[1] **E.M.L. Beale (1955)**, "Cycling in the Dual Simplex Algorithm", *Naval Res. Logist. Quart.*, 2(4), 1955, pp.269–276.

[2] **Å. Björck (1967/68)**, "Iterative Refinement of Linear Least Squares Solutions", *BIT* 7, pp. 257–278 and *BIT* 8, pp. 8–30.

[3] **Å. Björck (1987)**, "Least Square Methods", Working Paper, Department of Mathematics, Linköping University, S-581 83 Linköping, Sweden, 1987.

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

[5] **D.M. Gay (1985)**, "Electronic Mail Distribution of Linear Programming Test Problems", *Mathematical Programming Society COAL Newsletter* 13, pp.10–12.

[6] **P.E. Gill, S.J. Hammarling, W. Murray, M.A. Saunders & M.H. Wright (1986)**, "User's Guide for LSSOL", Technical Report SOL 86-1, Department of Operations Research, Stanford University.

[7] **P.E. Gill, W. Murray & M.H. Wright (1981)**, *Practical Optimization*, Academic Press, San Francisco.

[8] **A.J. Hoffman (1953)**, "Cycling in the Simplex Algorithm", National Bureau of standards, Report No. 2974, December 16, 1953, 7 pp.

[9] **C.L. Lawson & R.J. Hanson (1974)**, *Solving Least-Squares Problems*, Prentice-Hall, Englewood Cliffs, New Jersey.

[10] **S.L. Leichner, G.B. Dantzig & J.W. Davis**, "A Strictly Improving Phase I Algorithm Using Least-Squares Subproblems", Technical Report SOL 92-1, April 1992, Department of Operations Research, Stanford University.

[11] **I.J. Lustig (1987)**, "An Analysis of an Available Set of Linear Programming Test Problems", Technical Report SOL 87-11, August 1987, Department of Operations Research, Stanford University.

[12] **C. Van de Panne & A. Whinston (1969)**, "The Symmetric Foundation of the Simplex Method for Quadratic Programming", *Econometrica*, Vol. 37, pp.507–527.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1992	Technical Report

4. TITLE AND SUBTITLE

A Strictly Improving Linear Programming Algorithm Based on a Series of Phase I Problems

5. FUNDING NUMBERS

N00014-89-J-1659
DE-FG03-92ER25116

6. AUTHOR(S)

S. A. Leichner, G. B. Dantzig and J. W. David

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Department of Operations Research - SOL
Stanford University
Stanford, CA 94305-4022

8. PERFORMING ORGANIZATION REPORT NUMBER

1111MA

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research - Department of the Navy
800 N. Quincy Street
Arlington, VA 22217

Office of Energy Research
U.S. Department of Energy
Washington, DC 20585

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

SOL 92-2

12a. DISTRIBUTION/AVAILABILITY STATEMENT

UNLIMITED

12b. DISTRIBUTION CODE

UL

13. ABSTRACT *(Maximum 200 words)*

When used on degenerate problems, the simplex method often takes a number of degenerate steps at a particular vertex before moving to the next. In theory (although rarely in practice), the simplex method can actually cycle at such a degenerate point. Instead of trying to modify the simplex method to avoid degenerate steps, we have developed a new linear programming algorithm that is completely impervious to degeneracy.

This new method solves the Phase II problem of finding an optimal solution by solving a series of Phase I feasibility problems. Strict improvement is attained at each iteration in the Phase I algorithm, and the Phase II sequence of feasibility problems has linear convergence in the number of Phase I problems.

When tested on the 30 smallest *NETLIB* linear programming test problems, the computational results for the new Phase II algorithm (using the series of feasiblity problems) were over 15problems, it was almost two times faster, and on one problem it was four times faster.

14. SUBJECT TERMS

Linear Programming; least squares; strict improvement.

15. NUMBER OF PAGES

20 pages

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED			SAR