Implementation and Empirical Study of a Combined Phase I - Phase II Potential Reduction Algorithm For Linear Programming

Hitendra K. Wadhwa
Robert M. Freund

#3411-92-MSA March 1992

IMPLEMENTATION AND EMPIRICAL STUDY OF A COMBINED PHASE I - PHASE II POTENTIAL REDUCTION ALGORITHM FOR LINEAR PROGRAMMING

by

HITENDRA K. WADHWA
and
ROBERT M. FREUND

IMPLEMENTATION AND EMPIRICAL STUDY OF A COMBINED PHASE I - PHASE II POTENTIAL REDUCTION ALGORITHM FOR LINEAR PROGRAMMING

by

HITENDRA K. WADHWA
and
ROBERT M. FREUND

ABSTRACT

This paper consists of an implementation of an O(nL)-iteration combined Phase I - Phase II potential reduction algorithm for linear programming. The main distinguishing feature of this algorithm is that it allows the user to specify the desired balance between the Phase I and Phase II objectives. This paper presents an empirical study of this algorithm that is aimed at investigating different aspects of the algorithm's behavior and the effects of different parameters on its performance.

Contents

SECTION I

Introduction

This paper consists of an implementation of an O(nL)-iteration combined Phase I - Phase II potential reduction algorithm for linear programming. The main distinguishing feature of this algorithm is that it allows the user to specify the desired balance between the Phase I and Phase II objectives. This paper presents an empirical study of this algorithm that is aimed at investigating different aspects of the algorithm's behavior and the effects of different parameters on its performance.

In the last seven years a number of interior point algorithms for linear programming, based essentially on Karmakar's new approach [6], have been developed. Most of them have been "Phase II" algorithms, requiring an initial feasible interior solution. The problem of getting this initial solution has been addressed, as in the case of the simplex method, by developing a reformulated "Phase I" problem on which these algorithms could then be applied to yield a feasible interior solution.

Researchers in interior point linear programming methods have also sought to develop combined Phase I - Phase II algorithms that would start from some initial infeasible solution and proceed simultaneously to attain feasibility and optimality, instead of following the above sequential Phase I - Phase II approach. These methods are particularly valuable for facilitating "warm starts", where one seeks to solve a slightly modified version of a linear program by starting from the original LP's optimal solution. Anstreicher [1] developed one such combined Phase I - Phase II algorithm based on Karmakar's projective transformation approach, which was later modified and improved upon by Todd [8]. The same two researchers later developed combined Phase I - Phase II algorithms ([2], [8]) using the direct potential function reduction approach developed by Gonzaga [5], Freund [3] and Ye [11]. These algorithms seek to simultaneously reduce two potential functions, which may be appropriately termed the Phase I and Phase II potential functions, in order to approach feasibility and optimality respectively. The algorithm

presented by Freund [4] also uses the same formulation of the linear programming problem, and this algorithm forms the subject of this paper.

Unlike the algorithms of Anstreicher [2] and Todd [9], Freund's algorithm [4] uses only one potential function to simultaneously move toward feasibility and optimality. Anstreicher's and Todd's methods place equal emphasis on making progress on the Phase I and Phase II objectives. LP users often need to place a greater emphasis on one or the other of these two goals. The distinguishing feature of Freund's method is that it allows the user to specify the balance that the algorithm should maintain between the Phase I and the Phase II objectives over the iterations, thus making it adaptable to the different goals of users.

In Section II, we comment on the aforementioned desirability of flexibly balancing the two different objectives in a combined Phase I - Phase II problem, and then describe how Freund's algorithm addresses this issue. The algorithm itself is described in Section III, which also includes a discussion of some of its features and options. In Section IV we show how a linear program in standard form with an initial infeasible solution can be converted into the form required by the algorithm. In Section V, we discuss some implementation details and present and discuss the findings of our empirical study. Finally, in Section VI we summarize the study's findings and mention some unresolved questions for future research.

Appendix A contains tables that are referred to in Section V. Appendix B contains a copy of the computer programs that were developed as part of the implementation process.

SECTION II

The need for a balance in the Phase I and Phase II objectives

Linear programming helps in modelling and solving a variety of operations research problems, and consequently, the goals of different users can be quite varied. Such differences in goals result, in practice, in the desirability for having a flexible algorithmic environment that can be tailored to suit the specific needs of a user. Desirable algorithmic features include, for instance, the allowance for different tolerance levels for the accuracy of the optimal solution (which all LP algorithms provide), or the ability to weigh differently the goals of achieving feasibility and achieving optimality. The need for the latter is illustrated by the following general cases:

(i) *Branch and Bound*: In using branch and bound techniques for solving integer or mixed integer programming problems, lower bounds on subproblems may be generated by solving their linear programming relaxations. By getting good lower bounds very soon in the solution process, branches can often be immediately pruned, and thus the focus here is often on seeking efficient progress in the Phase II problem without paying much heed to the Phase I problem. For these LPs, one may expect the user to want to place more weight on the Phase II objective than on the Phase I objective.

(ii) *Partial equilibrium problems*: Certain partial equilibrium problems in economics can be formulated as linear programs, with the optimal solution to the linear program and the associated dual prices yielding the partial equilibrium solution (see Wagner [10]). In such problems, the only feasible solution that carries useful economic information is the optimal solution, and so both Phase I and Phase II objectives carry importance.

(iii) *Optimization with unclear/uncertain objectives* : In a number of actual optimization problems that can be modelled as linear programs, the 'real' objective is not clear. This may be because of the unclear or uncertain nature of some costs, or the presence of multiple goals and objectives. If at such times the constraints are clearly formulated, then we have a situation where solving the Phase I problem is of significantly greater import than the Phase II

problem, since the latter is relatively ill-defined. Clearly, in such a case the Phase I problem carries much greater weight that the Phase II problem.

In searching for a method to integrate a flexible balancing of the Phase I and Phase II objectives in a combined Phase I - Phase II algorithm, we must first define a measure of the *infeasibility* and *nonoptimality* of any solution, and then define a way in which more emphasis could be laid on reducing one over the other. We describe now the manner in which Freund's method addresses these issues.

Balancing Phase I and Phase II in Freund's algorithm

The algorithm in this paper follows the formulation of the combined Phase I -Phase II problem developed by Anstreicher [2]. This involves solving the following linear program:

$$\text{LP} \quad z^* = \quad \text{Min}_x \; c^T x$$

$$Ax = b \qquad (2.1a)$$
$$\xi^T x = 0 \qquad (2.1b)$$
$$x \geq 0 \qquad (2.1c)$$

where we are given an initial vector x^0 that is feasible for the following "Phase I" problem:

$$\text{LPI} \quad \text{Min}_x \; \xi^T x$$

$$Ax = b \qquad (2.2a)$$
$$\xi^T x \geq 0 \qquad (2.2b)$$
$$x \geq 0 \qquad (2.2c)$$

and in fact, by virtue of its (assumed) infeasibility in LP, has $\xi^T x^0 > 0$. We are also given an initial lower bound B^0 on the optimal solution value z^* of LP. We

describe how one can convert a general linear program to the above form in Section IV.

At any iteration k, the algorithm computes a new point x^k , feasible for LPI, and a new lower bound B^k on the optimal value z* of LP. We define $\xi^T x^k$ to be the 'feasibility gap', a measure of the infeasibility at x^k. The duality gap at x^k is $(c^T x^k - z^*)$, a measure of the nonoptimality at x^k.

The Phase I objective, then, is to reduce the feasibility gap $\xi^T x^k$ to 0, and, similarly, the Phase II objective is to reduce the duality gap $(c^T x^k - z^*)$ to 0. Since the optimal value z* is not known a priori, the algorithm instead uses B^k as a surrogate for z*, and measures the nonoptimality at iteration k by the gap $(c^T x^k - B^k)$.

(Note: One may observe here that the generation of good lower bounds (so that the B^k's are close to z*) would be important if these bounds are to be good surrogates for z*, and this is one of the issues studied in the implementation section of this paper.)

Now we are ready to describe the manner in which the algorithm balances the Phase I and Phase II objectives. The user is provided with a parameter, β, which can be set at the start of the algorithm to be any strictly positive real number. The algorithm then ensures that at each iteration k, the current point x^k and the lower bound B^k satisfy the inequality:

$$c^T x^k - B^k \leq \beta \xi^T x^k \qquad (2.3)$$

Through this inequality, the parameter β acts as the sought-after balancing mechanism, and we now show how this occurs.

Suppose first that the user would like to weigh the Phase II objective much more than the Phase I objective. We would then expect him/her to choose a small value for β, say $\beta = .001$. Why? Because, for this β, (2.3) ensures that the duality gap at each iteration is significantly smaller than the feasibility gap

(in fact, by at least a factor of 1000), and so the algorithm is laying much more stress on reducing the Phase II gap than on reducing the Phase I gap, as desired.

Suppose now that the user wants to weigh the Phase I objective much more than the Phase II objective. Assume for a moment that the inequality (2.3) was actually satisfied almost as an equality over the iterations. A large value of β (say $\beta = 1000$) would then ensure that the progress over the iterations in the Phase I objective would substantially dominate that in the Phase II objective. Since (2.3) is actually an inequality, we do not actually have a guarantee that this will happen as described, although from the nature of (2.3) and the large value of β chosen it would seem very likely that relatively greater progress would be made over the iterations in the feasibility gap reduction. More comments on this follow below.

Since β can be any positive real number, the algorithm provides the user with a lot of flexibility in how the two objectives may be weighed. Another way to look at (2.3) is by dividing both sides of the inequality by $\xi^T x^k$, which results in:

$$\frac{c^T x^k - B^k}{\xi^T x^k} \leq \beta$$

This suggests that one can look at β as the user-desired ratio of the duality gap to the feasibility gap over the iterations. Thus, a higher value of β would correspond to greater emphasis being placed on feasibility gap reduction, and a smaller value would correspond to greater emphasis being placed on duality gap reduction, just as we saw above.

As one may have observed, in order for the parameter β to provide very effective balancing control, (2.3) must hold close to equality over the iterations. Whether this actually does happen or not cannot be investigated theoretically, however, and this makes a strong case for an empirical study of the algorithm's behavior with respect to (2.3). Such a study formed an important part of this paper' research, and is described in Section V.

SECTION III

The algorithm

This section contains a description of the algorithm that forms the subject of this paper' study. It is assumed that there is given a linear programming problem in the form (2.1), with an initial solution x^0 and an initial lower bound B^0. The next section will describe how one can reformulate an LP in the standard form into one of the type (2.1).

Among the inputs required are q, which must satisfy $q \geq n+1 + \sqrt{n+1}$, $\beta > 0$, the balancing parameter, and two tolerance specifications, tola and tolb.

Notation

For any vector z, Z will denote the diagonal matrix whose diagonal is the vector z.

For any vector z, $z_{1:k}$ will denote the vector consisting of the first k components of z

For any vector $g \in R^b$ and matrix $M \in \mathbf{R}^{axb}$ with full row rank, g_M will denote the projection of g in the null space of M, i.e., $g_M = [I - M^T(MM^T)^{-1}M]g$

Summary of the algorithm

Inputs: A,b,c, ξ, β, x^0,B^0,q, tola, tolb

Step 0*(Initialization)*

$k = 0$

$e = (1,1,...,1) \in \mathbf{R}^n$

$e' = (e^T,1)^T$

$t^0 = B^0 - (-\beta\xi + c)^T x^0$

Step 1*(Rescaling and reinitialization)*

$\bar{x} = x^k$

$\bar{t} = t^k$

$$\bar{B} = B^k$$
$$\bar{A} = A\bar{X}$$
$$\bar{\xi} = \bar{X}\xi$$
$$\bar{\xi}' = (\bar{\xi}^T, 0)^T$$

Step 2 *(Updating lower bound using Fraley's restriction of the dual)*

Solve $FD_{\bar{x}}$ $\quad z_{\bar{x}} = \text{Max}_{\theta,\eta,s} \quad c^T\bar{x} - \xi^T\bar{x}\,\theta - \bar{x}^T s$

$$\theta(\bar{X}\xi)_{\bar{A}} + \bar{X}s + \eta(e - e_{\bar{A}}) = (\bar{X}c)_{\bar{A}}$$
$$s \geq 0$$

$$\bar{t} \leftarrow \bar{t} + \max\{\bar{B}, z_{\bar{x}}\} - \bar{B}$$
$$\bar{B} \leftarrow \max\{\bar{B}, z_{\bar{x}}\}$$

Step 3 *(Compute Primal Direction)*

$$A1 = \begin{bmatrix} \bar{A} & 0 \\ -\beta\xi^T\bar{X} + c^T\bar{X} & \bar{t} \end{bmatrix}$$

$$d1 = (q/\xi^T\bar{x})\bar{\xi}'_{A1} - e'_{A1}$$

If $\bar{\xi}^T d1 \geq 0$

$$d = d1$$

else

$$A2 = \begin{bmatrix} A1 \\ \bar{\xi}'^T \end{bmatrix}$$

$$d2 = (q/\xi^T\bar{x})\bar{\xi}'_{A2} - e'_{A2}$$
$$d = d2$$

Step 4. *(Line-Search of Potential Function)*

$$\bar{\alpha} = \text{ArgMin}_{\alpha} \quad q\log[\xi^T(\bar{x} - \alpha\bar{X}d_{1:n})] - \sum_{j=1}^{n} \log(\bar{x}_j - \alpha\bar{x}_j d_j) - \log(\bar{t} - \alpha\bar{t}d_{n+1})$$

$$\text{s.t.} \quad \bar{x} - \alpha\bar{X}d_{1:n} > 0$$
$$\bar{t} - \alpha\bar{t}d_{n+1} > 0$$

$x^{k+1} \leftarrow \bar{x} - \bar{\alpha}\bar{X}d_{1:n}$
$t^{k+1} \leftarrow \bar{t} - \bar{\alpha}\bar{t}d_{n+1}$
$B^{k+1} \leftarrow \bar{B}$

Step 5 *(Optimality check)*

If $\xi^T\bar{x} <$ tola and $(c^T\bar{x} - \bar{B}) <$ max{1, $|c^T\bar{x}|$} tolb
    end
else
    k = k+1
    go to step 1

Notes on the algorithm

In order to get a better understanding of the algorithm, consider the following family of linear programming problems:

LP_B $\quad \text{Min}_{x,t} \quad \xi^T x$

$$Ax = b \qquad (3.1a)$$
$$(-b\xi + c)^T x + t = B \qquad (3.1b)$$
$$x \geq 0, \; t \geq 0 \qquad (3.1c)$$

with $B \leq z^*$.

For $B = z^*$, it can be shown that the optimal value of LP_B is 0, and that an optimal solution to LP_{z^*} is also optimal for LP (2.1) (see Freund [4]). The algorithm presented above solves LP (2.1) by considering a sequence of LP_B's with increasing B values that approach z^*.

Taking an interior point approach, the algorithm replaces the family of LP_B's by the following family of potential reduction problems:

$$PR_B \qquad Min_{x,t} \qquad f(x,t) = q\log(\xi^T x) - \sum_{j=1}^{n} \log(x_j) - \log(t) \qquad (3.2)$$

$$Ax = b$$
$$(-b\xi + c)^T x + t = B$$
$$x > 0, t > 0$$

with $B \leq z^*$.

At the start of any iteration, the algorithm has a solution $\bar{x}$ and a lower bound $\bar{B}$ to LP (2.1). The algorithm updates the lower bound $\bar{B}$ by using a two dimensional restricted version of the dual of LP. This is the Fraley Dual $FD_{\bar{x}}$ defined in step 2. For two different derivations of this restricted dual problem, see Todd [9] and Freund [4].

(Note: FD can be solved as a two-dimensional linear program with n inequalities in the variables θ and η. One way of doing so is by using a Fourier-Motzkin elimination procedure, and this was the technique used in the implementation process discussed in this thesis.)

Once the lower bound has been set at $\bar{B}$, $LP_{\bar{B}}$ becomes the current problem, and the algorithm tries to move from $\bar{x}$ towards the solution for $LP_{\bar{B}}$ by moving in the direction d in step 3.

The direction d1 is the projection of the rescaled gradient of the potential function (3.2) in the null space of the constraints (3.1a-b).

The algorithm enforces monotonicity in the Phase I objective function ($\xi^T x$), and so, if -d1 is an ascent direction for this function, the algorithm instead uses the direction -d2, which is the projection of the rescaled gradient of the potential function (3.2) in the null space of the constraints (3.1a-b) and the constraint

$$\xi^T x = 0$$

After determining the direction of movement d, the algorithm performs a line search so as to get the maximum feasible decrease in the potential function. Todd and Burrell [7] have shown that the potential function (3.2) is quasiconvex, and so a line-search is easily performed by using, for example, a binary search procedure on the derivative of the potential function with respect to α over the interval $\alpha \in [0,u)$, where

$$u = \min_{1 \le i \le n+1} \{ 1/d_i : d_i > 0 \}$$

Such a binary search procedure was used in the implementation of the algorithm that is described in this paper.

'tola' and 'tolb' are the tolerance levels for the Phase I and Phase II objectives. In our implementation of the algorithm, we used a different stopping procedure from the one defined in step 5, and this is described in Section V.

Modifications

1. Early feasibility: The early feasibility option allows the user to test whether the direction of movement $\bar{d} = \bar{X} d$ (computed in step 3) will allow for a Phase I solution, by testing whether $\bar{x} - \alpha \bar{X} d_{1:n}$ is feasible for LP for some value of α.

The constraints in LP (2.1) are:

$$Ax = b \qquad (2.1a)$$

$$\xi^T x = 0 \qquad (2.1b)$$

$$x \ge 0 \qquad (2.1c)$$

Now $$A(\bar{x} - \alpha \bar{X} d_{1:n}) = A\bar{x} - \alpha A\bar{d} = b - 0 = b,$$

and so (2.1a) is satisfied.

In order for $\bar{x} - \alpha \bar{X} d_{1:n}$ to satisfy (2.1b), α must be given the value $\alpha = \xi^T \bar{x} / \xi^T \bar{X} d_{1:n}$. But then, in order for $\bar{x} - \alpha \bar{X} d_{1:n}$ to satisfy the last set of constraints (2.1c), we need

$$\xi^T \bar{x} / \xi^T \bar{X} d_{1:n} < 1/d_i \ , \ 1 \le i \le n \ . \qquad (3.3)$$

Thus, if the inequalities in (3.3) are satisfied, then $\bar{x} - (\xi^T \bar{x} / \xi^T \bar{X} d_{1:n}) \bar{X} d_{1:n}$ is a feasible interior solution for LP. We may then proceed to directly use a Phase II algorithm from this iteration onward.

The effect of this 'early feasibility' option on the algorithm's performance is an issue that the theory is unable to resolve, and a part of our empirical study in Section V is directed towards investigating this.

2. *'Rotation' instead of 'Parallel Shift'* : The algorithm above may be modified in the following manner:

Instead of maintaining a constant value of the balancing parameter β through the iterations, this value may be changed whenever the lower bound B increases in a manner that would cause the hyperplane

$$(-\beta \xi + c)^T x = B$$

to be rotated about the fixed affine space $\{x: \ c^T x = B^0 + \beta_0 \xi^T x^0$ and $\xi^T x = \xi^T x^0 \}$ instead of being shifted parallel to itself as above (β_0 is the initial value of β). The results in Freund [4] related to the complexity analysis of the algorithm can be extended to show that the above modification also leads to an O(nL)-iteration algorithm. The actual modification would involve the following addition to step 3:

$$\beta^{k+1} \leftarrow \beta^0 + (B^0 - B^{k+1}) / \xi^T x^0$$

SECTION IV

Converting a standard form LP into required format

This section contains a description of the manner in which an LP in the standard form with an accompanying (possibly infeasible) initial solution can be converted into the combined Phase I - Phase II format (2.1) that is used by Freund's algorithm.

The standard form of the LP is:

$$\text{LP} \quad z^* = \quad \text{Min}_{\hat{x}} \ \hat{c}^T\hat{x}$$

$$\hat{A}\hat{x} = \hat{b} \qquad (4.1a)$$

$$\hat{x} \geq 0 \qquad (4.1b)$$

Suppose we have a linear program in the above form, where $\hat{A}$ is an m x n matrix with row rank m. Suppose also that $\hat{x}^0$ is a given, possibly infeasible solution, and that $\hat{B}^0$ is a given lower bound on z^*. $\hat{B}^0$ may be a 'good' lower bound known to the user, or it may be an artificial bound. Todd [9] has developed an algorithm to generate a reasonable lower bound. In Section V, we will investigate the ability of Freund's algorithm to generate 'good' lower bounds very early in the solution process.

If $\hat{x}^0$ does not satisfy the constraints

$$\hat{A}\hat{x} = \hat{b}$$

we replace $\hat{x}^0$ by the solution to the problem:

$$\begin{aligned} &\text{Min}_{\hat{x}} \ \|\hat{x} - \hat{x}^0\| \\ &\text{s.t.} \quad \hat{A}\hat{x} = \hat{b} \end{aligned} \qquad (4.1)$$

This solution is given by $x^0_{\hat{A}} + \hat{A}^T(\hat{A}\hat{A}^T)^{-1}\hat{b}$, where $x^0_{\hat{A}}$ is the projection of $\hat{x}^0$ in the null space of the matrix $\hat{A}$. We denote this new initial solution by $\hat{x}^0$ also. Then we have

$$\hat{A}\hat{x}^0 = \hat{A}(x^0_{\hat{A}} + \hat{A}^T(\hat{A}\hat{A}^T)^{-1}\hat{b}) = 0 + \hat{b} = \hat{b}$$

(Note: The optimization problem described above seeks to find the $\hat{x}$ that is closest to $\hat{x}^0$ (in terms of its euclidean distance from $\hat{x}^0$) among those that satisfy (4.1). Actually, any of the solutions to (4.1) might be acceptable replacements of $\hat{x}^0$ for our purpose.)

If $\hat{x}^0 > 0$, we have a feasible interior solution to LP and we do not need to solve a Phase I problem - we may instead use any of the interior point Phase II algorithms (such as those of Ye [11], Freund [3], Gonzaga [5]) to solve LP. So let us assume that $\hat{x}^0_j \leq 0$ for some j.

Let $h \in \mathbf{R}^n$ be such that :

$\hat{b} \neq r\hat{A}h$ for any real number r, (4.2a)

$h \geq 0$, and (4.2b)

$\hat{x}^0 + h > 0$ (4.2c)

An $h \in \mathbf{R}^n$ that satisfies (4.4b)and (4.2c) is easily derived; if such an h does not satisfy (4.2a) also, it can be perturbed to give us the desired vector.

LP is equivalent to :

$$LP^2 \quad z^* = \quad \mathrm{Min}_{\hat{x},\, w} \quad \hat{c}^T\hat{x}$$

$$\hat{A}\hat{x} = \hat{b} \qquad (4.3a)$$

$$\hat{x} + wh \geq 0 \qquad (4.3b)$$

$$w = 0 \qquad (4.3c)$$

where now $(\hat{x}, w) = (\hat{x}^0, 1)$ satisfies all of the above constraints except (4.3c).

Let $x = \hat{x} + w\,h$. We can then write LP^2 in terms of the variables (x, w) as:

$LP^3 \quad z^* = \quad \text{Min}_{x,\, w} \quad \hat{c}^T x - w\,\hat{c}^T h$

$$\hat{A}x - w\hat{A}h = \hat{b} \qquad (4.4a)$$

$$x \geq 0 \qquad (4.4b)$$

$$w = 0 \qquad (4.4c)$$

We define $x^0 = \hat{x}^0 + h$. Then $(x, w) = (x^0, 1)$ satisfies all of the above constraints except (4.4c).

We will now eliminate w from LP^3. Assume that $\hat{A}h \neq 0$ (otherwise LP does not need a Phase I). Premultiplying (4.4a) by $(\hat{A}h)^T$, we get:

$(\hat{A}h)^T\hat{A}x - w\|\hat{A}h\|^2 = (\hat{A}h)^T\hat{b}$, which gives

$$w = \frac{h^T\hat{A}^T\hat{A}x}{\|\hat{A}h\|^2} - \frac{h^T\hat{A}^T\hat{b}}{\|\hat{A}h\|^2}$$

and so LP becomes equivalent to:

$LP^4 \quad z^* = \text{Min}_x \; \hat{c}^T[I - (hh^T\hat{A}^T\hat{A})/\|\hat{A}h\|^2]\,x + (\hat{c}^T h)(h^T\hat{A}^T\hat{b})/\|\hat{A}h\|^2$

$$[I - (\hat{A}hh^T\hat{A}^T)/\|\hat{A}h\|^2]\,\hat{A}x = [I - (\hat{A}hh^T\hat{A}^T)/\|\hat{A}h\|^2]\,\hat{b} \qquad (4.5a)$$

$$x \geq 0 \qquad (4.5b)$$

$$h^T\hat{A}^T\hat{A}x = h^T\hat{A}^T\hat{b} \qquad (4.5c)$$

We note that x^0 satisfies all of the above constraints except for (4.5c)

We are now almost done; only constraint (4.5c) needs some further attention. We need to replace it by a constraint of the form $\xi^T x = 0$.

Define $A = [I_m - (\hat{A}hh^T\hat{A}^T)/\|\hat{A}h\|^2]\hat{A}$

$c = [I_n - (hh^T\hat{A}^T\hat{A})/\|\hat{A}h\|^2]^T\hat{c}$

and $b = [I_m - (\hat{A}hh^T\hat{A}^T)/\|\hat{A}h\|^2]\hat{b}$

where I_r is the identity matrix of size r x r.

We note that $b \neq 0$, for otherwise $\hat{b} = [(h^T\hat{A}^T)\hat{b}/\|\hat{A}h\|^2]\hat{A}h$, which contradicts condition (4.2a) on h.

We now choose an i for which $b_i \neq 0$. Let $\alpha = h^T\hat{A}^T\hat{b}/b_i$. Then (4.5c) can be replaced by

$$(-\alpha A_{i.} + \hat{A}^T\hat{A}h)^T x = 0$$

Note that
$$\begin{aligned}(-\alpha A_{i.} + \hat{A}^T\hat{A}h)^T x^0 &= (-\alpha A_{i.} + \hat{A}^T\hat{A}h)^T(\hat{x}^0 + h) \\ &= -\alpha A_{i.}^T\hat{x}^0 + h^T\hat{A}^T\hat{A}\hat{x}^0 - \alpha A_{i.}^T h + h^T\hat{A}^T\hat{A}h \\ &= -\alpha([I_m - (\hat{A}hh^T\hat{A}^T)/\|\hat{A}h\|^2]\hat{A})_{i.}^T\hat{x}^0 + h^T\hat{A}^T\hat{b} - \\ &\quad \alpha([I_m - (\hat{A}hh^T\hat{A}^T)/\|\hat{A}h\|^2]\hat{A})_{i.}^T h + \|\hat{A}h\|^2 \\ &= -\alpha[I_m - (\hat{A}hh^T\hat{A}^T)/\|\hat{A}h\|^2]_{i.}^T\hat{A}\hat{x}^0 + h^T\hat{A}^T\hat{b} - \\ &\quad \alpha([I_m - (\hat{A}hh^T\hat{A}^T)/\|\hat{A}h\|^2]\hat{A})_{i.}^T h + \|\hat{A}h\|^2 \qquad (4.6)\end{aligned}$$

Now,
$$\begin{aligned}([I_m - (\hat{A}hh^T\hat{A}^T)/\|\hat{A}h\|^2]\hat{A})h &= \hat{A}h - (\hat{A}hh^T\hat{A}^T)(\hat{A}h)/\|\hat{A}h\|^2 \\ &= \hat{A}h - (\hat{A}h)(h^T\hat{A}^T\hat{A}h)/\|\hat{A}h\|^2 \\ &= \hat{A}h - (\hat{A}h)\|\hat{A}h\|^2/\|\hat{A}h\|^2 \\ &= 0,\end{aligned}$$

and so $\alpha([I_m - (\hat{A}hh^T\hat{A}^T)/\|\hat{A}h\|^2]\hat{A})_{i.}^T h = 0$

Thus, from (4.6) :

$$(-\alpha A_{i.} + \hat{A}^T\hat{A}h)^T x^0 = -\alpha[I_m - (\hat{A}hh^T\hat{A}^T)/\|\hat{A}h\|^2]_{i.}^T\hat{A}\hat{x}^0 + h^T\hat{A}^T\hat{b} - 0 + \|\hat{A}h\|^2$$

$$
\begin{aligned}
&= -\alpha\,[I_m - (\hat{A}h\,h^T\hat{A}^T)/\|\hat{A}h\|^2]_{i.}{}^T\hat{b} + h^T\hat{A}^T\hat{b} + \|\hat{A}h\|^2 \\
&= -\alpha b_i + h^T\hat{A}^T\hat{b} + \|\hat{A}h\|^2 \\
&= -(h^T\hat{A}^T\hat{b}/b_i.)b_{i.} + h^T\hat{A}^T\hat{b} + \|\hat{A}h\|^2 \\
&= \|\hat{A}h\|^2 \\
&> 0
\end{aligned}
$$

Define $\xi = -\alpha A_{i.} + \hat{A}^T\hat{A}h$. Then, from above, we have $\xi^T x^0 > 0$.

Also, LP^4 is equivalent to:

LP^5 $\quad z^* = \quad \text{Min}_x \; c^T x + (\hat{c}^T h)(h^T\hat{A}^T\hat{b})/\|\hat{A}h\|^2$

$$
\begin{aligned}
Ax &= b \qquad (4.6a) \\
\xi^T x &= 0 \qquad (4.6b) \\
x &\geq 0\,, \qquad (4.6c)
\end{aligned}
$$

with x^0 being feasible for the related Phase I problem:

LP5 - I $\quad \text{Min}_x \; \xi^T x$

$$
\begin{aligned}
Ax &= b \\
\xi^T x &\geq 0 \\
x &\geq 0\,,
\end{aligned}
$$

with $\xi^T x^0 > 0$.

Removing the constant term $(\hat{c}^T h)(h^T\hat{A}^T\hat{b})/\|\hat{A}h\|^2$ from the objective function of LP^5, we arrive at the desired format (2.1) for our linear program. x^0 is the initial solution and $B^0 = \hat{B}^0 - (\hat{c}^T h)(h^T\hat{A}^T\hat{b})/\|\hat{A}h\|^2$ is the initial lower bound.

In addition to having the problem set up in the above manner, the algorithm also requires that the starting point x^0 satisfy:

$$(-\beta\xi + c)^T x^0 < B^0 \qquad (4.7)$$

where β is the user determined balancing parameter ($\beta > 0$).

To do this, we proceed as follows. If the above x^0 does in fact satisfy (4.7), we are done. If not, then consider $x^0 + w^0h$, where

$$w^0 = \max\{0, (c^Tx^0 - B^0)/(\beta\|\hat{A}h\|^2)\} \qquad (4.8)$$

We note that :

Since $h \geq 0$ and $w^0 \geq 0$, so $x^0 + w^0h \geq 0$.

$Ah = ([I_m - (\hat{A}hh^T\hat{A}^T)/\|\hat{A}h\|^2]\hat{A})h = 0$, as shown earlier, and so

$$A(x^0 + w^0h) = Ax^0 = b$$

$$(-\beta\xi + c)^T(x^0 + w^0h) = -\beta\xi^Tx^0 + c^Tx^0 - \beta\omega^0\xi^Th + w^0c^Th \qquad (4.7)$$

Now $c^Th = ([I_n - (hh^T\hat{A}^T\hat{A})/\|\hat{A}h\|^2]^T\hat{c})^Th$

$= \hat{c}^T[I_n - (hh^T\hat{A}^T\hat{A})/\|\hat{A}h\|^2]\,h$

$= \hat{c}^Th - \hat{c}^T(hh^T\hat{A}^T\hat{A})\,h/\|\hat{A}h\|^2$

$= \hat{c}^Th - \hat{c}^Th(h^T\hat{A}^T\hat{A}h)/\|\hat{A}h\|^2$

$= 0$

and $\xi^Th = (-\alpha A_{i.} + \hat{A}^T\hat{A}h)^Th$

$= 0 + h^T\hat{A}^T\hat{A}h$

(since $Ah = ([I_m - (\hat{A}hh^T\hat{A}^T)/\|\hat{A}h\|^2]\hat{A})h = 0$, as shown above)

$= \|\hat{A}h\|^2$.

So, from (4.7),

$$\begin{aligned}(-\beta\xi + c)^T(x^0 + w^0h) &= -\beta\xi^Tx^0 + c^Tx^0 - \beta\omega^0\|\hat{A}h\|^2 \\ &\leq -\beta\xi^Tx^0 + c^Tx^0 - \beta\|\hat{A}h\|^2(c^Tx^0 - B^0)/(\beta\|\hat{A}h\|^2) \\ &= -\beta\xi^Tx^0 + c^Tx^0 - (c^Tx^0 - B^0) \\ &= -\beta\xi^Tx^0 + B^0 \\ &< B^0, \text{ since } \xi^Tx^0 > 0, \text{ and } \beta > 0.\end{aligned}$$

Hence $x^0 + w^0h$ satisfies all the required constraints, and can be used in place of x^0 as the initial solution.

SECTION V

The implementation study is concerned with investigating certain issues pertaining to the algorithm's performance and behavior. This section contains a discussion of these issues, an analysis of the empirical results, and some comments on the implementation itself.

Issues studied

1. *Stopping Criterion*: What is a practical and efficient strategy for getting the algorithm to compute a solution that lies within desired Phase I and Phase II tolerance limits?

2. *The effect of the balancing parameter* β: Does the ratio

$$\frac{c^T x^k - B^k}{\xi^T x^k}$$

actually lie quite close to the value of the parameter β over the sequence of iterations?

3. *Generation of good lower bounds*: How good is the Fraley dual lower bound updating procedure described in step 2 of the algorithm's summary in Section III ? How is the algorithm's performance affected by the use of a good initial lower bound instead of an artificial one?

4. *Effect of the parameter q:* q is the multiplier on the "$\log(\xi^T x)$" term of the potential function in PR_B (Section III), and features in Step 3 of the algorithm's summary in Section III. What value should the parameter q be given? Does the algorithm perform better for some values of q over others?

5. *Effect of the early feasibility option*: What effect does the inclusion of this option have on the algorithm's performance? Can we characterize the situations in which this option improves or worsens the performance of the algorithm?

6. *Different 'Parallel Shift' approaches versus 'Rotation'* : How does the 'Rotation' version of the algorithm compare with the 'Parallel Shift' approach? In the 'Parallel Shift' case, what is the best setting for the parameter β from a performance-based perspective (i.e., ignoring the Phase I - Phase II balancing consideration)?

Before referring to these issues, some aspects of the testing and implementation process are discussed next.

Implementation and testing details

Programming Language: The algorithm was encoded in MATLAB, a mathematical software package that provides a convenient programming environment for matrix-based problems, and itself uses LINPACK and EISPACK algorithms. MATLAB deals with real numbers in double precision. Since the aim of the research was to study certain basic algorithmic issues, the nature of the code developed was not in itself a primary consideration, though, as mentioned later in this section, certain numerical issues did need to be resolved. The iteration count was used as the measure of the algorithm's performance.

Test Problems: The empirical testing was done on a library of test problems that consisted of two sets of 15 problems each. One set of problems had the constraint matrix $\hat{A}$ to be of size 50 x 100, and the other set of problems had the constraint matrix $\hat{A}$ to be of size 25 x 50. In this thesis, the terms 'Size 100' and 'Size 50' will be used to denote the sizes of problems belonging to the former and latter categories respectively. Each of these LP problems was generated randomly in the manner described below.

A random LP problem of size 100 was generated as follows: A constraint matrix $\hat{A}$ of size 50 x 100 was generated with each element of the matrix being derived from a standard Normal distribution, independently of the other elements. Two vectors, $\hat{c} \in R^{100}$ and $\hat{x} \in R^{100}$ were also generated, with each element of the two vectors being derived from a uniform [0,1] distribution, again independently of the other elements. Then $\hat{b} \in R^{50}$ was derived as $\hat{b} = \hat{A}\hat{x}$. An LP in the standard form (4.1) was then provided by the data $(\hat{A}, \hat{b}, \hat{c})$.

Note that this LP has an interior solution $\hat{x}$. Also, since $\hat{c} \geq 0$ (by the manner in which it was derived), and every feasible $\hat{x}$ also satisfies $\hat{x} \geq 0$, the optimal solution to this LP must also be non-negative, and so $\hat{B}^0 = 0$ is a (hopefully, good) lower bound on this LP. In fact, given any feasible solution $\hat{x}$, we have the following result:

$$\hat{B}^0 \leq 0 \leq z^* \leq \hat{c}^T\hat{x} \leq n = 100.$$

Since the intent was to start the algorithm from an infeasible point, and $\hat{x}$ was already an interior feasible solution to the LP, an initial infeasible vector $\hat{x}^0 \in \mathbf{R}^{100}$ was still needed. This was generated by deriving each of its elements from a standard Normal distribution, independently of the other elements. Thus, the complete data for the problem was given by $((\hat{A}, \hat{b}\ \hat{c}, \hat{x}^0$ and $\hat{B}^0)$.

Random LP problems of size 50 were derived similarly.

The procedure described in Section IV was then used to arrive at a transformed version of each LP on which the algorithm could be directly applied.

Base Case: To test various issues, one needed to either study some aspect of the algorithm's behavior under some specific setting of an algorithmic option or parameter, or compare the algorithm's performance under two or more alternative settings of some such option or parameter. A base case was set up in order to fix a 'default' setting for each of these algorithmic options and parameters. These settings under the base case are given below:

<u>Base Case</u>

(q, b, tola, tolb and 'early feasibility' are defined in Section III)

$q = n+1 + \sqrt{n+1}$ (n = size of the problem)
$\beta = 1$
No early feasibility option
tola = tolb = 10^{-3}
Good lower bound (that which was derived from the problem generation process described above).

The value of q chosen provides an O(nL)-iteration limit on the algorithm's convergence. A β value of 1 implies that an equal emphasis is being sought between Phase I and Phase II gap reduction over the iterations. The reasons for choosing the above tolerance levels is given later.

Numerical Issues: As the iterates x^k get close to the optimal solution, some of their coefficients become very small in value. This can lead to certain numerical problems in practice that may stall progress in the final stages of the solution process. One problem in particular is the high degree of inaccuracy in the computation of the search direction d caused by the bad conditioning of the matrix $[(A1)(A1)^T]$ and its inverse (see Step 3 of the algorithm in Section III). Unless these numerical issues are resolved, the algorithm may perform very inefficiently in practice near the optimum , and this may distort the implementation results to varying extents. The task of devising algorithmic and implementation strategies that circumvent these problems falls in the domain of numerical analysis and computation, and the research described here was not concerned with tackling these issues.

Unfortunately, in the testing process some significant numerical problems in the context described above did emerge. These led to a need to devise a practically sound manner of dealing with the final stage of the solution process (including the choice of a suitable stopping criterion). Some of the test cases on which the problems persisted had to be ignored in the final compilation and analysis of the results.

The numerical problems discussed above precluded the use of tight tolerances, for given the level of computational imprecision in the final stages, such tolerances would not be effective. After testing some cases with the tolerance levels of 10^{-5}, 10^{-4} and 10^{-3}, the last one was chosen as the value for both tola and tolb, since the computer program was unable to achieve the higher levels of precision consistent with the tighter tolerances. This low level of precision by the program was unexpected and somewhat puzzling, and the probable

causes behind this behavior were not immediately apparent to the author or his advisor.

Analysis and discussion of results

1. *Stopping Criterion:* At first, the stopping criterion described in Step 5 of the algorithm's summary in Section III was used. However, in a number of cases the solution process was significantly affected by the factors outlined in the discussion on numerical issues above. Typically, the Phase I objective value would come within the tolerance limits before the numerical problems started playing a dominating role - after that point, the directions computed were highly inaccurate, and in many cases convergence to within tolerance limits in the Phase II objective required an extremely large number of additional iterations (with the algorithm having to be stopped in some cases before it had satisfied the Phase II tolerance level).

The following stopping procedure was then applied:

If $\xi^T\bar{x} <$ tola and $(c^T\bar{x} - \bar{B}) < \max\{1, |c^T\bar{x}|\}$tolb

End

else if $\xi^T\bar{x} <$ tola

Solve the linear program

LPII $\quad \text{Min}_x \; c^Tx$

$Ax = b$

$\xi^Tx = \xi^T\bar{x}$

$x \geq 0$

using $\bar{x}$ as the initial interior feasible solution and $B = \bar{B} + (\xi^T\bar{x})\,\theta'$ as the initial lower bound, where θ', along with some η', s', is the solution to the restricted dual $FD_{\bar{x}}$ of step 2 that was solved to give the present lower bound $\bar{B}$.

(The reasons for this step are provided below.)

This LP has an initial interior feasible solution, and so any of the 'Phase II ' interior point algorithms can be applied to solve it. An implementation of the potential function reduction algorithm described in Freund [3] was used for this purpose.

The rationale for arriving at the above final-stage procedure is as follows: One intuitively suspects that, given the numerical problems, an important reason for the algorithm's bad behavior in practice after it has achieved the desired Phase I reduction could be its persistence with the 'Phase I potential function'

$$f(x,t) = q\log(\xi^T x) - \sum_{j=1}^{n} \log(x_j) - \log(t)$$

to determine directions of movement d when it is Phase II convergence that is required. Hence one may think of shifting to some 'Phase II potential function' after the Phase I tolerance is achieved. We could thus look at the 'feasible region' given by

$$\begin{aligned} &Ax = b \\ &0 \le \xi^T x \le \xi^T \bar{x} \\ &x \ge 0 , \end{aligned}$$

and try to reduce the Phase II gap to satisfy the tolerance limit tolb within this region. Given the manner in which the problem is derived from the original LP (2.1), it can be shown that the problem

$$\begin{aligned} \text{Min}_x \quad & c^T x \\ & Ax = b \\ & 0 \le \xi^T x \le \xi^T \bar{x} \\ & x \ge 0 , \end{aligned}$$

has the same optimal solution as the problem

$$\text{Min}_x \quad c^T x$$

$$Ax = b$$
$$\xi^T x = \xi^T \bar{x}$$
$$x \geq 0 .$$

One final ingredient still needs explanation: the specification of the lower bound B. To see why B is a valid lower bound on the linear program (LPII), consider the family of linear programs

LP_α $\quad Min_x \; c^T x$

$$Ax = b$$
$$\xi^T x = \alpha$$
$$x \geq 0 .$$

Note that LP_0 is the same as LP(4.1), which is the problem being solved. The dual of LP_α is given by

D_α $\quad Max_{\lambda,\mu} \; b^T\lambda + \alpha\mu$

$$A^T\lambda + \mu\xi \leq c$$

Note that the feasible region for D_α is independent of α.

Now suppose that $\bar{B}$ is the current lower bound on LP(4.1), and that $\bar{x}$ is the current iterate. Also assume that $\bar{B}$ is not the initial lower bound B^0. Then, from the algorithm's summary in Section III, it is clear that $\bar{B}$ must have been derived in Step 2 as the solution value to a restriction $FD_{\tilde{x}}$ of the dual Do of LP (2.1) (note that $\tilde{x}$ may be different from $\bar{x}$ since the lower bound may have been updated to $\bar{B}$ at an earlier iteration). Hence, there is some solution $(\theta', \eta', \sigma')$ to $FD_{\tilde{x}}$ for which the objective value equals $\bar{B}$. Since $FD_{\tilde{x}}$ is a restriction of D_0, so in fact there is some solution (λ', μ') of Do for which the objective function value equals $\bar{B}$. It can also be easily seen from looking at how $FD_{\tilde{x}}$ is a restriction of D_0, that in fact $\mu' = \theta'$. Since the feasible region of all the D_α's are the same, and the objective function is given by $b^T\lambda + \alpha\mu$, so for $\alpha = \xi^T\bar{x}$ there is a solution (λ', θ') to D_α with objective solution value equal to $\bar{B} + (\xi^T\bar{x})\,\mu'$, and this must then be a valid lower bound on LP_α.

The above modification did result in resolving the numerical problems for most test problems, though for some problem instances the algorithm's performance continued to be hampered by them. These instances were therefore excluded from the final compilation and analysis of the empirical results, and an asterisk has been printed in place of any data for such cases in the tables that follow.

2. *The effect of the balancing parameter* β: As mentioned in Section I, an important question that needs to be studied empirically is whether, over the sequence of iterations, the ratio

$$r^k = \frac{c^T x^k - B^k}{\xi^T x^k} \qquad (5.1)$$

does in fact lie quite close to the value of the parameter β. To investigate this issue, the value of r^k was recorded after each iteration of the algorithm for a number of test problems. Table 1 provides some information on this matter. The following inferences are easy to draw from this table:

- For higher values of β ($\beta \geq 1$), r^k lies quite close to β.
- For very low values of β, r^k is often negative in many problem instances, which implies that for those instances the iterates are more or less tracing a 'known-superoptimal' path to the optimum solution.

(Note: When the ratio (5.1) is negative, we have $c^T x^k < B^k$, and so, in such a case, the objective value at the current solution is known to be lower than the optimal solution value z^*. This is what is meant by the solution being 'known-superoptimal'.)

We note that the ratio (5.1) staying quite close to β implies that the iterates are staying quite close to the constraint

$$(-\beta\xi+c)^T x + t = B \qquad (3.1b)$$

of the problem LP_B. This helps in giving us an intuitive understanding for the inferences drawn above. When β is large, the constraint (3.1b) intuitively looks like the feasibility constraint

$$\xi^T x = 0 \qquad (2.1b)$$

and so the attraction force by the '$q\log(\xi^T x)$' term of the potential function (3.2) will serve to attract the iterates close to the constraint. When β is small, however, the constraint (3.1b) is intuitively very different from the feasibility constraint, and in this case the "-log(t)" term of the potential function will serve to repel the iterate from the constraint. Hence, in this case, the iterates may not lie consistently close to this constraint, and thus r^k may not remain close to β over the iterations.

3. *Generation of good lower bounds*: Step 2 of the algorithm's summary in Section III describes the use of a Fraley dual (a restriction of the dual of the original problem) in updating the lower bound at the start of each iteration. As mentioned in Section II, it seems important that the algorithm be able to compute good lower bounds on the optimal solution value in a timely manner. To explore this aspect of its behavior empirically, and also to answer the question of how the performance of the algorithm is affected by the use of good versus artificial initial lower bounds, the base case algorithm was applied to each of the test problems of size 100 with the good initial lower bound being replaced by an artificial bound of $B^0 = -10^5$). The lower bounds generated over the iterations were recorded.

We term the case of using a good initial lower bound (the base case) as Case I and that of using an artificial initial lower bound as Case II. The 'Case I' and 'Case II' columns of Table 2 provide data on the algorithm's performance in the two cases. The algorithm performed quite unfavorably in Case II as compared to Case I. This difference in performance may then be attributed to the following factors:

1. The lower bounds used in Case I are, at least initially, much better than those used for Case II.

2. The starting point in Case I is much closer to the feasible region than that in Case II. (Here, our measure of the degree of closeness of a point x to the feasible region is $\xi^T x$. Note that this will be zero for a feasible point.)

The analysis provided here is more on an intuitive level, and these results can be contradicted in odd circumstances. We notice, for instance, that for problem 9 in Table 2, the algorithm took fewer iterations in Case II than in Case I.

The first column of Table 3 shows the number of iterations taken by the algorithm in Case II to generate the first good lower bound, and this points towards the influence of the first factor. An intuitive explanation for why the second factor may be true is given next.

The initial solution x^0 has to satisfy

$$(-\beta\xi + c)^T x^0 < B^0 \qquad (5.2)$$

Suppose first that we are starting from an artificial lower bound, so that B^0 is very negative. Then the initial solution x^0 ($= \hat{x}^0 + h$) that is derived from the transformation process described in the first part of Section IV will probably not satisfy (5.2), and we will instead have to apply the method discussed in the last part of that section to arrive at a positive real number w^0 for which $x' = x^0 + w^0 h$ satisfies this inequality. This x' will then be used as the initial point for the algorithm. We now show why this new point x' will in general lie much further from the feasible region than the previous point x^0.

We recall that the degree of infeasibility of a point x is given by the feasibility gap $\xi^T x$. We note, as in Section IV, that $c^T h = 0$. Now since x' will satisfy (5.2), we have

$$(-\beta\xi + c)^T (x^0 + w^0 h) < B^0,$$

which implies

$$-\beta\xi^T (x^0 + w^0 h) + c^T x^0 < B^0 \qquad (5.3)$$

It is easy to see from (5.3) that $\xi^T x = \xi^T(x^0 + w^0 h)$ will need to be much larger than $\xi^T x^0$, and so x' is much further from the feasible region (for which $\xi^T x = 0$) than x^0.

Now consider the case where B^0 is actually a good lower bound. Then either x^0 will satisfy (5.2), and hence will be the starting point, or we will again use the same process as above to arrive at an $x'' = x^0 + w'^0 h$ which satisfies this inequality. However, in this case, since B^0 is much less negative than it was in the case above, x" will not be shifted as far away from the feasible region as x' was. Hence, it should be a starting point that lies much closer to the feasible region than x'.

In order to seek any improvement in the algorithm's performance under an artificial initial lower bound, we would have to focus on the second factor mentioned above, since the first one is a direct and inevitable consequence of using an artificial instead of a good bound. We would thus like to find a way by which we can prevent a large shift away from x^0 while initiating the solution process. This objective led to the following modification to the first stage of the algorithm. The algorithm would start from the given initial solution x^0, with (5.2) being satisfied by instead choosing a different initial value for β, which would be such that $\beta > (c^T x^0 - B^0)/\xi^T x^o$. This value of β would be used until the algorithm generated a 'good' lower bound (much less negative than the artificial one), at which point the algorithm would shift over to the user-specified value for β, using the technique of Section IV if required in order to ensure that the current iterate x^k (or its derivative of the form $x^k + wh$, for some positive real number w) satisfied (5.2).

In this modified version of the algorithm, then, the solution process begins from the x^0 that is derived from the first part of Section IV, and the user-specified β is only used once a good lower bound is generated. Interestingly, the Fraley dual bound-updating technique, when used with this modified form of the algorithm, turned out to be very effective. Tables 2,3 and 4 provide some data on the results, which are now discussed.

On the problems of size 50, three versions of the algorithm were initially run:
1. The base case, with good initial lower bounds (Case I).

2. Same as above, except with artificial initial lower bounds (Case II).
3. Same as the above, except with the algorithm modified as described above. (Case III)

Table 2 provides the iteration count for the three cases. We observe that not just does the algorithm in Case III perform much better than that in Case II, but it almost performs as well as the case with the good initial lower bounds (Case I). In order to gain some more insight into the nature of the impact that the modification in Case III had on the algorithm's behavior, the data shown in Table 3 was collected. This table shows the number of iterations it took in Cases II and III for the algorithm to generate a good lower bound. As can be seen, this number was much higher in Case II that in Case III. This implies that in fact the effect of the two factors mentioned in the beginning of this part of this section cannot be considered independently of each other, for the second factor seems to influence the first one: solutions that are far away from the feasible region result in poor bounds being generated, and solutions that are near the feasible region help in generating good bounds.

Table 4 shows the number of iterations taken by the algorithm for the problems of size 100 for Cases I and III.

It is striking to note that in Case III (for both size 100 and size 50 problems), despite starting from as low an artificial bound as 10^{-5}, good bounds were generated in an average of 2 iterations. As a consequence, the algorithm's performance was not much different from that in Case I. This seems to imply that with this modified approach, the value of knowing and using a good initial lower bound is very little. This issue needs to be explored further.

4. *Effect of the parameter q:* Theoretically, q needs to satisfy

$$q \geq n+1 + \sqrt{n+1} \qquad \text{and} \quad q = O(n)$$

in order for the algorithm to provably converge in O(nL)-iterations. However, in practice, it is useful to test different values of q before determining at what value to set q.

The empirical results presented in Table 5 show the number of iterations taken by the algorithm for all 15 problems of size 50 under five different settings of q: $q = n+2$, $n+1 + \sqrt{n+1}$, $n+1 + (n+1)$, $n+1 + (n+1)^2$, $n+1 + (n+1)^3$. The data in the table seems to suggest that the algorithm's performance, on the average, improves as q is increased, though it seems to plateau between $n+1 + (n+1)$ and $n+1 + (n+1)^3$. It would be interesting to explore the algorithm's performance for even higher values of q.

5. *Effect of the early feasibility option*: The Phase II procedure used in the early feasibility option was a potential-reduction algorithm (with line searches and Fraley dual lower bound updates) based on Freund's [3] approach. Tables 6a and 6b provides comparative performance data between the base case version and the case with the early feasibility option for the size 50 and size 100 problems. From the data in these tables we can infer that the early feasibility option significantly enhances the performance of the algorithm, by allowing the algorithm to concentrate exclusively on Phase II gap reduction once it has achieved feasibility early on in the solution process.

There were some instances where this option seemed to make the algorithm perform worse. Unfortunately, further inferences on when this option may or may not be useful could not be drawn, since there was no clear pattern to be observed in the tests performed.

6. *Different 'Parallel Shift' approaches versus 'Rotation'* : The test problems were solved by using seven different modifications of the base case algorithm. These modifications were :

(i) $\beta = 100$, (ii) $\beta = 10$, (iii) $\beta = 1$, (iv) $\beta = 0.1$, (v) $\beta = 0.01$

(vi) Fixing $t^0 = 1$ in the initialization step of the algorithm (see Section III) and then using the equation $t^0 = B^0 - (-\beta\xi + c)^T x^0$ to determine the value for β.

(vii) Using the 'Rotation' version of the algorithm (see Section III). In this case, the initial value of β was determined through the equation $t^0 = B^0 - (-\beta_0\xi + c)^T x^0$, with $t^0 = 1$.

The results are shown in Tables 7a and 7b. The higher value of β seem to perform better, but the results cannot be termed conclusive, and so no definite inferences could be drawn from them.

SECTION VI

Conclusions

This final section contains a brief outline of the main findings of the empirical study, and a discussion of the important issues that need to be explored further.

Main findings

The aim of the implementation and empirical testing was to explore certain aspects of the algorithm's behavior. On the basis of the implementation results, we can draw the following conclusions:

(1) *For high values of β (eg, $\beta>1$), this parameter does provide an effective balancing mechanism between the Phase I and Phase II objectives.* This is a very encouraging result, for it implies that the goal of providing this flexibility in the solution procedure has been at least partly realized.

(2) *Starting the algorithm with an artificial initial lower bound instead of a good bound does not cause any significant deterioration in the algorithm's performance.* While the direct application of the algorithm with an artificial bound causes it to perform relatively badly, the modified version described in the previous section performs almost at par with the 'good initial bound' case. In some sense, then, the knowledge of a good initial lower bound is not of much consequence.

(3) *The early feasibility option significantly enhances the performance of the algorithm for problems that have a non-empty interior feasible region.* The last qualification is important for two reasons. Firstly, all the problems in the test library *did* satisfy this assumption (see the problem generation description in Section V), and secondly, our geometric intuition makes us suspect that this result may actually not be true for problems where the feasible regions lie in a 'lower-dimensional' space.

(4) *Higher values of β seem to work better than smaller values.* This deduction must be made with some caution, for the test results relating to this issue cannot be termed conclusive.

(5) *The optimal value of q is higher than $n+1 + \sqrt{n+1}$.* This result is not surprising, for similar conclusions have been reached by researchers who have studied the empirical behavior of other interior point linear programming algorithms.

Open questions for future research

(1) *What is the optimal setting for β?* This question was only answered in part by this study, and further work will have to be done before any firm conclusions can be drawn.

(2) *How is the algorithm's behavior and performance affected when it is applied to a different set of problems?* This study concerned itself with testing the algorithm on a specific set of randomly generated problems. Does the algorithm behave in a similar fashion when applied to problems that have been generated randomly in a different manner, or have been derived from actual applications?

(3) *How does the algorithm perform on "warm start" problems?* A "warm start" problem is derived by perturbing an already-solved linear program, and uses as its initial solution the optimal solution of the original linear program. Such problems are often encountered in practice, and serve as an important class of linear programming problems. The algorithm discussed in this thesis could be applied to solve such linear programs, and it would be interesting to study the empirical behavior of the algorithm on such problems.

(4) *What is the relationship between the size of a problem and the number of iterations taken by the algorithm to solve it?* While the theory can provide us with a worst-case bound of O(nL) on these iterations, an empirical study is required to get some understanding of the "average-case" behavior of the algorithm. Such a study would involve the solving of test problems of a number of different sizes, and an analysis of the subsequent results to

number of different sizes, and an analysis of the subsequent results to determine a functional relationship between the two variables mentioned in the above question.

References

[1] K.M. Anstreicher, " A combined Phase I - Phase II projective algorithm for linear programming", *Mathematical Programming* 43 (1989) 209-223

[2] K.M. Anstreicher, " A combined Phase I - Phase II scaled potential algorithm for linear programming", CORE Discussion Paper 8939, CORE Catholic University of Louvain, Belgium, 1989

[3] R.M.Freund, "Polynomial-time algorithms for linear programming based only on primal scaling and projected gradients of a potential function", *Mathematical Programming* 51 (1991) 203-222

[4] R.M.Freund, " A potential reduction algorithm with user-specified Phase I - Phase II balance, for solving a linear program from an infeasible warm start", Technical Report No. 981, School of OR&IE, Cornell University, Ithaca, NY, 1988.

[5] C.C.Gonzaga, "Polynomial affine algorithms for linear programming", *Mathematical Programming* 49 (1990) 7-21

[6] N. Karmakar, "A new polynomial time algorithm for linear programming", *Combinatorica* 44 (1984) 373-395

[7] M.J.Todd and B. Burrell, "An extension of Karmakar's algorithm for linear programming using dual variables", *Algorithmica* 1 (1986) 409-424

[8] M.J.Todd, "On Anstreicher's combined Phase I - Phase II projective algorithm for linear programming", Technical Report No. 776, School of OR&IE, Cornell University, Ithaca, NY, 1988, to appear in *Mathematical Programming*

[9] M.J.Todd, "Combining phase I and phase II in a potential reduction algorithm for linear programming", Technical Report No. 907, School of

OR&IE, Cornell University, Ithaca, NY, 1990, to appear in *SIAM Journal on Optimization*

[10] M.H. Wagner, "Supply-demand decomposition of the national coal model", *Operations Research* 29 (1981) 1137-1153

[11] Y. Ye, "An $O(n^3L)$ potential reduction algorithm for linear programming", *Mathematical Programming* 50 (1991) 239-258

APPENDIX A

TABLE 1

(The behavior of the ratio $r^k = (c^T x^k - B^k) / \xi^T x^k$ over the sequence of iterations)

Size 50 Problems

Note: The numerical problems mentioned in this section caused the computations over the last few (4-5) iterations to be quite imprecise, and so these iterations have been ignored in the analysis.

The results mentioned below refer to the behavior of the above ratio after the first 2-4 iterations.

b = 100 : In all cases, r^k was between 98 and 100

b = 10 : In 14 of the 15 cases, r^k was between 9.6 and 10

b = 1 : In 12 of the 15 cases, r^k was between 0.9 and 1

b = 0.1 : Much more variation in the results; r^k lay between:

0.095 and 0.1 in 4 cases
0.05 and 0.1 in another 6 cases

For the rest of the problems, r^k did not consistently lie in any reasonable proximity to 0.1

b = 0.01 : r^k lay beween 0.007 and 0.01 in 5 cases

For the rest of the problems, r^k did not consistently lie in any reasonable proximity to 0.1. In many cases, r^k was often negative.

TABLE 2

Base case with good vs. artificial initial lower bounds

Case I: Base case (with good initial lower bound)
Case II: Base case (with artificial initial lower bound)
Case III: Modified version, base case (with artificial initial lower bound)

	No. of iterations for algorithm to converge		
	Case I	**Case II**	**Case III**
Size 50 Problem No.			
1	20	29	17
2	22	46	23
3	*	44	20
4	20	41	21
5	21	45	23
6	25	35	25
7	24	46	28
8	21	31	22
9	29	27	27
10	21	45	20
11	31	39	28
12	21	37	21
13	24	48	23
14	20	34	21
15	26	34	27
Average	**23.2**	**38.7**	**23.1**
St. Deviation	**3.5**	**6.9**	**3.3**

* : These problem instances were ignored since their solutions were significantly affected by numerical problems.

TABLE 3

Generation of good lower bounds
(Base case with artificial initial lower bounds)

Case II: Base case (with artificial initial lower bound)
Case III: Modified version, base case
(with artificial initial lower bound)

No. of iterations taken to generate first good bound

Size 50 Problem No.	Case II	Case III
1	1	1
2	9	3
3	10	2
4	9	1
5	9	2
6	2	1
7	9	5
8	4	1
9	2	1
10	7	2
11	4	1
12	4	1
13	5	1
14	3	1
15	2	1
Average	**5.3**	**1.6**
St. Deviation	**3.2**	**1.1**

TABLE 4

Base case with good vs. artificial initial lower bounds

Case I: Base case (with good initial lower bound)
Case III: Modified version, base case (with artificial initial lower bound)

No. of iterations for algorithm to converge

Size 100 Problem No.	Case I	Case III
1	31	28
2	24	27
3	28	31
4	29	36
5	30	29
6	28	28
7	30	29
8	33	36
9	31	32
10	*	37
11	37	39
12	22	26
13	28	28
14	28	29
15	27	28
Average	**29.0**	**30.9**
St. Deviation	**3.6**	**4.1**

* : These problem instances were ignored since their solutions were significantly affected by numerical problems.

TABLE 5

Base case with different values of q

$q = n+1 + (n+1)^t$

	No. of iterations for algorithm to converge				
t =	**0**	**0.5**	**1**	**2**	**3**
Size 50					
Problem No.					
1	24	20	15	16	*
2	25	22	19	19	21
3	25	21	15	16	15
4	*	20	16	15	14
5	29	21	18	22	21
6	30	25	21	30	*
7	28	24	19	17	17
8	25	21	18	19	17
9	28	29	24	*	*
10	24	21	17	15	15
11	27	31	23	21	22
12	23	21	24	19	19
13	27	24	18	17	17
14	23	20	19	20	15
15	30	26	27	32	*
Average	**26.3**	**23.1**	**19.5**	**19.9**	**17.5**
St. Deviation	**2.5**	**3.4**	**3.6**	**5.2**	**2.8**

* : These problem instances were ignored since their solutions were significantly affected by numerical problems.

TABLE 6a

Base case along with the early feasibility option

I-II: Total number of iterations to converge
II only: Iterations in the Phase II stage only
(after achieving early feasibility)

	No. of iterations for algorithm to converge		
	Base Case	**With E.Feas.**	
Size 50		**I - II**	**II only**
Problem No.			
1	20	17	7
2	22	18	8
3	*	19	7
4	20	19	0
5	21	18	7
6	25	14	7
7	24	15	10
8	21	12	6
9	29	14	11
10	21	15	6
11	31	12	10
12	21	12	9
13	24	14	6
14	20	14	7
15	26	17	3
Average	**23.2**	**15.3**	**6.9**
St. Deviation	**3.5**	**2.5**	**2.8**

* : These problem instances were ignored since their solutions were significantly affected by numerical problems.

TABLE 6b

Base case along with the early feasibility option

I-II: Total number of iterations to converge
II only: Iterations in the Phase II stage only
(after achieving early feasibility)

	No. of iterations for algorithm to converge		
	Base Case	**With E.Feas.**	
Size 100		**I - II**	**II only**
Problem No.			
1	31	12	8
2	24	10	4
3	28	17	8
4	29	22	14
5	30	19	10
6	28	18	9
7	30	21	12
8	33	16	10
9	31	14	11
10	*	38	8
11	37	16	13
12	22	11	6
13	28	21	6
14	28	17	10
15	27	18	10
Average	**29.0**	**18.0**	**9.3**
St. Deviation	**3.6**	**6.6**	**2.7**

* : These problem instances were ignored since their solutions were significantly affected by numerical problems.

TABLE 7a

Base Case under seven different scenarios:
'Parallel Shift' with different 's, Constant slack and 'Rotation'
C.S. : 'Constant Slack' option Rtn: 'Rotation' option

Number of iterations for algorithm to converge

	Beta						
	100	10	1	0.1	0.01	C.S.	Rtn
Size 50							
Problem No.							
1	16	17	20	22	24	16	20
2	27	21	22	25	29	23	23
3	*	*	*	*	*	20	22
4	16	16	20	22	25	15	19
5	18	18	21	25	27	17	21
6	19	22	25	27	26	26	32
7	22	23	24	26	32	25	25
8	17	19	21	24	25	23	22
9	19	20	29	29	27	27	27
10	23	21	21	25	29	21	20
11	20	22	31	23	26	24	29
12	23	22	21	22	25	21	21
13	23	21	24	27	28	22	23
14	21	19	20	24	27	20	20
15	21	26	26	26	27	25	27
Average	**20.4**	**20.5**	**23.2**	**24.8**	**26.9**	**21.7**	**23.4**
St. Deviation	**3.1**	**2.6**	**3.5**	**2.1**	**2.1**	**3.6**	**3.8**

* : These problem instances were ignored since their solutions were significantly affected by numerical problems.

TABLE 7b

Base Case under seven different scenarios:
'Parallel Shift' with different values of Beta, Constant slack and 'Rotation'
C.S. : 'Constant Slack' option Rtn: 'Rotation' option

Number of iterations for algorithm to converge

	Beta						
	100	10	1	0.1	0.01	C.S.	Rtn
Size 100							
Problem No.							
1	22	24	31	33	44	32	30
2	32	26	24	28	27	26	25
3	24	26	28	35	36	28	27
4	33	31	29	30	33	27	28
5	34	32	30	*	41	35	28
6	36	27	28	30	34	27	31
7	28	29	30	32	37	26	28
8	26	30	33	31	39	41	36
9	19	28	31	34	35	37	29
10	*	*	*	*	*	20	27
11	32	30	37	36	32	34	36
12	25	19	22	23	25	24	23
13	21	22	28	32	36	21	28
14	31	25	28	30	32	29	36
15	31	25	27	31	36	26	29
Average	**28.1**	**26.7**	**29.0**	**31.2**	**34.8**	**28.9**	**29.4**
St. Deviation	**23.4**	**6.3**	**9.9**	**11.7**	**12.3**	**6.1**	**3.6**

* : These problem instances were ignored since their solutions were significantly affected by numerical problems.

APPENDIX B

The following computer programs, coded in MATLAB, are included in this appendix:

1. PLP : This program includes the Phase I - Phase II algorithm that forms the subject of this paper.

2. PLPPHASEII : This program solves the Phase II linear programming problem, and uses the algorithm described in Freund [3].

3. PPROCESS: This program takes a problem instance (of an LP in standard form), and converts it to a problem in the required format, feeding the resulting problem data into the program TESTRUN.

4. TESTRUN: This program calls repeatedly on the solver LP, each time with the same problem, but with different settings of the algorithmic parameters.

5. FD : This program computes the solution to a Fraley dual problem.

6. FM : This program computes the solution to a two-variable LP, by using a Fourier-Motzkin approach.

```
function [x,k,kk,error] =      plp
                  (A,b,c,y,x1,K,B,i,f,l,p,q,file,fileratios,changeq)

%interior point combined Phase I - Phase II lp code
%
% i is an indicator for which line-updating procedure is to be used
% (i=0 for constant slack, i=1 for parallel shift, and i=2 for fixed point)

% f indicates whether to use the quadratic equation or the fraley dual method
% for updating the lower bound (f=0 for quad.eq., f=1 for fraley dual)


%INITIALIZATION
%
n = max(size(c));
m = max(size(b));
xxx(1:n) = x1;
xxx(n+1) = K;
x = xxx';
alpha = (B- c'*x(1:n) - K)/ (y'*x(1:n));
L = alpha*y + c;

        kk=0;
%         'kk' is the iteration counter for the phase II problem when the
%       'p=1' option is used.

r = .8;
k = 0
% k is the iteration counter
tol = 10^(-3);
fprintf(file,'tolerance  =  %7.5g\n',tol)


  while y'*x(1:n) > tol    | abs(c'*x(1:n)-B) > tol*max(1,abs(c'*x(1:n)))

fprintf(file,'(%5.4g,',  (c'*x(1:n)-B)/(y'*x(1:n))     )
fprintf(file,'  %5.4g)',  B)

      if y'*x(1:n) < tol
           A2 = [A ;y' ];
            b2 = [b;y'*x(1:n)];
             Bound = B + xfdnew*(y'*x(1:n));

      fprintf(file,'\n Phase I is within tolerance, so I am into PhaseII!\n')
      fprintf(file,'  %5.4g (old),  %5.4g (new)\n',  c'*x(1:n),c'*xfeas)

                [xphaseII,kk] = plpphaseII (A2,b2,c,x(1:n),Bound,f,l,changeq);
           x = [xphaseII;0];
           k = k + kk ;
           return
      end

 k = k+1;
```

```
        if k==50
            return
        end

% PROJECTION AND RESCALING

        X = diag(x(1:n));
        Yy = X*y;
        Aa = A*X;
        cc = X*c;
        Ll = alpha*Yy + cc;
        e = ones(Yy);
        yp = Proj(Aa,Yy);
        ep = Proj(Aa, e);
        cp = Proj(Aa,cc);
        Lp = alpha*yp + cp;

if f == 1

% SOLVE THE FRALEY DUAL

        cfd(1) = -(yp'*e) + (e'*Yy);
        cfd(2) = ep'*ep - n;
        Afd = yp;
        Bfd = e - cp;
        bfd = cp;

        [xfd, yfd, ee] = fd (Afd, Bfd, bfd, cfd);

        if ee == 2

             error = 'fraley is unbounded in lpnew'

% fd unbounded implies LP is infeasible

            return

        elseif ee == 1

% if ee were = 0 then the fd would be infeasible, so we'd ignore it


            bound = - cfd(1)*xfd - cfd(2)*yfd - cp'*ep + cc'*e;

          if bound > B
              xfdnew = xfd;
        if i == 0
                    alpha = (bound - c'*x(1:n) - x(n+1))/ (y'*x(1:n));
              L = alpha*y + c;
             Ll = alpha*Yy + cc;
             Lp = alpha*yp + cp;
```

```
            elseif i == 1

                x(n+1) = x(n+1) + bound - B;

            elseif i == 2

                a1 = 1/(y'*x1);
                b1 = (-K-c'*x1)/(y'*x1);
                a2 = 1 - (y'*x(1:n))/(y'*x1);
                b2 = (K+c'*x1)*(y'*x(1:n))/(y'*x1) - c'*x(1:n);

                x(n+1) = a2*bound + b2;
                alpha = a1*bound + b1;
                L = alpha*y + c;
                Ll = alpha*Yy + cc;
            Lp = alpha*yp + cp;
            end

        B = bound;


    end

    end
end


%-----------------------------------------------------------------------
% COMPUTE d (IN RESCALED SPACE)
%-----------------------------------------------------------------------
    gdummy(1:n) = (q/(y'*x(1:n)))*Yy - ones(Yy);
    gdummy(n+1) = -1;
    g = gdummy';

    Ad1 = [Aa zeros(b);Ll' x(n+1)];
    d1 = Proj(Ad1, g);

    if Yy'*d1(1:n) >= 0
        d = d1;
    else
      Ad2 = [Ad1;Yy' 0];
      d2 = Proj(Ad2, g);
      d = d2;
    end

%-----------------------------------------------------------------------
% PRIMAL OR DUAL UPDATE (NO DUAL UPDATE IN CASE OF FD)
%-----------------------------------------------------------------------

 if p == 1
    dd = X*d(1:n);
    feas = (y'*x(1:n))/ (y'*dd);
    xfeas = x(1:n) - feas*dd;
    chk = xfeas >= zeros(c);
```

```
      if ones(c)'*chk == n
         eee = eye(n);
         for j = 1:n
            if y == eee(1:n,j)
% then we need to eliminate the jth variable, which'll always be 0
%               for jj = 1:n
%                  if jj < j
%                     A2(1:m,jj) = A(1:m,jj);
%                     c2(jj) = c(jj);
%                     xf(jj) = xfeas(jj);
%                  elseif jj > j
%                     A2(1:m,jj-1) = A(1:m,jj);
%                     c2(jj-1) = c(jj);
%                     xf(jj-1) = xfeas(jj);
%                  end % of the " if " loop
%               end % of the jj loop
%               c2 = c2';
%               xf = xf';

      fprintf(file,'\n I am going into phaseII!\n')
      fprintf(file,'Change in objective value: ')
      fprintf(file,' %5.4g (old), %5.4g (new)\n', c'*x(1:n),c'*xfeas)


               A(:,j) = [];
               c(j,:) = [];
               xfeas(j,:) = [];
                 [xphaseII,kk] = plpphaseII (A,b,c,xfeas,B,f,l,changeq);


% Now we need to restore the jth variable

                 x = [xphaseII(1:j-1);0;xphaseII(j:n);0];
               k = k + kk;
                return
            end % of the " if y==eye..." loop
         end % of the j loop

% you still want to make sure that xfeas has no zero components; if there
% are some, too bad

         chk2 = xfeas > zeros(c)+tol;

         if ones(c)'*chk2 == n
          A2 = [A;y'];
          b2 = [b;0];

      fprintf(file,'\n I am going into phaseII!\n')
      fprintf(file,'Change in objective value: ')
      fprintf(file,' %5.4g (old), %5.4g (new)\n', c'*x(1:n),c'*xfeas)

               [xphaseII,kk] = plpphaseII (A2,b2,c,xfeas,B,f,l,changeq);
            x = [xphaseII;0];
```

```
            k = k + kk ;
            return
        else
              error = 'some x component went to 0 on shooting to feasibility'
           fprintf(file,'some x component went to 0 on shooting to feasibility\n')
            return
        end
      end % of the " if ones(c)'*chk == n" loop

  end % of the " if p = 1 " loop



      if norm(d) >= r-tol
         if l==1
% now for a line search

%       first, find an upper bound for a, the step length

      tt = 0;
      upper = [];
      for t = 1:n+1

        if d(t,1) > 0
            tt = tt + 1;
            upper(tt) = 1/d(t,1);
        end
      end

      if Yy'*d(1:n) > 0
         tt = tt + 1;
           upper(tt) = (Yy'*e)/(Yy'*d(1:n));
      end

      if tt > 0
          upbd = min(upper);

      else
          keyboard
          return
      end
% NOW WE HAVE TO EXPLORE a IN THE RANGE [0, upbd]

      lwbd = 0;
      a = (upbd + lwbd)/2;

      da = d./([e;1] - a*d);
      fa = -q*(Yy'*d(1:n))/(Yy'*e - a*Yy'*d(1:n)) + sum(da);

      j = 0;
      while abs(fa) > 10^(-3)

        j = j+1;
        if fa < 0
```

```
            lwbd = a;
        else
            upbd = a;
        end

        a = (upbd + lwbd)/2        ;
       da = d./([e;1] - a*d);
          fa = -q*(Yy'*d(1:n))/(Yy'*e - a*Yy'*d(1:n)) + sum(da);

% if j = 10, you wanna end
        if j == 10;
          fa = 0;
        end
      end

% at the end of it all, you've got your step size 'a'

      elseif l==0
          stepsize = 1 + norm(d) - ((1+2*norm(d))^.5)
          a = stepsize/norm(d);
      end
%       ... of the "if l=1 " loop

            x = x - a*diag(x)*d;

% make sure the step shouldn't be a '+'; also the rescaling of d
  else

        if f==1
            error = 'fraley yields small d in lpnew'
              return

        elseif f==0

           gp = (q/(y'*x(1:n)))*yp - ep;
% Solve the quadratic for updating B
        if i == 0

% CONSTANT SLACK CASE

      cbar1 = [ gp', -1];
      wbar1 = [ yp',  0];
      vbar1 = [ cp', x(n+1)];

      [a1,a2,a3] = qq (cbar1', wbar1', vbar1', r);


% updating L

            alpha   = max(roots([a1,a2,a3]));
            L = c + alpha*y;
            B = L'*x(1:n) + x(n+1);
% PARALLEL SHIFT CASE
```

```
            elseif i == 1

        cbar1 = [ gp', -1];
        wbar1 = [ zeros(gp)', 1];
        vbar1 = [ Lp', 0];

        [a1,a2,a3] = qq (cbar1', wbar1', vbar1', r);

         B = B + max(roots([a1,a2,a3])) - x(n+1);
        x(n+1) = max(roots([a1,a2,a3]));

% I'm assuming you want the bigger root since it should correspond to
% the bigger of the B' s.

        pAd1 = [Aa zeros(b);Ll' x(n+1)];
        pd1 = Proj(pAd1,g);

%           pcccp = norm(pd1);
        if Yy'*pd1(1:n) <  0

% check: should it be > instead?

% case of d2 , parallel shift:
%

                RobA = [Aa, zeros(b) ; Yy' 0];
                 cbar = Proj(RobA, g);
                 wbar = Proj(RobA, [zeros(gp) ; 1] );
                vbar = Proj(RobA, [Ll; 0] );

                 [a1,a2,a3] = qq(cbar, wbar, vbar, r);

                 B = B + max(roots([a1,a2,a3])) - x(n+1);
                  x(n+1) = max(roots([a1,a2,a3]));
%                pA2 = [RobA; Ll', x(n+1)];
%                 pd2 = Proj(pA2,g);
            end

          elseif i==2

% FIXED POINT CASE; to be ignored at present

        a1 = 1/(y'*x1);
        b1 = (-K-c'*x1)/(y'*x1);
        a2 = 1 - (y'*x(1:n))/(y'*x1);
        b2 = (K+c'*x1)*(y'*x(1:n))/(y'*x1) - c'*x(1:n);

        a3 = a2/a1;
        b3 = b2 - (a2*b1/a1);

% I am assuming that a higher alpha corresponds to a higher B; check

% check the a1, a2, b1, b2 above
```

```
        vbar = [ cp+ b1*yp; b2];
        wbar = [ a1*yp ; a2];
        cbar = [ gp ; -1];

        [aa1, aa2, aa3] = qq(cbar, wbar, vbar, r);
        B   =   max(roots([aa1,aa2,aa3]));

%       RobA2 = [Aa zeros(b); vbar' + NewB*wbar' ];

%       Robd2 = Proj(RobA2, g);
%       theta = -(cbar'*(vbar + NewB*wbar))/(norm(vbar + NewB*wbar)^2);
%         Robd22 = cbar + theta*vbar + theta*NewB*wbar;

        x(n+1) = a2*B + b2;
        alpha = a1*B + b1;
        L = alpha*y + c;
        Ll = X*L;
        Anew = [Aa zeros(b); Ll' x(n+1)];

        d1new = Proj(Anew,g);

        if Yy'*d1new(1:n) < 0

% put in the d2 stuff here


                RobA = [Aa zeros(b);Yy' 0];

                vbar = Proj(RobA, [cc+b1*Yy; b2] );
                wbar= Proj(RobA, [a1*Yy; a2]);
                cbar = Proj(RobA,g);
                [aa1, aa2, aa3]  = qq(cbar, wbar, vbar, r);
                B   =    max(roots([aa1,aa2,aa3]));
                x(n+1) = a2*B + b2;
                alpha = a1*B + b1;
                L = alpha*y + c;

        end



          end
%           ... of the " if i= 0 or 1 or 2 "  loop

        end
%         ... of the" if f= 0 or 1 " loop

%----------------------------------------
%     plot([0,x(1)],[B,B-alpha*x(1)])
%      pause(.9)
%counter=1
%----------------------------------------
    end
```

```
%          ... of the " if norm(d) > r or otherwise ..." loop
   end
%  ... of the " while   y'*x(1:n) ...   " loop
        return
```

```
function [x, k] = plpphaseII (A,b,c,xfeas,B,f,l,changeq)

% This function solves the problem:

%              min c'x st Ax=b, x>=0

% starting form the initial feasible point xfeas and a lower bound B
% on the optimal solution value. It uses a projected gradient -
% potential reduction approach.

%INITIALIZATION
%
n = max(size(c));


      if changeq == 1
            q = (n+1);
      elseif changeq == 2
            q = n+ (n^.5);
      elseif changeq == 3
            q = n + n;
      elseif changeq == 4
            q = n +(n^2);
      elseif changeq == 5
            q = n +((n)^3);
      end



x = xfeas;

r = .8;
k = 0;
% k is the iteration counter

tol = 10^(-3);

while (c'*x - B) > tol*max(1,abs(c'*x(1:n)))
k = k+1;

      if k==50
            return
        end


% RESCALING AND PROJECTING

      e = ones(c);
      X = diag(x);
      Aa = A*X;
      cc = X*c;
      ep = Proj(Aa,e);
      cp = Proj(Aa,cc);
```

```
if f == 1

% SOLVE THE ONE-DIMENSIONAL VERSION OF THE FRALEY DUAL

% You wanna solve:  max (-e'cp + cc'e) + (n - e'ep)lambda
%                   s.t. lambda(e - ep) <= cp


ii = 0;
jj = 0;
II = [];
JJ = [];


for tt = 1:n

   if (e(tt) - ep(tt)) > 0
       ii = ii+1;
       II(ii)  =  cp(tt)/(e(tt)-ep(tt));
   elseif (e(tt) - ep(tt)) < 0
       jj = jj + 1;
       JJ(jj)  =  cp(tt)/(e(tt)-ep(tt));
   end

end

% Now, II contains a list of the upper bounds on lambda, and JJ contains
% a list of the lower bounds

if ii == 0
   if (n - e'*ep) > 0
       e = 2
       return
   end
end

% The above case corresponds to the dual being unbounded, so that the lp
% is infeasible; this is a screw up, since you had a feasible solution
% to the lp to start with!

if jj ~= 0
    minlambda = max(JJ);
    maxlambda = min(II);

    if maxlambda >= minlambda
%       then the dual is feasible
        lambda = maxlambda;
        bound = (-e'*cp + cc'*e) + (n - e'*ep)*lambda;
   else
        bound = B;
   end
else
    lambda = min(II);
```

```
            bound = (-e'*cp + cc'*e) + (n - e'*ep)*lambda;

        end
        if B < bound
              ch = 1;
        else
              ch=0;
        end
        B = max(B, bound);



        end % of the " if f = 1" loop

%-------------------------------------------------------------------
% NOW TO COMPUTE d (in rescaled space)
%-------------------------------------------------------------------

        g = (q/(c'*x - B))*cc - e;

        d = (q/(c'*x - B))*cp - ep;

        if cc'*d < 0
           d = Proj([Aa;cc'],g);
        end

        if norm(d) > r-tol
           if l ==1

% now for a line search

%          first, find an upper bound for a, the step length

      tt = 0;
        upper = [];

       for t = 1:n

         if d(t,1) > 0
            tt = tt + 1;
               upper(tt) = 1/d(t,1);
         end
       end


      if tt > 0
            upbd = min(upper);

       else
error = 'error in linesearch in lpphaseII'
             return
       end
% NOW WE HAVE TO EXPLORE a IN THE RANGE [0, upbd]
```

```
      lwbd = 0;
       a = (upbd + lwbd)/2;

       da = d./(e - a*d);
         fa = -q*(cc'*d)/(cc'*e - a*cc'*d - B) + sum(da);

     j = 0;
       while abs(fa) > 10^(-3)

       j = j+1;
       if fa < 0
           lwbd = a;
       else
           upbd = a;
       end

         a = (upbd + lwbd)/2  ;
         da = d./(e - a*d);
           fa = -q*(cc'*d)/(cc'*e - a*cc'*d - B) + sum(da);

% if j = 10, you wanna end
       if j == 10;
          fa = 0;
       end
     end

% at the end of it all, you've got your step size 'a'

     elseif l==0
   stepsize = 1 + norm(d) - ((1+2*norm(d))^.5) ;
        a = stepsize/norm(d);
     end
%        ... of the "if l=1 " loop


       x = x - a*diag(x)*d;

     else

       if f==1
              error = 'fraley in lpphaseII yields small d'
       fprintf(file,'\nfraley in lpphaseII yields small d\n')
       elseif f==0
%             Solve the quadratic for updating B

            a1 = r*r - (ep'*ep);
            a2 = 2*q*cp'*ep;
            a3 = -q*q*(cp'*cp);


            delta = min(roots([a1,a2,a3]));
            if delta < 0
               delta = max(roots([a1,a2,a3]));
            end
```

```
%              check that norm d(delta) = r


               B = cc'*e - delta;
          end
%              ... of the "if f = 0 or 1 " loop

%-------------------------------------------------------------
%%% plot([0,5],[B,B])
%%% pause(.9)
%-------------------------------------------------------------
       end
%           ... of the " if norm(d) > r or < r " loop

   end
% of the initial " while (c'*x - B) > tol"   loop
return
```

```
function   pprocess   (n,k)

% This program generates k random problems of size (n/2,n),  processes
% them to get them into the desired format, and feeds them to a solution
% and output module.

% Code modified on Oct 23 to accomodate the program Problems


if n ~= 10
  if n~=50
       if n ~= 100
            error = 'n should be 10,50 or 100!!'
            return
       end
  end
end


m  =  round(n/2);


 for counter = 1:k

% "counter" is the number of random problems to be worked on.

%       rand('normal')
%        A = rand(m,n);
%       while rank(A) < m
%          A = rand(m,n);
%       end    % ...of the 'while..' loop

%       rand('uniform')
%       x  =  rand(n,1);
%       c  =  rand(n,1);



       [A,x,c]  =  problems(counter,n);



       b = A*x;
       x1 = A\b;

       [h]  =  geth(x1);

% Now we have A, b, x1, and h such that Ax1 = b, x1 + theta*h > 0 for
% some theta > 0

       if h == 0
```

```
% then you are feasible
% PUT IN SOMETHING HERE!


        end

        H = - A*h;

% Now to find a j for which H(j) ~= 0

        j = 1;
        while H(j) == 0
          j = j+1;
        end

% theta = (1/H(j))*(b(j) - A(j,1)y(1) - A(j,2)y(2) - .....)


        AAA = A;
        bbb = b;
        A(j,:) = [];
        b(j) = [];


        for jj = 1:m-1

          if jj<j

            for i = 1:n
                AA(jj,i) = A(jj,i) - (H(jj)/H(j))*AAA(j,i);
            end

              bb(jj) = b(jj) - (H(jj)/H(j))*bbb(j);


          else

            for i = 1:n
                AA(jj,i) = A(jj,i) - (H(jj+1)/H(j))*AAA(j,i);
            end

              bb(jj) = b(jj) - (H(jj+1)/H(j))*bbb(j);

          end
        end

% Make sure bb is a column vector

[rowbb,colbb] = size(bb);
if rowbb == 1
        bb = bb';
end
```

```
        for i = 1:n
            cc(i) = c(i) + (AAA(j,i)/H(j))*(c'*h);
        end

% Make sure that cc is a column vector

          [rowcc,colcc] = size(cc);
        if rowcc == 1
            cc = cc';
        end

          lb = (bbb(j)/H(j))*(c'*h);

% Now for the Yy = u constraint (that was derived from the 'theta = 0'
%                                                    constraint)

        Y = A(j,:)'/H(j);
        u = b(j)/H(j);

        if u ~= 0

% Now to find a k for which bb(k) ~= 0

            k = 1;
              while bb(k) == 0
                k = k+1;
              end

            alpha = bb(k)/u;

            Y = AA(k,:)' - alpha*Y;

        end   %.. of the 'if u ~= 0....'   loop

% Finding the initial point y1

        theta = 0;
        for i = 1:n

            if h(i) ~=0
                theta = max(theta, -x1(i)/h(i));
            end

        end

        theta = theta + 1;

        y1 = x1 + theta*h;

        if Y'*y1 < 0
          Y = -Y;

        else
          if Y'*y1 == 0
```

```
            err = 1;
            error = 'The initial solution is feasible';
        end
    end

% At this point you have processed the original randomly generated problem:

%       min cx st Ax = b, x >=0,

%            with initial point x1 (st Ax1 = b)

%            and lower bound 0

% to yield the following problem in the desired format:

%       min cc'y st AAy = bb, Yy = 0, y>=0,

%            with initial point y1 (st AAy1 = bb, y1>=0, but Yy>0)

%            and with lower bound lb

    ptestrun2(AA,bb,cc,Y,y1,lb,counter,h)

end     % ... of the 'counter = ...'  master loop

return
```

```
function   testrun(A,b,c,Y,xx,lb,   counter)
%
% This routine takes the data A,b,c,Y,x,lb from the function 'process' and
% solves a number of lp's using 'lpnew'. The output is fed into the
%  file 'output.testrun2', using the routine 'output'.



n = max(size(c));
file = 'testing';

number = 0;
% number keeps count of the number of lps solved in testrun2,
%                                    and is part of the output


% LINE SEARCH : this option is always on
      l=1;
%---------------------------------------


% EARLY FEASIBILITY

 for  p=0:1

%-------------------------------------------------------



% LINE SHIFT

 i=1

%_____________________________________________________________________



% FRALEY DUAL
       f=1;
%for f = 0:1
%-----------------------------------


% LOWER BOUND
for  bound=1:2
if  bound==1
B = lb;
else
B = -10000;
end
%-------------------------------------------------
```

```
% SLACK
K=1;
%----------------------------------------



% q : This takes 5 values

%                       for changeq = 1:5
%
%                           if changeq == 1
%                                 q = (n+2);
%                           elseif changeq == 2
%                                 q = n+1+ ((n+1)^.5);
%                           elseif changeq == 3
%                                 q = n+1 + n+1;
%                           elseif changeq == 4
%                                 q = n+1 +((n+1)^2);
%                       elseif changeq == 5
%                                 q = n+1 +((n+1)^4);
%                           end
%

      for changeq=1:2

      if changeq == 1
            q = n+1+ ((n+1)^.5);
      else
            q = n+1 +((n+1)^3);
      end
%_____________________________________________________________________


      fprintf(file,'Here are the (duality gap)/(feasibility gap)')
      fprintf(file,' and the B(lb) numbers: (ratio, B)\n')
      fprintf('ratios', '\ncounter = %3.0f, number = %3.0f\n',counter,number)

      [x,k,kk,error] = lp(A,b,c,Y,xx,K,B,i,f,l,p,q,file);

      number = number + 1;

      output(file,K,B,c,Y,x(1:n),i,p,f,q,k,kk,error,number,counter)
      output2(fileq,k,kk,counter,number,changeq,n,i,p,B)


% Now for a bunch of 'end' statements for all the loops
                  end % q
%                 end % K
            end % B
%       end % f
%     end % i
```

```
  end % p
% end % l

return
```

```
function [x,y,e] = fd(A,B,b,c)
% solves the fraley dual problem : min c(1)x + c(2) y      s.t.
%                                  Ax + By <= b

      if c(1) ~= 0

% then z = c(1)x + c(2)y implies x = .....

            A1 = (1/c(1))*A;
            B1 = B - (c(2)/c(1))*A;
            [z,y,e]= fm(A1,B1,b,0);


            if e~=1
% then the lp is infeasible or unbounded
             return
            end

            x = (1/c(1))*z - (c(2)/c(1))*y;
            e=1;
            return
      elseif c(1)==0
            if c(2)>=0
                [y,x,e]= fm(B,A,b,0);

            elseif c(2)<0
                [y,x,e]= fm(B,A,b,1);

            end
      end
```

```
function  [x,y,e]  =  fm(A,B,b,i)
%
% this function solves min (if i=0) or max (if i=1) x, s.t. Ax+By <= b
% where A, B and b are column vectors
%
  m = max(size(A));

 i1 = 0;
 i2 = 0;
 j1 = 0;
 j2 = 0;

 for t = 1:m

       if B(t) > 0

% then y <= b(t)/B(t) - (A(t)/B(t))*x

              i1=i1+1;
              I1(i1,1) = b(t)/B(t);
              I1(i1,2) = -A(t)/B(t);

       elseif B(t) < 0

% then y >= b(t)/B(t) - (A(t)/B(t))*x

              i2 = i2+1;
              I2(i2,1) = b(t)/B(t);
              I2(i2,2) = -A(t)/B(t);
       elseif B(t) == 0
              if A(t) > 0
                     j1=j1+1;
                     J1(j1) = b(t)/A(t);

% J1 = set of upper bounds on x

              elseif A(t) < 0
                     j2=j2+1;
                     J2(j2) = b(t)/A(t);

% J2 = set of lower bounds on x

              elseif A(t) == 0
                     if b(t) < 0
                            e = 0;
                            return
                     end
% e = 0 implies problem is infeasible
%
              end
       end
  end
%
% Now, size of I1 = (i1,2), size of I2 = (i2,2), size of J1 = j1, etc.
```

```
%
  for t=1:i1
        for w=1:i2
%
% we have I2(w,1) + I2(w,2)x <= y <= I1(t,1) + I1(t,2)x
% or [ I2(w,2) - I1 (t,2) ]*x <= I1(t,1) - I2(w,1)
%
                if I2(w,2)-I1(t,2) > 0
                        j1=j1+1;
                        J1(j1) = (I1(t,1)-I2(w,1))/(I2(w,2)-I1(t,2));
                elseif I2(w,2)-I1(t,2) < 0
                        j2=j2+1;
                        J2(j2) = (I1(t,1)-I2(w,1))/(I2(w,2)-I1(t,2));
                elseif I2(w,2)-I1(t,2) == 0
                        if I2(w,1) > I1(t,1)
                                e = 0;
                                return
                        end
% check the above end, as well as if some other stuff is needed in this
% last part
                end
        end
  end
%
% Now we have all lower bounds on x in J2, and all upper bounds in J1
%
 if j1~= 0
        if j2 ~= 0
                if max(J2) > min(J1)
                        e = 0;
                        return
                end
        end
  end

 if i == 0
        if j2 == 0
                e = 2;

% e = 2 implies that the problem is unbounded
                return
        else
                x = max(J2);
        end
  elseif i == 1
        if j1 == 0
                e = 2;
                return
        else
                x = min(J1);
        end
  end

% Now to find y
```

```
 for t = 1:i1
       IX1(t) = I1(t,1) + I1(t,2)*x;
 end

 for t = 1:i2
       IX2(t) = I2(t,1) + I2(t,2)*x;
 end

% Now IX1 = set of upper bounds on y and
%      IX2 = set of lower bounds on y

% Actually, you don't need IX2 !!

  y = min(IX1);
% check if this is OK, or if one still needs to check for infeasibility

 e = 1;
 end
```