Note on Implementing the New Sphere Method for LP Using Matrix Inversions Sparingly

Katta G. Murty
Department of Industrial and Operations Engineering,
University of Michigan,
Ann Arbor, MI 48109-2117, USA
Phone: 734-763-3513,
Fax: 734-764-3451
murty@umich.edu
www-personal.engin.umich.edu/˜ murty

Mohammad R. Oskoorouchi
Department of Information Systems and Operations Management,
College of Business Administration,
California State University San Marcos,
San Marcos, CA 92096-0001, USA,
moskooro@csusm.edu
www.csusm.edu/oskoorouchi

January 2008, revised July 2008

Abstract

A new IPM (Interior Point Method) for LPs has been discussed in [9, 10] based on a centering step that attempts to maximize the radius of the inscribed sphere with center on the current objective plane, and using descent directions derived without using matrix inversions. The method is a descent method and may be called the **sphere method for LP**. In contrast to all the existing IPMs which involve heavy matrix inversions in each step, an advantage of the new method is that it can be implemented with no matrix inversions, or using them only sparingly. We discuss various techniques for implementing this method. These implementations offer the prospect of extending the superior performance of existing software systems for LP, to models that do not have the property of being very sparse.

Key words: Linear programming (LP), interior point methods (IPMs), solving LPs without matrix inversions, ball center of a polytope, ball center on the objective plane, descent directions.

1 Introduction

The celebrated simplex method, and the interior point methods (IPMs) developed subsequent to the introduction of Karmarkar's projective scaling method in 1984, are currently the main algorithms used in software for solving linear programs (LPs). All these algorithms are based on matrix inversion operations. Also, in every step of these methods, the matrix inversion operations involve every constraint in the problem including any redundant constraints in the model. These matrix inversion operations limit the application of these algorithms in large scale applications to only those in which the coefficient matrix is very sparse.

Real life LP models usually contain quite a few redundant constraints. The process of identifying and eliminating them from the model using the methods available in theory is prohibitively expensive (in fact the effort needed by these methods to identify and weed out all the redundant constraints from an LP model, is many times more than that needed to solve the original model with all the redundant constraints in it), and hence is not practical. Presolvers based on various heuristics are commonly employed in LP solvers to identify and eliminate redundant constraints in the model, they are reported to be effective, albeit not optimal (see Cartis and Gould [2006]). Instead of depending on these presolvers, methods which have the ability to avoid redundant constraints in their computational work by themselves, have a definite advantage over existing methods.

The *density* of an LP (or its coefficient matrix) refers to the percentage of nonzero entries in the data. Problems in which this percentage is small (typically $\leq 1\%$) are said to be *sparse*; as this percentage increases, the problem becomes more and more dense. Many LP models in real world applications tend to be very sparse. Programmers have developed techniques to exploit this sparcity, in matrix inversion operations of the simplex and interior point methods. Using these they produced implementations with reasonable memory requirements that can solve large scale models fast. However, the effectiveness of these techniques fades as the density of the coefficient matrix increases, that's why solving large scale dense LPs is hard by existing methods.

In several important application areas, dense LP models do arise. For example, a typical LP model for a Data Envelopment Analysis (DEA) problem has a coefficient matrix that is essentially 100% dense [1, 3]. Also many LP models arising in location problems, distribution problems, and supply chain problems are typically dense. That's why there is a lot of research being carried out targeting dense LP models. In solving LPs which are not very sparse, it is definitely advantageous to have to deal with smaller number of constraints.

It seems that practitioners are quite content with obtaining solutions not necessarily optimal, but close to being so; but they want a method that can obtain this approximate solution faster than existing methods. In many applications, this requires algorithms that can give good performance on models that may not be very sparse. For this, we need to investigate fast methods that satisfy the following properties:

1. Should be a descent method (i.e., starting with a feasible soultion the method should maintain feasibility throught, and the objective value should improve monotonically in every step)
2. Should be implementable with no matrix inversions, or using matrix inversion opertations only sparingly
3. If matrix inversion operations are used, they should never involve redundant constraints in the model, and should only involve a small subset of constraints selected intelligently.

We believe that the future of algorithmic research in LP is in this area.

The **sphere method** that we will discuss in this paper is an IPM that is a descent method, and in contrast to all other IPMs developed earlier, this method can be implemented with no matrix inversions, or using them only sparingly. Another advantage of this method is that if matrix inversion operations are used in any iteration of the algorithm, only a small subset of constraints (called the *touching set of constraints* in that iteration) will be involved in those matrix inversion operations. Also, amazingly, redundant constraints, if any in the model, never enter into the touching set. So, it offers the prospect of extending the superior performance of existing software systems for LP, to models that do not have the property of being very sparse. So, it seems to be well suited to the goals mentioned above.

The sphere method belongs to the class of predictor-corrector type IPMs, early versions of it are discussed in [9, 10]. In this paper we discuss some improvements in that algorithm, and some techniques useful in implementing it. First we will discuss the concepts used in the algorithm. It considers LP in the form

$$\begin{aligned} \text{minimize} \quad & z(x) = cx \\ \text{subject to} \quad & Ax \geq b \end{aligned} \tag{1}$$

where A is an $m \times n$ data matrix, with a known initial interior feasible solution x^0 (i.e., $Ax^0 > b$). If the set of feasible solutions does not have an interior; or if it does, but an interior feasible solution is not known; we modify the problem with the usual big-M augmentation involving one artificial variable as follows:

$$\begin{aligned} \text{minimize} \quad & z(x) = cx + Mx_0 \\ \text{subject to} \quad & Ax + ex_0 \geq b \\ & x_0 \geq 0 \end{aligned}$$

where e is the column vector of all 1s in R^m, and M is a large positive penalty parameter. ($x^0 = 0$, x_0^0) where x_0^0 is a sufficiently large positive number, gives an initial interior feasible solution to this Phase I problem. This Phase I problem is in the same form as (1), we solve it instead. For other such Phase I strategies, see (Cartis, Gould [2006]).

How to Implement the Algorithm for Solving a General LP ?

The most popular IPM for software implementations is the primal-dual IPM [4 to 8, 11 to 14], because: **(i)** it gives optimum solutions to both the primal and the dual when both have feasible solutions; and **(ii)** it provides a lower bound that serves as an indicator to check how far left to go to reach the optimum. Also, in many algorithms, specifying a good termination condition to be used in practice is not easy . The lower bound in the primal-dual format provides an automatic practical termination condition.

We will show how to convert our algorithm into a primal-dual algorithm for LPs in general form. Consider an LP in general form in which there may be equality constraints on the variables, inequality constraints, and bounds on individual variables. By combining the bounds on individual variables with the inequality constraints, the problem is in the form

$$\begin{array}{rrcl} \text{Minimize} & f\xi & & \\ \text{subject to} & F\xi & = & h \\ & G\xi & \geq & g \end{array} \tag{2}$$

where F is a matrix of order $p \times q$, say. Let π, μ be dual vectors corresponding to the constraints in the two lines in (2). Solving (2) and its dual involves finding a feasible solution to the following system

$$\begin{array}{rcl} F\xi & = & h \\ \pi F + \mu G & = & f \\ (G\xi, \mu, -f\xi + \pi h + \mu g) & \geq & (g, 0, 0) \end{array} \tag{3}$$

.

Solving (3) is the same as solving the LP

$$\begin{array}{l} \text{Minimize} \quad \sum_{i=1}^{p}(F_{i.}\xi - h_i) + \sum_{j=1}^{q}(\pi F_{.j} + \mu G_{.j} - f_j) \text{ subject to} \\ \qquad (F\xi, G\xi, \pi F + \mu G, \mu, -f\xi + \pi h + \mu g) \geq (h, g, f, 0, 0) \end{array} \tag{4}$$

(4) is in same form as (1). Also, since we are applying the algorithm without matrix inversions or using them sparingly, having all these additional constraints over those in the original LP (2) in the model may not make it numerically difficult to handle. If both (2) and its dual have feasible solutions, at the optimum, the objective value in (4) will be 0, so this provides a convenient lower bound to judge how far is left to go. So, in the sequel we will discuss the method for solving the LP in the form (1).

We assume that the rows of A, denoted by $A_{i.}$ for $i = 1$ to m, have been normalized, so $||A_{i.}|| = 1$ ($||.||$ denotes the Euclidean norm) for all $i = 1$ to m; also $||c|| = 1$. We will use the following notation:

$K =$	Set of feasible solutions of (1).
$K^0 =$	$\{x : Ax > b\}$ = interior of K.
$\delta(x) =$	Min$\{A_{i.}x - b_i : i = 1$ to $m\}$, defined for $x \in K^0$, it is the radius of the largest ball inside K with x as its center, since $\lvert\lvert A_{i.}\rvert\rvert = 1$ for all i.
$B(x, \delta(x)) =$	Defined for $x \in K^0$, it is the largest ball inside K with x as its center.
$T(x) =$	Defined for $x \in K^0$, it is the set of all indices i satisfying: $A_{i.}x - b_i =$ Minimum$\{A_{p.}x - b_p : p = 1$ to $m\} = \delta(x)$. The hyperplane $\{x : A_{i.}x = b_i\}$ is a tangent plane to $B(x, \delta(x))$ for each $i \in T(x)$, therefore $T(x)$ is called the **index set of touching constraints in (1)** at $x \in K^0$. See Figure 1.
$t_{\min}, t_{\max} =$	Minimum, maximum values of $z(x)$ over K respectively.
$\delta[t] =$	It is the Maximum$\{\delta(x) : x \in \{x : cx = t\}\}$, i.e., the maximum radius of the ball that can be inscribed inside K with its center restricted to $\{x : cx = t\}$. Notice the difference between $\delta(x)$ defined over K^0; and this $\delta[t]$ defined over the interval $[t_{\min}, t_{\max}]$ of the real line.
t^*	= Value of $t \in [t_{\min}, t_{\max}]$ that maximizes $\delta[t]$.

Each iteration of the sphere algorithm consists of only two steps, a centering step and a descent step. The centering step is a corrector step, it tries to move the current interior feasible solution into another one with higher value for $\delta(x)$ without sacrificing objective quality. The descent step is a predictor step, that results in a strict decrease in objective value.

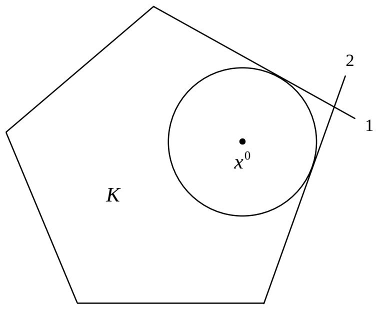

Figure 1: $x^0 \in K^0$, and the ball shown is $B(x^0, \delta(x^0))$, the largest ball inside K with x^0 as center. Facetal hyperplanes of K corresponding to indices 1, 2 are tangent planes to this ball, so $T(x^0) = \{1, 2\}$.

2 The Ball Center of a Polytope as a whole, and Ball Center of a Polytope On a Given Objective Plane; and How to Compute these Ball Centers Approximately

In this section we first give the theoretical definition of the ball center of the given polytope K, and the ball center of K on a given objective plane intersecting K; these are important concepts used in the sphere method. Then we discuss methods for computing these ball centers approximately.

An earlier version of this method was discussed in [9]. In the centering step in that version, the concept of the "center" of a polytope is defined as a point which is the center of a largest radius ball inscribed inside the polytope; i.e., for K it will be a point x that is optimal to the LP (5). But the optimum face for (5) in the x-space may contain multiple points, in that case the description of the algorithm in [9] did not clearly specify which of those multiple optimum points of (5) in the x-space should be selected as the "center" of the polytope K defined by (1). Here, we will specify rules that will make that choice "precise". For this we introduce the concepts of the *ball center of a polytope*, and the *ball center of a polytope on a given objective plane* $\{x : cx = t\}$.

The definition of the "center" of a polytope used in our algorithm is very different from that used in earlier IPMs [4 to 8; 11 to 14]. To distinguish, since the "center" that we use is the center of a largest ball inside the polytope, we will use the word **ball center** of the polytope for the center that we use.

Our definitions guarantee that every polytope K (i.e., a bounded convex polyhedron) has a well defined and unique ball center for the polytope as a whole. Also, when H is a hyperplane having a nonempty intersection with the polytope K, the ball center for K on H is again well defined and unique.

These concepts are well defined only for convex polytopes, and may not really work for unbounded convex polyhedra, we discuss this issue at the end of this section. For this reason, we will first give the definitions under the assumption that the set of feasible solutions of the LP (1), K, is a polytope.

Even though the Conceptual Algorithm discussed in Section 3 is based on the ball centers, in practice in implementations of the algorithm, in every iteration we will only compute the ball center on the current objective plane approximately.

What happens to the LP (1) if we do not know whether its set of feasible solutions is bounded or not, or if we know that it is unbounded? In the unbounded case, even though the concepts of ball centers may not be well defined, the implementation of the algorithm based on the approximate computation of ball centers can be carried out as usual. In this case in one of the descent steps, the step length for the move may turn out to be $+\infty$. That is an indication that the objective value in (1) is unbounded below on its set of feasible solutions; and the algorithm terminates if this happens.

Even though the sphere method uses only ball centers of K on objective planes H, we will also discuss the related concept of the ball center of a polytope as a whole (and how to compute it approximately), becuse it can be used in the approximate computation of the ball center of the polytope on a hyperplane.

The Ball Center of a Polytope

Now we will continue with the definition of the ball center of a whole polytope. A polytope of dimension 1 is a line segment, its ball center is its unique midpoint. See Figure 2.

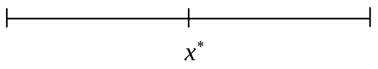

Figure 2: The ball center of a 1-dimensional polytope (a line segment) is its mid-point x^*.

Now consider the polytope K of dimension n represented by (1). Its ball center x^* is a point in K^0 which is the center of a largest radius ball inscribed inside K. Letting $\delta^* = \delta(x^*)$, (x^*, δ^*) is therefore an optimum solution of the LP

$$\begin{array}{rll} \text{Maximize} & \delta & \\ \text{subject to} & \delta \leq A_{i.}x - b_i, & i = 1 \text{ to } m \end{array} \qquad (5)$$

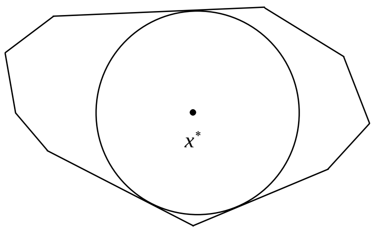

Figure 3: When the largest inscribed ball in K is unique, its center x^* is the ball center of K.

If the optimum solution of this LP is unique, it will be (x^*, δ^*), and x^* is the ball center of K. See Figure 3.

If the optimum solution of (5) is not unique, all of its optimum solutions are of the form (x, δ^*) for $x \in S$, where S is the optimum face of (5) in the x-space. In this case the ball center of K is defined recursively by dimension to be the ball center of the lower dimensional polytope S. This definition guarantees that every polytope has a unique ball center. See Figure 4 for an illustration.

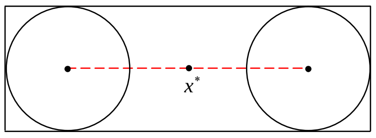

Figure 4: A 2-dimensional polytope K for which the largest inscribed ball is not unique. S, the set of centers of all such balls, the optimum face of (5) in the x-space, is the dashed line segment in this polytope. So here the ball center of K is the ball center of S, which is its mid-point x^*.

Ball Centre for (1), On the Objective Plane $\{x : cx = t\}$ for Given t

Each iteration of the sphere method begins with the current point, which is the interior feasible solution obtained at the end of the previous iteration. Consider Iteration $r+1$, suppose it begins with the current point x^r. Let $cx^r = t$ be the current objective value in (1). The centering step in this iteration tries to find the point $x \in K^0 \cap \{x : cx = t\}$ which maximizes $\delta(x)$, it is an optimum solution of the LP

$$\begin{array}{lrcll} \text{Maximize} & \delta & & & \\ \text{subject to} & \delta & - & A_{i.}x \leq -b_i, & i=1 \text{ to } m \\ & cx & = & t & \end{array} \tag{6}$$

If the optimum solution x for (6) is unique, denote it by $x(t)$, it is called the **ball center for (1) (or for its set of feasible solutions K) on the current objective plane** $\{x : cx = t\}$. In this case, the unique optimum solution of (6) is $(x(t), \delta[t] = \delta(x(t)))$. See Figure 5.

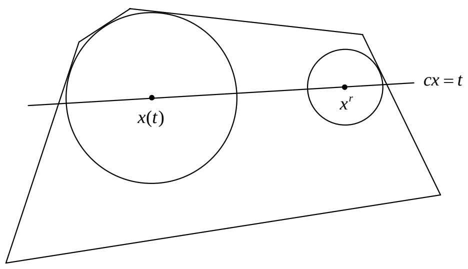

Figure 5: When the optimum solution for (6) is unique, the largest ball inside K with center on the current objective plane $\{x : cx = t\}$ is unique (like here, it is the large ball in the figure), its center is $x(t)$, the ball center for (1) corresponding to the present objective value t.

In general, even though the optimum δ in (6) is always unique, there may be multiple points x which are optimal. So, let $S(t)$ denote the optimum face of (6) in the x-space. In this case, the ball center for (1) on the objective plane $\{x : cx = t\}$ is defined to be the ball center of the polytope $S(t)$ as defined earlier.

This definition guarantees that for each $t \in (t_{\min}, t_{\max})$, the ball center for (1) on the objective plane $\{x : cx = t\}$ is unique.

The Case of Unbounded Convex Polyhedra

Suppose the set of feasible solutions, K of (1) is an unbounded polyhedron. Typically, the radius of a maximum inscribed ball in K will be ∞ (i.e., δ is unbounded above in the LP (5)). Even if the radius of the maximum inscribed ball is finite, the optimum face of (5) in the x-space may be an unbounded polyhedron itself, so the ball center of K is not defined.

Even when K is an unbonded convex polyhedron, the hyperplane H may be such that $K \cap H$ is bounded, i.e., is a polytope; in this case the ball center of

K on H is well defined. If $K \cap H$ is also unbounded, then the ball center of K on H is also not defined.

Computation of Approximate Ball Centers

Several techniques have been discussed in [9, 10] to compute these ball centers approximately. In Sections 2.1, 2.2 we give mathematical descriptions of some techniques used for computing approximately the ball center of the polytope K, and the ball center of K on the objective plane $\{x : cx = t\}$ for given t.

2.1 Approximate Computation of the Ball Center of a Polytope

We consider (5), the problem of computing the ball center of the whole polytope K beginning with an initial interior feasible x^r. We summarize these techniques and discuss ways to implement them.

2.1.1 LSFN, Using Line Search Steps in Facetal Normal Directions

One advantage of this technique is that it needs no matrix inversions. Beginning with the initial $x^{r,0} = x^r$, it generates a sequence of points $x^{r,k}$, $k = 1, 2, \ldots$ along which the radius of the ball δ is strictly increasing.

At the current point $x^{r,k}$, a direction y is called a **profitable direction**, if $\delta(x^{r,k} + \alpha y)$ strictly increases as α increases from 0. [9] has the following result, which makes it easy to check whether any given direction y is profitable at the current point.

Result: A given direction $y \in R^n$ is a profitable direction at the current interior feasible solution $x^{r,k}$ iff $A_{i.}y \geq 0$ for all $i \in T(x^{r,k})$. Also, $x^{r,k}$ is an optimum solution for (5) iff there is no profitable direction at it, i.e., iff the system: $A_{i.}y \geq 0$ for all $i \in T(x^{r,k})$, has no nonzero solution y.

Since the goal in this centering step is to increase the minimum distance of x from each facetal hyperplane of K, the procedure uses only the directions normal to the facetal hyperplanes of K for the line searches, i.e., directions in $\Gamma_1 = \{A_{i.}^T, -A_{i.}^T : i = 1 \text{ to } m\}$. This is the set of directions normal to facetal hyperplanes of K. The procedure continues as long as profitable directions for line search are found in Γ_1, and terminates with the final point as an approximate center of K, which is denoted by $\bar{x}^r$. See Figure 6.

Once a profitable direction y at the current point $x^{r,k}$, has been found, the optimum step length α in this direction that maximizes $\delta(x^{r,k} + \alpha y)$ over $\alpha \geq 0$ is $\bar{\alpha}$, where $(\bar{\delta}, \bar{\alpha})$ is the optimum solution of the 2-variable LP (7).

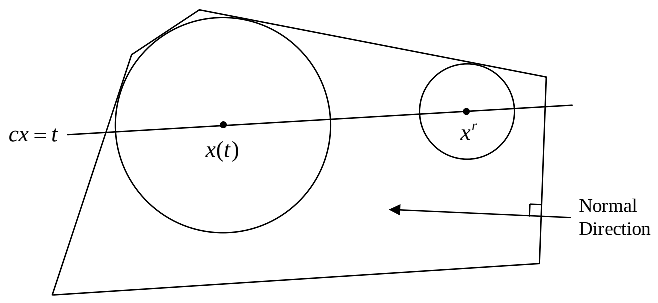

Figure 6: Moving from the current point x^r, in the direction which is the orthogonal projection of the normal to the facet of K on the right, on the objective plane $\{x : cx = t\}$, leads to $x(t)$.

$$\begin{array}{rrcll} \text{Maximize} & \delta & & & \\ \text{subject to} & \delta - \alpha A_{i.}y & \leq & A_{i.}x^{r,k} - b_i & i = 1, \ldots, m \\ & \delta, \alpha \geq 0 & & & \end{array} \tag{7}$$

and $\bar{\delta}$ is the optimum objective value $\delta(x^{r,k} + \bar{\alpha}y)$. So, the line search for the maximum value of δ in the direction y involves solving this 2-variable LP, which can be carried out efficiently (e.g., by the simplex algorithm) as discussed in [9, 10].

In every pivot step of the simplex method, we have to select an entering variable to enter the current basic vector, among all those eligible in that step. In extensive computational research carried out over the last 60 years for implementing the simplex method, programmers have developed strategies to search for eligible variables and select one among them as the entering variable, to make the code for the simplex method work well. Exactly the same strategies can be used to select a profitable direction among the facetal normal directions from Γ_1 to carry out the line search, to make sure that the implementation of this technique works well.

2.1.2 LSCPD, Sequence of Line Search Steps Using Computed Profitable Directions

This technique, discussed in [10], consists of a sequence of at most n line search steps in profitable directions computed by solving a system of linear equations. After each step in the sequence, the index set of touching constraints at the current solution x keeps growing by at least one more constraint; and we will stop the sequence when the set of coefficient vectors of the touching set of constraints becomes linearly dependent. That's why the number of steps in the sequence is at most n. The entire sequence needs a single matrix inversion, carried out in stages adding one row and column to the matrix at a time; so it uses matrix inversion operations sparingly.

The sequence begins with the initial point in the sequence $x^{r,0} = x^r$. When $x^{r,k}$ is the current solution, from the Result discussed in Section 2.1.1, we know that any solution y of the system

$$A_{i.}y = d_i \qquad \text{for all } i \in T(x^{r,k}) \tag{8}$$

where $d = (d_i : \quad i \in T(x^{r,k})) \neq 0$ is any nonnegative vector, is a profitable direction to move at $x^{r,k}$.

We will use (8) with $d = e$, the column vector of all 1s of appropriate dimension; to generate a profitable direction which is a basic solution of (8), to move. Once a profitable direction y is determined, the step length to move in this direction is determined by solving the 2-variable LP (7) as in Section 2.1.1

So, this sequence begins with the initial point $x^{r,k}$ for $k = 0$, solves (8) with $d = e$ for a basic solution. If this system has no solution, then this technique is terminated with the initial point as the center obtained in it.

If it does yield a solution y^0, suppose it is the basic solution with respect to a basic vector y_{B^0} and basis B^0 for the system. If α^0 is the optimum step length maximizing $\delta(x^{r,0} + \alpha y^0)$, notice that as α increases from 0, $A_{i.}(x^{r,0} + \alpha y^0) = A_{i.}x^{r,0} + \alpha$ increases at the same rate as α for all $i \in T(x^{r,0})$. This implies that at the solution $x^{r,1} = x^{r,0} + \alpha^0 y^0$ obtained after this line search step, we will have $T(x^{r,1}) \supset T(x^{r,0})$.

The sequence will now continue the same way with $x^{r,1}$ as the initial solution for the next step. Suppose $T(x^{r,0}) = \{1, ..., s\}$ and $T(x^{r,1}) = \{1, ..., s, s+1\}$. Then the profitable direction y^1 to be used at $x^{r,1}$ is computed by solving the system

$$A_{i.}y = 1 \qquad \text{for all } i \in \{1, ..., s+1\} \tag{9}$$

Let $\mathcal{A}^0 = (B^0 \vdots D^0)$ denote the coefficient matrix of (8) for $k = 0$ with rows $\{A_{i.} : i \in T(x^{r,0})\}$, with columns partitioned into the basic, nonbasic parts with respect to the basic vector y_{B^0} for it; and $(A^{B^0}_{s+1.}, A^{D^0}_{s+1.})$ the corresponding partition of $A_{s+1.}$.

The set of touching constraint coefficient vectors $\{A_{i.} : i \in T(x^{r,1})\}$ is linearly independent iff $A^{D^0}_{s+1.} - A^{B^0}_{s+1.}(B^0)^{-1}D^0 \neq 0$.

Therefore if $A^{D^0}_{s+1.} - A^{B^0}_{s+1.}(B^0)^{-1}D^0 = 0$, terminate the sequence with $x^{r,1}$ as the final center obtained in it. On the other hand if this vector is $\neq 0$, select a nonzero entry in it, suppose it is in the column of the variable y_j, then let $\mathcal{A}_{.j}$ be the column corresponding to y_j in $\mathcal{A}$, the coefficient matrix of (9). Then $y_{B^1} = (y_{B^0}, y_j)$ is a basic vector for (9). The corresponding basis for (9) is

$$B^1 = \begin{pmatrix} B^0 & \vdots & \mathcal{A}_{.j} \\ \cdots & & \cdots \\ A^{B^0}_{s+1.} & \vdots & a_{s+1,j} \end{pmatrix}.$$

where $a_{s+1,j}$ is the coefficient corresponding to y_j in $A_{s+1.}$. Hence

$$(B^1)^{-1} = \begin{pmatrix} P & \vdots & Q \\ \cdots & & \cdots \\ R & \vdots & S \end{pmatrix}$$

where

$$\begin{aligned} S &= 1/(a_{s+1,j} - A^{B^0}_{s+1.}(B^0)^{-1}\mathcal{A}_{.j}) \\ R &= (-A^{B^0}_{s+1.}(B^0)^{-1})/(a_{s+1,j} - A^{B^0}_{s+1.}(B^0)^{-1}\mathcal{A}_{.j}) \\ Q &= -(B^0)^{-1}\mathcal{A}_{.j}S \\ P &= (B^0)^{-1} + QR/S \end{aligned}$$

So, $(B^1)^{-1}$ can be obtained by updating $(B^0)^{-1}$ using the above formulas.

The sequence repeats the same way with $x^{r,1}$, until it terminates at some stage. Thus, in this sequence, whenever system (8) is augmented by a new constraint in a step, the basic vector and basis inverse in this step can be updated quite efficiently for the next step as discussed above.

2.2 Computing an Approximate Ball Center of K on the Current Objective Plane

There are two different ways for computing the ball center of K on the objective plane $\{x : cx = t\}$ for any given value of t. We discuss them separately. Which among the two performs better has to be determined by computational experiments.

2.2.1 Computing the Ball Center on the Objective Plane Directly

Consider the current objective value t, and the current objective plane $\{x : cx = t\}$. The model for finding the ball center on this objective plane is (6); the only difference between this and (5) is the additional constraint $cx = t$ in (6).

So, an approximate solution to (6) can be obtained directly by adopting the line search techniques discussed in Sections 2.1, 2.1.1, 2.1.2 to this situation.

The only change required to adopt the line search techniques LSFN in Section 2.1.1 to this problem is to look for profitable directions from the set

$\Gamma_2 = \{P_{.1}, ..., P_{.m}, -P_{.1}, ..., -P_{.m}\}$, where $P_{.i} = (I - c^T c)A_{i.}^T$, the orthogonal projection of $A_{i.}$ (the direction normal to the facet of K defined by the i-th constraint in (1)) on the hyperplane $\{x : cx = 0\}$, for $i = 1$ to m, instead of the set Γ_1. Since Γ_2 is the set of directions which are orthogonal projections of the directions in Γ_1 on the plane $\{x : cx = 0\}$, any step length from a point in the current objective plane, in a direction from Γ_2, will keep the point on the current objective plane.

To adopt the line search sequence LSCPD discussed in Section 2.1.2 to this problem, right from the initial step in this sequence, we include the additional constraint $cy = 0$ in all systems of the form (8) in the sequence; and continue as discussed in Section 2.1.2.

However, for implementing LSCPD in centering steps in various iterations of the sphere method, we will adopt a slightly different strategy which provides additional advantages. This strategy is discussed in Section 4.1.

2.2.2 Transforming the Problem of Finding An Approximate Ball Center On the Current Objective Plane into that of Finding An Approximate Ball Center of a Polytope

For any given value, t, of the objective function cx, the set of feasible solutions of (1) with this objective value is $K \cap \{x : cx = t\}$, represented by

$$\begin{aligned} Ax &\geq b \\ cx &= t \end{aligned} \tag{10}$$

Take any nonzero entry in c, say c_n. Then in (10), we can use the equality constraint to express the variable x_n as $(t - c_1x_1 - ... - c_{n-1}x_{n-1})/c_n$ in terms of the objective value t and the other variables. Substituting this expression for x_n in all the inequality constraints in (10), we get a representation of $K \cap \{x : cx = t\}$ in terms of the remaining variables $X = (x_1, ..., x_{n-1})^T$ in the form

$$DX \geq d + td^* \tag{11}$$

say. We will denote the set of feasible solutions of (11) for given t by $\mathcal{K}(t)$ in the X-space. Each point $X \in \mathcal{K}(t)$ corresponds to a unique point x in $K \cap \{x : cx = t\}$ through the expression given above for x_n.

Let $X(t)$ denote the approximate ball center for the polytope $\mathcal{K}(t)$ obtained using the line search techniques discussed in Sections 2.1, 2.1.1, 2.1.2. The point in the x-space corresponding to $X(t)$ can be taken as the approximate ball center on the objective plane $\{x : cx = t\}$.

3 The Concept of the Sphere Method

We will now provide a description of the algorithm as discussed in [9]. Iteration 1 begins with the initial interior feasible solution x^0 that is available. We will discuss the steps in the general iteration $r+1$ conceptually.

Iteration $r+1$**:** It begins with x^r, the current point, the interior feasible solution obtained at the end of the previous iteration. Let $t = cx^r$ be the current objective value in (1). Go to the centering step.

Centering step: Starting with the current point x^r, find the ball center for (1) on the current objective plane $\{x : cx = t\}$, which we will denote by $\bar{x}^r$. So, $(\bar{x}^r, \delta(\bar{x}^r))$ is an optimum solution of the LP, (6), defining it.

The reason for getting a ball with the maximum possible radius with center on the current objective plane in the centering step is this: from the center of a ball with radius δ, we can move a step length of at least δ in any direction. Maximizing δ helps to make longer steps towards optimality in the descent steps in each iteration.

The centering problem (6) is itself another LP of the same size as the original LP (1). But it is an LP with a very special structure. For example, it is a parametric right hand side LP with the parameter t. Also, to implement this algorithm for solving (1), an exact solution of (6) is not essential, and we show that the special structure of (6) can be exploited to get a good approximate solution for it fast. So, even though in concept $\bar{x}^r$ is defined to be the ball center of K on the current objective plane, in implementation we will take it as an approximation of this ball center.

Descent step: In all the earlier metods for solving LPs (simplex method, earlier interior point methods), in each iteration only one descent step is taken; and the method begins the next iteration with the point obtained at the end of this descent step.

Among the two steps in each iteration of the sphere method, the centering step is the most expensive. Once the center is computed in an iteration, the effort needed for a descent step in a given descent direction is just one minimum ratio computation, which is cheap in comparison. That's why, in contrast to earlier methods, we carry out descent steps in five different descent directions (all obtained directly without additional computation) in each iteration, and take the best result obtained from all of them as the output of this iteration. In this section, we explain what these five descent directions are.

Let $\bar{x}^r$ denote the approximate ball center on the current objective plane obtained in Iteration $r+1$ of the algorithm. From [9, 10] we get the following descent directions.

$d^1 = -c^T$, negative objective coefficient vector, and

> $d^2 = \bar{x}^r - \bar{x}^{r-1}$ = direction of the path of ball centers being generated

Our computational work has indicated some additional descent directions that give good results too. We discuss these here. Let c^i denote the orthogonal projection of c^T on $\{x : A_{i.}x = 0\}$, i.e.,

$$c^i = (I - A_{i.}(A_{i.})^T)c^T, \quad \text{for } i = 1 \text{ to } m. \tag{12}$$

The directions $-c^i$ for $i \in T(\bar{x}^r)$ are called **GPTC** (gradient projection on touching constraint) directions. We found that carrying out descent steps at $\bar{x}^r$ in each of these GPTC directions leads to good improvements in the objective value. For referencing it, we will denote

> d^3 = direction among the GPTC directions that gives maximum reduction in objective value when the descent step is taken from the center $\bar{x}^r$.

d^3 will only be determined after descent steps in each direction $-c^i$ for $i \in T(\bar{x}^r)$ are carried out from $\bar{x}^r$.

As another descent direction from $\bar{x}^r$, we also try

> $d^4 = (\sum(-c^i : \text{for } i \in T(\bar{x}^r))/|T(\bar{x}^r)|$, the average direction of all the GPTC directions.

Also, for $i \in T(\bar{x}^r)$, let x^{ir} denote the orthogonal projection of the center $\bar{x}^r$ on the touching facetal hyperplane $\{x : A_{i.}x = b_i\}$; it is the point where this facetal hyperplane touches the ball $B(\bar{x}_r, \delta(\bar{x}^r))$. The points x^{ir} for $i \in T(\bar{x}^r)$ are called the **touching points (TPs)** of the ball $B(\bar{x}_r, \delta(\bar{x}^r))$ with its touching facetal hyperplanes of K.

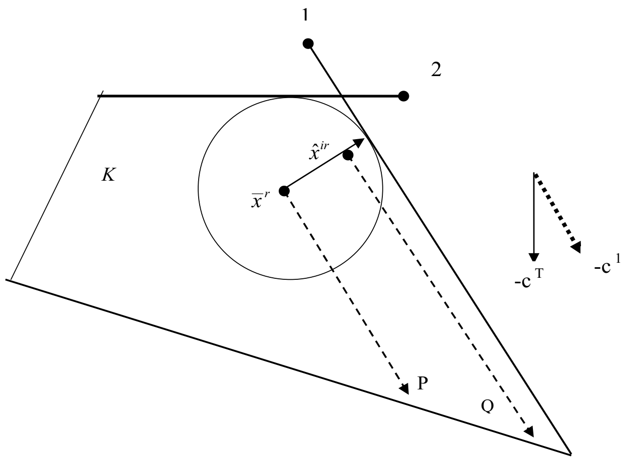

Figure 7: Descent steps in a GPTC direction. Here $\bar{x}^r$ is the current center, $T^(\bar{x}^r) = \{1,2\}$. Directions $-c^T$ pointing down south, $-c^1$ = orthogonal projection of $-c^T$ on facetal hyperplane of constraint 1, are shown. x^{1r} = TP of constraint 1, $\hat{x}^{1r}$ = NTP corresponding to constraint 1. Descent step from $\bar{x}^r$ $[\hat{x}^{1r}]$ in direction $-c^1$ are shown, leading to points P $[Q]$ respectively. Here Q is a much better point than P.

Let $0 < \epsilon < 1$ be a small positive tolerance ($\epsilon = 0.1$ works well). Then for $i \in T(\bar{x}^r)$, the point on the line segment joining $\bar{x}^r$ and x^{ir} close to the TP x^{ir},

$$\hat{x}^{ir} = \epsilon\bar{x}^r + (1-\epsilon)x^{ir}$$

is called the **near touching point (NTP)** corresponding to the tangent plane $\{x : A_{i.}x = b_i\}$ of the ball $B(\bar{x}_r, \delta(\bar{x}^r))$.

We found that carrying out a descent step in the GPTC $-c^i$ from the NTP $\hat{x}^{ir}$, for each $i \in T(\bar{x}^r)$; produces good reductions in objective value, particularly as $\bar{x}^r$ is getting closer to an optimum solution of the original LP. We will denote

> d^5 = the GPTC direction $-c^i$ that gives maximum reduction in objective value when the descent step is taken from the corresponding NTP $\hat{x}^{ir}$.

Each descent step carried out in this iteration requires one minimum ratio computation. For example, consider a descent step from the current center $\bar{x}^r$ in thr descent direction y (i.e., satisfying $cy < 0$). If the step length is λ, the move leads to the point $\bar{x}^r + \lambda y$. Select a small positive number ϵ_1 as the tolerance for minimum $\{A_{i.}x - b_i : i = 1 \text{ to } m\}$ for the point x to be in the interior of K.

Then we will take the step length from $\bar{x}^r$ in the direction y to be: $(-\epsilon_1)$ + (the maximum step length possible while remaining inside K), which is

$$\gamma = \text{minimum}\{\frac{-A_{i.}\bar{x}^r + b_i + \epsilon_1}{A_{i.}y} : i \text{ such that } A_{i.}y < 0\}$$

and then the point obtained at the end of this descent step will be $\bar{x}^r + \gamma y$ if γ is finite.

If $\gamma = \infty$, the objective function $z(x)$ is unbounded below in (1), and $\{\bar{x}^r + \lambda y : \lambda \geq 0\}$ is a feasible half-line along which $z(x)$ diverges to $-\infty$ on K. Terminate the method if this occurs.

4 How To Implement the Algorithm to Solve (1) With an Initial Point $x^0 \in K^0$?

In this section we discuss the work in a general iteration of the sphere method, and how the various steps outlined in Sections 2, 3 are implemented in it for solving the LP (1).

In describing the concept of the sphere method for solving the LP (1) in Section 3, it was mentioned that the aim of the centering step when the objective value is t, is to compute the ball center of K on the current objective plane $\{x : \ cx = t\}$. This requires using line search steps each of which is profitable (i.e., increases $\delta(x)$) while keeping cx unchanged at t. But in practice, if the same step that increases $\delta(x)$ can also decrease the value of cx, that is even better for solving the LP (1). For this reason, in the implementation of the sphere method discussed below, in the centering step we select line search directions which are both profitable, and are also descent directions for cx if possible.

The first iteration begins with the given initial interior feasible solution x^0. We will now discuss the general iteration $r+1$ beginning with the initial interior feasible solution x^r.

In Sections 2.2.1, 2.2.2, we discussed two different strategies to compute the ball center on the current objective plane $\{x : cx = cx^r\}$. In the following Sections 4.1, 4.2, we discuss the implementation based on the strategies in Section 2.2.1, 2.2.2 respectively.

4.1 Implementation by Approximating the Ball Center on Objective Plane Directly

Here we will use the approach of computing the approximate ball center on the current objective plane directly based on the strategy discussed in Section 2.2.1.

Centering Step: First apply LSFN as discussed in Section 2.2.1, selecting profitable directions for the line search from the set Γ_2 (instead of Γ_1) until the improvement in the radius of the ball δ per step decreases below some selected tolerance. At that time suppose the point in the sequence being generated is $x^{r,k}$

With $x^{r,k}$, switch and initiate an LSCPD sequence of Section 2.1.2 with $x^{r,k}$ as the initial point of the sequence. We will now follow a different procedure than what is described in Section 2.2.1. Let $x^{r,g}$ denote a general point in the sequence being generated in this sequence.

When $x^{r,g}$ is the current point, we look for a profitable direction to move at $x^{r,g}$ which is a basic solution of the system

$$A_{i.}y = 1 \quad \text{for all } i \in T(x^{r,g}). \tag{13}$$

Notice that the additional constraint "$cy = 0$" as discussed in Section 2.2.1 is not included in (13). Suppose we obtain a basic solution y^0 for this system, with respect to the basic vector y_{B^0} and basis B^0. There are two cases to consider.

Case 1: If $cy^0 \leq 0$, then y^0 is not only profitable at $x^{r,g}$ (i.e., $\delta(x^{r,g} + \alpha y^0)$ increases as α increases from 0), but it is also a descent direction for cx. So, we carry out a line search step at $x^{r,g}$ in the direction y^0 exactly as described in Section 2.1.1 and continue. This moves increases $\delta(x)$ and also may decrease cx.

Case 2: If $cy^0 > 0$, there are two subcases to consider here. Let $(B^0 \vdots D^0)$ be the partition of the coefficient matrix of (13) into basic, nonbasic parts with respect to the basic vector y_{B^0} for it. Let $(c_{B^0} \vdots c_{D^0})$ be the corresponding partition of the row vector c.

Subcase 2.1: $c_{D^0} - c_{B^0}(B^0)^{-1}D^0 = 0$. In this subcase cy = a constant = cy^0 in every solution of (13). So, in this subcase using any solution of (13) as the direction for the move helps increase $\delta(x)$, but also increases cx. So we terminate the sequence at this stage, and take the current point $x^{r,g}$ as the approximation to the ball center on the objective plane through $x^{r,g}$, and go to the descent step discussed next.

Subcase 2.2: There is a nonzero entry in $c_{D^0} - c_{B^0}(B^0)^{-1}D^0$, suppose it occurs in the column of the nonbasic variable y_j and that nonzero entry is denoted by $\bar{c}_j$. Let the column vector of this nonbasic variable y_j in D^0 be denoted by $(D^0)_{.j}$. Then the solution y^1 of (13) given by: all the variables except y_j in the nonbasic vector y_{D^0} are $= 0$, and

$$\begin{aligned} y_{B^0} &= y^0_{B^0} - \theta(B^0)^{-1}(D^0)_{.j} \\ \text{Nonbasic } y_j &= \theta. \end{aligned}$$

where $\theta = (-1 - cy^0)/\bar{c}_j$, satisfies $cy^1 = -1$. Now carry out a line search step at the current point $x^{r,g}$ in the direction y^1 exactly as in Section 2.1.1 and continue. As under Case 1, this move not only increases $\delta(x)$ but also decreases cx.

This seems to output a good approximation to the ball center on the current objective plane through the final point obtained. With that go to the Descent steps discussed next.

Descent Steps: Let $\bar{x}^r$ denote the ball center obtained. We list the various descent steps carried out in this iteration, as described in Section 3, giving each a number for ease in referring to it. In each the step length is determined by a minimum ratio computation as illustrated at the end of Section 3. After each descent step, include the point obtained at the end of it, along with its objective value, in a **list**.

D1, Descent Step 1: From the ball center $\bar{x}^r$ take a descent step in the direction $d^1 = -c^T$.

D2, Descent Step 2: From the ball center $\bar{x}^r$ take a descent step in the direction $d^2 = \bar{x}^r - \bar{x}^{r-1}$.

D3, Descent Steps 3: From the ball center $\bar{x}^r$ take descent steps in the GPTC directions $-c^i$ for each $i \in T(\bar{x}^r)$. Define the direction d^3 as in Section 3.

D4, Descent Step 4: From the ball center $\bar{x}^r$ take a descent step in the direction d^4 defined in Section 3.

D5.1, Descent Steps 5.1: For each $i \in T(\bar{x}^r)$, take a descent step from the NTP $\hat{x}^{ir}$ in the GPTC direction $-c^i$. At the end let $\tilde{x}^{r1}$ denote the best point (i.e., the one corresponding to the least objective value) among the $|T(\bar{x}^r)|$ points obtained at the end of these descent steps; and let d^5 denote the direction used in the step that yielded this point.

Find the best point in the List, let it be x^{r+1}. With x^{r+1} as the initial interior feasible solution, go to the next iteration.

4.2 Implementation Based on Approximating the Ball Center on Current Objective Plane Using the Strategy Discussed in Section 2.2.2

If the strategy discussed in Section 2.2.2 is to be used for centering, then find the approximate ball center on the current objective plane as discussed in Section 2.2.2. First apply LSFN of Section 2.1.1 on the transformed problem until the improvement in $\delta(x)$ per step decreases below some selected tolerance. At that time switch to LSCPD until it terminates. With the final point obtained go to the Descent step, which is the same as in Section 4.1, and continue.

Usually the objective coefficient vector c can be expected to be a dense vector in practice. Remember that in the strategy for centering discussed in Section 2.2.2, the constraint "$cx = t$" is used to eliminate the variable x_n from consideration in this centering step. When c is a dense vector, the resulting system (11) after this elimination step may get several dense rows making it hard to deal with in this implementation. The strategy discussed in Section 4.1 avoids this difficulty, and hence may give better performance, but it is better to compare the two strategies computationally to check out.

Termination Condition: Termination conditions used in other interior point algorithm implementations can be used here also.

The case when K is unbounded: Suppose K, the set of feasible solutions of (1), is an unbounded polyhedron. As discussed earlier, in this case the ball center of K is not defined. But the sphere method can still be applied with the approximate ball centers computed as in Sections 2, 4. As shown in [9] this method will work even when K is unbounded. If cx is unbounded below on K, it will terminate in some iteration with the step length in one of the descent steps as ∞, this is the indication that cx is unbounded below on K.

5 Summary of Computational Results

In this section we present some computational results of implementing our method. We use MATLAB 7.0 to implement the algorithm and test it on some randomly generated problems. Our goal in this section is to computationally test different strategies discussed in the paper for centering and descent steps. At this point we are interested in the performance of the algorithm with each one of these directions and choose the most efficient ones for future development of a software.

While we believe that our algorithm has the potential to eventually outperform the current algorithms for linear programming, but the results presented here are not intended to test this hypothesis. We use a preliminary code that is written by the built-in MATLAB functions. For example in LSCPD, as the algorithm moves from one step to the next, we do not use the updating formulae given in Section 2.1.2 to update the inverse; instead in these preliminary experiments, in each step we solve the system of equations (8) to be solved in that step using the MATLAB linear equation solver from scratch. Therefore the numerical results illustrated here are only preliminary.

To explore the full potential and practical power of the algorithm proposed in this paper, and compare its convergence speed with that of simplex and IPMs, a computer code in a low-level programming language should be developed that uses the advanced techniques of numerical linear algebra, in particular appropriate methods to update the inverse of a matrix as it grows gradually as described in Section 2.1.2. We have not done that in these initial experiments.

From these preliminary experiments, we are able to get some measures of efficiency of our technique, and compare its performance with the simplex algorithm in terms of the average percent move toward the optimal solution per iteration. Also we run our code on problems where $m >> n$, which includes highly redundant problems and show that in terms of cpu time even our preliminary code outperforms simplex method on these problems.

Problem Generation: To generate entries of the coefficient matrix, and vector c, we utilize the MATLAB function "*randn*" that generates random numbers from the Standard Normal distribution. To ensure feasibility of the LP generated, we use the function "$-rand$" that generates random numbers from the Uniform distribution in $(-1, 0)$ for the RHS vector. Then, we include box constraints $l \leq x \leq u$, where l and u are n-vectors with negative and positive random entries respectively. Therefore the initial point, $x^0 = 0$, is strictly feasible for the system of constraints generated. The data is normalized by dividing each nonzero row of A (and the corresponding RHS element of b) by the norm of that row. The vector c also is normalized. We run our test problems on a laptop computer with Intel(R) Pentium(R) M processor 2.00GHz and 2.00 GB of RAM.

Computational Issues: In the centering step, although a direction y that satisfies $A_{i.}y \geq 0$ for all $i \in T(x^{r,k})$ is profitable at $x^{r,k}$, the improvement in δ using that direction might be very small if $A_{i.}y$ is small. Therefore it is necessary to use a tolerance for this condition and check $A_{i.}y \geq \varepsilon > 0$. On the other hand, this tolerance ϵ should depend on the proximity of $x^{r,k}$ to the optimal solution of the LP. That is as we get closer to the optimal solution of the LP, the tolerance ε should be reduced. Therefore we start the algorithm with a relatively large tolerance, say $\varepsilon = 0.01$ and at the end of each iteration we update the tolerance by $\frac{\varepsilon}{r+1}$.

Another issue with LSFN is that although it is less expensive than LSCPD, we found that as the approximate center gets closer to the actual center, the improvement in δ in it becomes only marginal even with the above correction to the tolerance. Therefore we need another tolerance to stop LSFN when the improvement in δ becomes small. Through some trial and error we found that the reduction in tolerance should depend not only on iteration r but also on m and n and at the same time it should not be too large. Thus in each iteration if the improvement in δ in LSFN falls below $\frac{\varepsilon}{(r+1)\sqrt{\min(m,n)}}$, we stop LSFN and start LSCPD.

Again in LSCPD, as the approximate center gets close to the true center the step lengths α in steps of the sequence become marginal. Therefore we need another strategy to terminate the sequence LSCPD as step lengths α in the steps of the sequence become small. For this the reduction in tolerance should be larger and faster. So, we start the procedure with $\varepsilon = 0.0001$ and update it at each iteration by $\frac{\varepsilon}{(r+1)\sqrt{\max(m,n)}}$, and terminate LSCPD when the step length α in a step falls below this tolerance. The resulting center from this step is $\bar{x}^r$.

The descent steps start at $\bar{x}^r$ resulting from LSCPD. We compute the four sets of descent directions discussed in Sections 3, 4 and compare their performance. We need another tolerance factor to determine how close to the boundary to stop each descent step. For this we use the same tolerance as under LSCPD. In summary, here is the computational version of the algorithm.

Computational Version of the Algorithm

Let $x^0 = 0$ be the initial point, $\varepsilon = 1.0\times10^{-6}$, $\varepsilon_1 = 1.0\times10^{-2}$, $\varepsilon_2 = 1.0\times10^{-4}$ $k_1 = \sqrt{\min(m, n)}$, $k_2 = \sqrt{\max(m, n)}$, and $r = 0$.

Step 1. Let $x^{r,0} = x^r$. Perform the centering step using LSFN described in Section 8.1 to obtain $x^{r,k}$ for $k = 1, 2, \dots$, until no normal direction y in Γ_2 exists such that $A_{i.}y \geq \frac{\varepsilon_1}{r+1}$, for all $i \in T(x^{r,k})$, or the improvement in δ falls below $\frac{\varepsilon_1}{k_1(r+1)}$.

Step 2. Starting from the final $x^{r,k}$ obtained in Step 1, perform the centering steps of LSCPD discussed in Section 4.2 to obtain direction y that satisfies (14). Update $x^{r,k+1} = x^{r,k} + \alpha y$. Stop this step when $\alpha < \frac{\varepsilon_1}{k_2(r+1)}$.

Step 3. Compute directions in the four classes

- Class 1: $d^1 = -c^T$,
- Class 2: $d^2 = \bar{x}^r - \bar{x}^{r-1}$,
- Class 3: $-c^i$ defined in (13) for each $i \in T(\bar{x}^r)$,
- Class 4: $d^4 = (\sum(-c^i : \text{for } i \in T(\bar{x}^r))/|T(\bar{x}^r)|$.

Carry out descent steps from $\bar{x}^r$ in each direction in the four classes, to within $\frac{\varepsilon_2}{k_2(r+1)}$ of the boundary of the feasible region. Let d^3 be the direction from Class 3, that gives the best result when the descent step is taken from $\bar{x}^r$.

In the same way, for each $i \in T(\bar{x}^r)$ compute the NTP $\hat{x}^{ir}$, and carry out a descent step from it in the direction c^i. Let d^5 denote the direction in Class 3 that gives the best result when the descent step is taken from its corresponding NTP.

Let d be the direction corresponding to the best point, x^{r+1}, (having the largest reduction in objective value) in all these descent steps.

Step 4. If $\frac{\|x^{r+1}-x^r\|}{\|x^{r+1}\|} < \varepsilon$, stop. Otherwise set $r = r + 1$ and go back to Step 1.

Numerical Results: Let us first show the typical behavior of the algorithm when implemented on a randomly generated problem with a moderate size of

Table 1: sphere radius and objective values before and after LSFN and LSCPD for a random problem with $m = 150$ and $n = 50$

Itr.	cx^{r-1}	c_{fn}	t_{fn}	δ_{fn}	c_{pd}	t_{pd}	δ_{pd}	cx^r
1	0	15	6	0.5737	34	22	2.0528	-3.3157
2	-12.5277	18	5	0.4685	36	30	2.0534	-14.7995
3	-21.5544	6	6	0.0786	43	50	1.4048	-22.8807
4	-27.4741	3	4	0.0291	44	50	0.5963	-28.0413
5	-29.9104	6	5	0.0562	41	49	0.2705	-30.1247
6	-30.9835	3	6	0.0293	34	47	0.1060	-31.0603
7	-31.3951	1	6	0.0040	30	46	0.0393	-31.4304
8	-31.6208	0	38	0.0000	8	49	0.0034	-31.6242
9	-31.6417	0	45	0.0000	2	48	0.0004	-31.6421
10	-31.6447	0	48	0.0000	0	48	0.0000	-31.6447

$m = 150$ and $n = 50$. Table 1 illustrates the performance of our two centering steps. The first and second columns of this table show the iteration, and the value of the objective function of the original LP in the beginning of the centering step at that iteration, respectively. The column under c_{fn} shows the number of calls to LSFN to update the sphere radius, the column indicated by t_{fn} shows the number of touching constraints, and δ_{fn} is the radius of the sphere at the end of LSFN. c_{pd}, t_{pd} and δ_{pd} have similar meanings for one sequence of LSCPD. The last column shows the objective value of the original LP at the end of the centering steps (not descent step). For example in the first iteration, the objective value is initially zero, there are 15 calls to LSFN and the resulting radius is 0.5737 where 6 constraints are touching. Then LSCPD sequence had 34 steps that increases the radius to 2.0528 with 22 touching constraints at which point the objective value of the original LP is also improved to -3.3157.

As Table 1 shows, in earlier iterations LSFN is used to obtain an approximate center, and LSCPD is utilized as a supplement to improve the sphere radius. As we get closer to the optimal objective value, the use of LSFN is significantly reduced but LSCPD is used instead to obtain the approximate center. This table also shows that the sequence LSCPD not only improves δ but also reduces the objective value of the original LP, because some directions y used in steps of LSCPD satisfy $cy < 0$ as described in Section 4.1.

Table 2 illustrates the performance of the algorithm in descent steps. As mentioned earlier we consider five classes of descent directions. To study these directions from computational viewpoint for future references, we compare their performances.

The first two columns of this table show the iteration and the objective value at the current center. The next five columns of the table indicated by $cx^{r+1}_{d^i}$, for $i = 1, \dots, 5$, illustrate the value of the objective function after the descent step along direction d^i, and the column under d shows the direction that gave the best result in the problem.

Table 2: Descent steps based on directions $d^1, \dots, d^5$ on a random problem with $m = 150$ and $n = 50$

Itr.	cx^r	$cx_{d^1}^{r+1}$	$cx_{d^2}^{r+1}$	$cx_{d^3}^{r+1}$	$cx_{d^4}^{r+1}$	$cx_{d^5}^{r+1}$	d
1	-3.3157	-10.3051	-3.3157	-11.9094	-10.5648	-12.5277	d^5
2	-14.7995	-21.0559	-17.7847	-21.1657	-21.0721	-21.5544	d^5
3	-22.8807	-27.1514	-24.6089	-27.2263	-27.1355	-27.4741	d^5
4	-28.0413	-29.8521	-28.9576	-29.8712	-29.8445	-29.9104	d^5
5	-30.1247	-30.9531	-30.2478	-30.9535	-30.9497	-30.9835	d^5
6	-31.0603	-31.3802	-31.2448	-31.3804	-31.3788	-31.3951	d^5
7	-31.4304	-31.5522	-31.6208	-31.5544	-31.5521	-31.5615	d^2
8	-31.6242	-31.6346	-31.6288	-31.6406	-31.6346	-31.6417	d^5
9	-31.6421	-31.6432	-31.6426	-31.6438	-31.6432	-31.6447	d^5
10	-31.6447	-31.6447	-31.6447	-31.6447	-31.6447	-31.6461	d^5

Table 2 gives two useful information items. First, in 9 out of 10 iterations d^5 has been selected as the best direction, i.e., Descent Step 5.1 gave the best results among all the descent steps used. We had almost the same experience with all of our test problems. So it seems that the direction d^5 obtained in Descent Step 5.1, is the direction to be selected in our descent step. Secondly, it shows that our descent direction significantly reduces the objective value at each iteration. In this example the average reduction in objective value has been about 13.2% per iteration with the higher percentage in earlier iterations.

We now compare our algorithm with the Primal Simplex method on some randomly generated problems of small to moderate size. Our measure is based on the average percent move toward the optimality per iteration. That is at each iteration, for both our algorithm and the Primal Simplex algorithm, we measure

$$\frac{cx^r - cx^{r+1}}{cx^r - cx^*},$$

where x^* is the true optimal solution of LP.

The reason that we use only small to moderate size problems is both because our code is still under development and is not finalized yet, and also because the simplex method takes a very long time to solve problems of large size. Moreover this comparison between the average percent move toward the optimality can be done with problems of any size.

Table 3, summarizes the results of the comparison between our algorithm and the simplex method. We use the MATLAB function "linprog" when its option "LargeScale" is turned off to use the simplex method. The first column of this table gives the problem size (m, n). The second column shows the density of the coefficient matrix excluding the bound constraints on individual variables. The density is determined when the random data is generated by the MATLAB functions "*sprandn*" and "*sprand*". "*sprandn*" and "*sprand*" generate random numbers from a standard normal and uniform distribution in $(0, 1)$,

Table 3: Comparison of the average percent move toward the optimality per iteration of the Sphere Method and the Simplex Method

(m, n)	density	k_1	LSFN	LSCPD	e_1	k_2	e_2
(30, 10)	100	5	14	29	73.55	14	20.38
	50	4	4	24	68.64	15	23.01
	10	3	1	1	99.82	4	58.15
(60,20)	100	9	25	86	46.27	69	7.93
	50	9	31	80	57.43	72	6.88
	10	6	2	54	27.97	28	12.93
(90, 30)	100	15	35	200	30.33	158	4.35
	50	16	14	217	15.60	181	3.80
	10	14	9	173	12.28	78	8.83
(150, 50)	100	22	48	269	17.31	453	2.07
	50	19	25	302	8.02	535	1.64
	10	17	17	123	4.70	488	1.27
(300, 100)	100	19	58	644	19.36	2085	0.49
	50	16	42	572	15.02	2421	0.38
	10	15	30	691	9.38	2598	0.22

respectively with a given density. For instance, an $m \times n$ matrix with density of 10% means that the number of nonzero entries of the matrix is approximately $0.10 \times m \times n$. We use problems with 10%, 50% and 100% densities to illustrate our computational results on sparse and dense problems.

The third to sixth columns of the table represent results from the Sphere method: its number of iterations, and the total calls to LSFN, LSCPD centering procedures respectively, and e_1 represents the average percent move toward optimality per iteration. The last two columns of the table give the results from the Simplex method: the number of iterations and the average percent move toward the optimality per iteration.

Average cpu times per iteration are not given in Table 3 because for the simplex method we used the finished MATLAB code "linprog", whereas for the sphere method we used various individual MATLAB functions for each step separately.

The results illustrated in Table 3 for these randomly generated problems show that our algorithm has clear advantage over the simplex method. Comparing these results for both methods shows that each iteration moves faster toward the optimal solution which results in fewer iterations.

We now illustrate the performance of our algorithm on problems where $m >> n$, and compare our cpu time with that of simplex method. We tested our code with four random problems generated in three steps. At each step we added additional redundant constraints. The results are presented in Table 4. For these problems we limited the use of LSFN to take advantage of LSCPD

Table 4: Comparison of CPU time for highly redundant problems

n	m	LSFN	LSCPD	sphere method	simplex
50	500	8	31	45	69
	1000	11	30	42	132
	1500	10	31	44	375
100	700	29	102	145	248
	1200	34	126	180	584
	1700	65	255	350	929
200	900	269	974	1356	1422
	1200	157	498	702	3863
	2000	230	688	1012	5100
300	900	344	1274	1748	4758
	1800	206	733	1168	7488
	3000	244	855	1264	9999

property that redundant constraints never enter into the touching constraints set.

The columns of this table show the amount of time (in seconds) for the major subroutines (LSFN and LSCPD), as well as the total time of our algorithm and simplex method.

The results of Table 4 clearly shows the advantage of our algorithm when applied to highly redundant problems. Even our preliminary code outperforms simplex algorithm on these problems. The same advantage of the sphere method over simplex method was observed when we generated problems with large number of constraints (not necessarily redundant), which further confirms our intuition that dense problems with large number of constraints can be handled efficiently with sphere method.

Table 4 also shows that a large proportion of the cpu time for sphere method is taken by the centering procedures LSFN (20%) and LSCPD (70%). Since these two subroutines contain many loops, the use of individual MATLAB functions for each step separately slows the whole process. This confirms our earlier statement that implementing the procedure described in Section 2.1.2. by a low-level programming language would substantially improve the performance of the algorithm.

6 Future Research Directions

We are investigating additional directions for line search to include in the the procedures used for computing approximate ball centers, to accelerate their convergence rate, and to improve the quality of the approximation to the optimum centre. We are also investigating additional descent steps that may speed up the method.

Also, we have so far been able to test the numerical performance of some alternatives in the algorithm using a preliminary MATLAB code. More extensive testing is needed to determine the best alternatives, and then prepare a good code with these alternatives for tests using available large scale test problems.

The sphere method also opens up important new research topics in computational linear algebra. To get the best results from an implementation of the simplex method, reserachers developed efficient factorization techniques to update the inverse of a matrix as one of its column vectors changes from one step to the next. An important component of the sphere method is the LSCPD sequence of steps in which a matrix grows by a row and a column from one step to the next. To get the best results from the sphere method, we need to develop appropriate techniques to update the inverse as the matrix grows in this way.

Acknowledgements: We thank Huang Chia-Hui, and Katta Sreeramamurthy for the nice figures. Also, we are grateful to the referee for suggesting the name *ball center*, and for making many suggestions to improve the paper.

7 References

[1] A. I. Ali, C. S. Lerme, and L. M. Seiford, “Components of Efficiency Evaluation in Data Envelopment Analysis”, EJOR, 80(1995)462-473.

[2] C. Cartis and N.I.M. Gould, “Finding a Point in the Relastive Interior of a Polyhedron”, RAL Technical Report 2006-016, available from Optimization Online, December 2006,

[3] W. W. Cooper, L. M. Seiford, and K. Tone; *Data Envelopment Analysis: A Comprehensive Text with Models, Applications, References and DEA-Solver Software*, 2nd Edition, Springer, NY, 2006.

[4] G. B. Dantzig and M. N. Thapa, *Linear Programming, 1. Introduction, 2. Theory and Extensions.* Springer-Verlag New York, 1997.

[5] M. Kojima, S. Mizuno, and A. Yoshise, “A primal-dual interior point algorithm for linear programming”, *Progress in Mathematical Programming: Interior Point and Related Methods*, N. Megiddo, ed., Springer-Verlag, New York, ch. 2 (29-47) 1989.

[6] N. Megiddo, “Pathways to the optimal set in linear programming”, *Progress in Mathematical Programming: Interior Point and Related Methods*, N. Meggiddo, ed., Springer-Verlag, New York, ch. 8 (131-158) 1989.

[7] S. Mehrotra, “On the implementation of a primal-dual interior point method”, *SIAM Journal on Optimization*, 2 (575-601) 1992.

[8] R. D. C. Monteiro and I. Adler, “Interior path-following primal-dual algorithms, Part I: Linear programming”, *Mathematical Programming* 44 (27-41) 1989.

[9] K. G. Murty, “A new practically efficient interior point method for LP”, *Algorithmic Operations Research*, 1 (3-19) 2006; paper can be seen at the website: http://journals.hil.unb.ca/index.php/AOR/index.

[10] K. G. Murty, “Linear equations, Inequalities, Linear Programs (LP), and a New Efficient Algorithm” Pages 1-36 in *Tutorials in OR*, INFORMS, 2006.

[11] R. Saigal. *Linear Programming A Modern Integrated Analysis.* Kluwer Academic Publishers, Boston, MA, 1995.

[12] G. Sonnevend, J. Stoer, and G. Zhao, “On the complexity of following the central path of linear programming by linear extrapolation”, *Mathematics of Operations Research* 62 (19-31) 1989.

[13] S. J. Wright. *Primal-Dual Interior-Point Methods.* SIAM, Philadelphia, PA, 1997.

[14] Y. Ye. *Interior Point Algorithms, Theory and Analysis*, Wiley-Interscience, New York, 1997.