Karmarkar's Linear Programming Algorithm

J. N. HOOKER

Graduate School of Industrial Administration
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Editor's Note: Occasionally an event occurs in our field that captures the attention of all — academics and practitioners, public sector and private sector specialists, methodologists and modelers. The report of Karmarkar's algorithm is such an event. The following article describes the importance of the advance and provides both an intuitive explanation of its strengths and weakness as well as enough technical detail for readers to implement the method. The article is technical by *Interfaces'* standards. But the subject is central to MS/OR and is addressed clearly in this article. Dr. Karmarkar was invited to comment and has yet to respond as we go to press.

N. Karmarkar's new projective scaling algorithm for linear programming has caused quite a stir in the press, mainly because of reports that it is 50 times faster than the simplex method on large problems. It also has a polynomial bound on worst-case running time that is better than the ellipsoid algorithm's. Radically different from the simplex method, it moves through the interior of the polytope, transforming the space at each step to place the current point at the polytope's center. The algorithm is described in enough detail to enable one to write one's own computer code and to understand why it has polynomial running time. Some recent attempts to make the algorithm live up to its promise are also reviewed.

Narendra Karmarkar's new projective scaling algorithm for linear programming has received publicity that is rare for mathematical advances. It attracts attention partly because it rests on a striking theoretical result. But more important are reports that it can solve large linear programming problems much more

Copyright © 1986, The Institute of Management Sciences
0091-2102/86/1604/0075$01.25

PROGRAMMING, LINEAR — ALGORITHMS

This paper was refereed.

rapidly than the simplex method, the method of choice for over 30 years.

(Although the name "projective algorithm" is perhaps the most popular to date, J. K. Lagarias has suggested that the algorithm be called the "projective scaling algorithm" to distinguish it from a variant that is naturally called the "affine scaling algorithm." When I asked Dr. Karmarkar about this suggestion, he endorsed it.)

The simplex and projective scaling methods differ radically. George Dantzig's simplex method [1963] solves a linear programming problem by examining extreme points on the boundary of the feasible region. The projective scaling method is an interior method; it moves through the interior of the feasible polytope until it reaches an optimal point on the boundary.

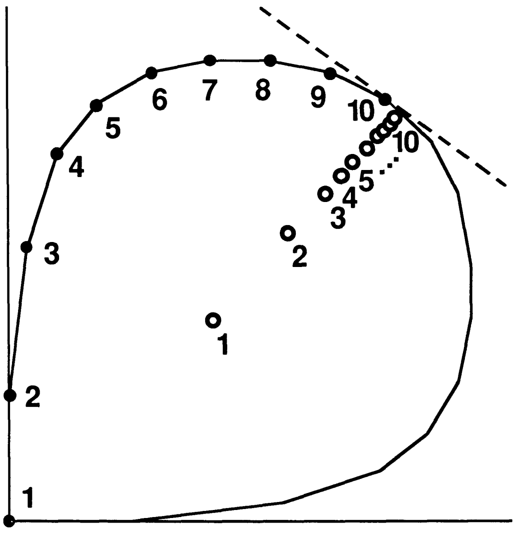

Figure 1: Illustration of how the simplex method (dots) and projective scaling method (circles) approach the optimal solution of a small problem. Here the projective scaling method requires at least as many iterations as the simplex method, but in large problems it requires only a fraction as many.

Figure 1 illustrates how the two methods approach an optimal solution. In this small problem the projective scaling method requires no fewer iterations (circles) than the simplex method (dots). But a large problem would require only a fraction as many projective as simplex iterations.

The main theoretical attraction of the projective scaling method is its vastly superior worst-case running time (or worst-case "complexity"). Suppose we define the size of a problem to be the number N of bits required to represent the problem data in a computer. If an algorithm's running time on a computer is never greater than some fixed power of N, no matter what problem is solved, we say that the algorithm has polynomial worst-case running time. The projective scaling method is such an algorithm [Karmarkar 1984a; 1984b], and the simplex method is not.

Why does this matter? It matters because the running time of a nonpolynomial algorithm like the simplex method can grow exponentially with the problem size in the worst case; that is, there are classes of problems in which the running time is proportional to 2^N and thus is bounded by no fixed power of N. On these problems the simplex method is a miserable failure, because running time explodes very rapidly as the problem size increases. Such a problem having, say, 100 variables would already run trillions of years on a Cray X-MP supercomputer (among the world's fastest). The projective scaling method, on the other hand, never exhibits this sort of exponential explosion.

Curiously, the simplex method

performs quite well in practice, despite its dreadful worst-case behavior. It appears that nothing remotely resembling a worst-case problem ever arises in the real world. Still, one might expect a method that performs much better in the worst case, such as the projective scaling method, to excel on real-world problems as well. Such was the hope for the first and only other LP algorithm known to run in polynomial time, L. G. Hačijan's ellipsoid method [1979]. Hačijan's achievement was a theoretical breakthrough and likewise made the front page. But it requires that calculations be done with such high precision that its performance on typical problems is much worse than the simplex method's.

The projective scaling method, however, is more promising. It is free of the ellipsoid method's precision problem, and its worst-case behavior is substantially better. More important, one hears claims that when properly implemented it is much faster than the simplex method, even 50 times faster, on large real-world problems (for example, Kolata [1984]). I have yet to see these claims substantiated, but it is much too early to close the case.

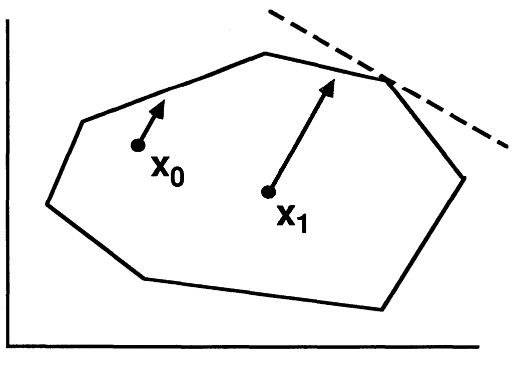

Figure 2: One can improve the current solution substantially by moving in the direction of steepest descent (arrows) if it is near the center of the feasible region, as is x_0, but generally not if it is near the boundary, as is x_1.

My aim here is to present the projective scaling method in enough detail to allow one to write one's own computer implementation and to understand why its running time is polynomially bounded. I also mention some recent developments and computational experience and draw tentative conclusions.

The Basic Idea

Karmarkar [1984c] has said that his approach is based on two fundamental insights.

(1) If the current solution is near the center of the polytope, it makes sense to move in a direction of steepest descent (when the objective is to minimize).

(2) The solution space can be transformed so as to place the current solution near the center of the polytope, without changing the problem in any essential way.

The first insight is evident in Figure 2. Since $\mathbf{x}_0$ is near the center of the polytope, one can improve the solution substantially by moving it in a direction of steepest descent. But if $\mathbf{x}_1$ is so moved, it will hit the boundary of the feasible region before much improvement occurs.

The second insight springs from the observation that when one writes down the data defining a linear program, one, in a sense, overspecifies the problem. The scaling of the data, for instance, is quite arbitrary. One can switch from feet to inches without changing the problem in any important sense. Yet a transformation of the data that leaves the problem essentially unchanged may nonetheless ease its solution. Rescaling, for example, may reduce numerical instability.

Karmarkar has observed that there is a transformation of the data more general than ordinary rescaling but equally natural. It occurs every time one views the graph of an LP problem at an oblique angle. The projection of the graph on one's retina is a distortion of the original problem; it is a special case of a projective transformation. Straight lines remain straight lines, while angles and distances change. Yet it seems clear that the distortion scarcely alters anything essential about the problem, since we readily solve such problems visually.

Karmarkar's new projective scaling algorithm for linear programming has received publicity that is rare for mathematical advances.

A key property of projective transformations is that a suitable one will move a point strictly inside a polytope to a place near the center of the polytope. One can verify this with Figure 2 by viewing it at an angle and distance that makes $\mathbf{x}_1$ appear to be at the center of the polytope.

The basic strategy of the projective scaling algorithm is straightforward. Take an interior point, transform the space so as to place the point near the center of the polytope and then move it in the direction of steepest descent, but not all the way to the boundary of the feasible region (so that the point remains interior). Transform the space again to move this new point to a place near the center of the polytope and keep repeating the process until an optimum is obtained with the desired accuracy.

A problem must satisfy two conditions before Karmarkar's original algorithm will solve it: it must be in (almost) "homogeneous" form, and its objective function must have a minimum value of zero. It is not hard to put any given problem in the required form, but it is more difficult to deal with the second requirement.

The Required Form of the Problem

The Karmarkar algorithm requires that the problem be transformed to one that has a very special form, namely

$$\begin{aligned}&\text{minimize } \mathbf{c}^T\mathbf{y}\\ &\text{subject to } A\mathbf{y} = \mathbf{0} \qquad (1)\\ &\qquad \mathbf{e}^T\mathbf{y} = 1,\ \mathbf{y} \geq \mathbf{0},\end{aligned}$$

where A is an $m \times n$ matrix, and $\mathbf{e}$ is a vector of n ones. Also, an interior feasible starting solution for (1) must be known. (An *interior* solution is one in which every variable is strictly positive.)

An example problem that is already in the right form is,

$$\begin{aligned}&\text{minimize } 2y_1 + y_2 + y_3\\ &\text{subject to } 2y_1 + y_2 - 3y_3 = 0 \qquad (2)\\ &\qquad y_1 + y_2 + y_3 = 1,\ y_1, y_2, y_3 \geq 0.\end{aligned}$$

Figure 3 is a plot of the problem. The triangular area, which we will call the "simplex," is the set of points satisfying the normalization constraint $y_1 + y_2 + y_3 = 1$ and the nonnegativity constraints y_1, y_2, $y_3 \geq 0$. The line segment stretching across the triangle is the feasible polytope for the problem. Some contours of the objective function appear, and the optimal point $\mathbf{y}^* = (0,3/4,1/4)$ is indicated. Note that the objective function value at this

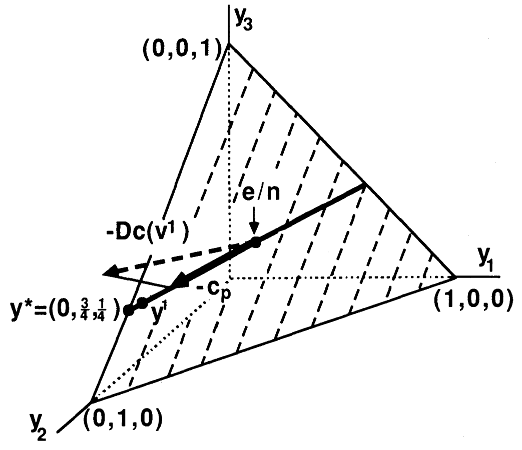

Figure 3: Illustration of a three-variable problem in the required form. The "simplex," or triangular area, is the set of points satisfying the normalization and nonnegativity constraints. The line segment stretching across the triangle is the feasible polytope. The dashed lines are objective function contours, y^1 is the solution obtained at the end of the first iteration, and y^* is the optimal solution.

point is one, not zero as required. We can spot an interior starting feasible solution: $\mathbf{y} = (1/3,1/3,1/3) = \mathbf{e}/n$, which is marked in the figure.

Offsetting the Objective Function

There is no generally accepted way to deal with the requirement that the objective function $\mathbf{c}^T\mathbf{y}$ have a minimum value of zero. Karmarkar originally proposed a "sliding objective function" approach [1984a], but it was intended for the theoretical purpose of proving polynomial complexity and is totally unsuited for practical use. He later proposed that a linear program be solved by solving the primal/dual feasibility problem [1984b], but this approach roughly quadruples the size of the problem. He has also alluded to a "two dimensional search" method [1985].

Here we adopt a method devised by Todd and Burrell [1985], which does not enlarge the problem, is fairly easy to implement, and delivers an optimal solution of the dual problem as a byproduct. Anstreicher [1986] has described a similar method that has an entirely different, geometrical motivation.

Todd and Burrell's method works basically like this. Let us say we are given a problem, such as (2), that is in the form (1) but whose objective function has some unknown minimum value v^*. If we only knew v^*, we could offset $\mathbf{c}^T\mathbf{y}$ by v^* to get an objective function $\mathbf{c}^T\mathbf{y} - v^*$ with a minimum value of zero. In fact, since $\mathbf{e}^T\mathbf{y} = 1$, we could observe that $\mathbf{c}^T\mathbf{y} - v^* = \mathbf{c}^T\mathbf{y} - v^*\mathbf{e}^T\mathbf{y} = (\mathbf{c} - v^*\mathbf{e})^T\mathbf{y}$. Then we could replace $\mathbf{c}$ in (1) with $\mathbf{c} - v^*\mathbf{e}$ and solve the problem Karmarkar's way.

Since we don't know v^*, Todd and Burrell propose that we use an estimate v of v^*, updated as we go along. At each iteration we replace $\mathbf{c}$ in (1) with $\mathbf{c}(v) = \mathbf{c} - v\mathbf{e}$. This method should work if the estimate v becomes arbitrarily close to the true minimum v^* in the course of the algorithm. To make sure that it does, Todd and Burrell identify a dual feasible solution at each iteration. They let v be the corresponding value of the dual objective function, which by duality theory is a lower bound on v^*. They show that these dual feasible solutions converge to the dual optimal solution, so that v converges to v^*, as desired.

To see how to find a dual feasible solution, just write down the dual of (1). It is,

$$\begin{aligned}&\text{maximize } v\\&\text{subject to } A^T\mathbf{u} + \mathbf{e}v \le \mathbf{c}\,.\end{aligned}\tag{3}$$

Note that for any m-vector $\mathbf{u}$ whatever,

$(\mathbf{u},v)$ is a dual feasible solution if $v = \min_j \{ (\mathbf{c} - A^T\mathbf{u})_j\}$. It remains only to choose a $\mathbf{u}$ at each iteration so that the $(\mathbf{u},v)$'s converge to an optimal dual solution as the algorithm progresses.

The Main Algorithm

The main algorithm begins with a problem in the form (1) and with a starting point $\mathbf{y}^0$ that lies in the interior of the polytope. The statement of the algorithm may be easier to follow if, on first reading, the material dealing with dual variables is ignored. This material is enclosed in brackets.

Step 0. Set the iteration counter k to zero, and let $\mathbf{y}^0$ be a point interior to the polytope. [Let the first value $\mathbf{u}^0$ of the vector $\mathbf{u}$ of dual variables be the solution of the equation $AA^T\mathbf{u}^0 = A\mathbf{c}$, and let the first estimate of v be $v^0 = \min_j \{ (\mathbf{c} - A^T\mathbf{u}^0)_j\}$.]

In the example, the initial point is $\mathbf{y}^0 = (1/3,1/3,1/3)$. [Since $AA^T = 14$ is a scalar in this problem, and $A\mathbf{c} = 2$, $\mathbf{u}^0$ is easily seen to be 1/7. Thus $v^0 = \min\{12/7,6/7,10/7\} = 6/7$.]

Step 1. Transform the space so as to put the current point $\mathbf{y}^k$ at the center of the simplex. The transformation T and its inverse are given by,

$$\mathbf{z} = T(\mathbf{y}) = \frac{D^{-1}\mathbf{y}}{\mathbf{e}^TD^{-1}\mathbf{y}}, \quad \mathbf{y} = T^{-1}(\mathbf{z}) = \frac{D\mathbf{z}}{\mathbf{e}^TD\mathbf{z}}$$

where $D = \mathrm{diag}(y_1{}^k, \ldots, y_n{}^k)$. Note that $T(\mathbf{y}^k) = \mathbf{e}/n$. The problem (1) becomes,

$$\begin{aligned}&\text{minimize} \quad \frac{\mathbf{c}^TD\mathbf{z}}{\mathbf{e}^TD\mathbf{z}} \\ &\text{subject to} \quad \begin{array}{l} AD\mathbf{z} = \mathbf{0} \\ \mathbf{e}^T\mathbf{z} = 1,\ \mathbf{z} \geq \mathbf{0}\ . \end{array}\end{aligned} \tag{4}$$

In the example, the starting point $\mathbf{y}^0 = (1/3,1/3,1/3)$ is already at the center of the simplex, so that T is just the identity transformation, and $\mathbf{y} = \mathbf{z}$. Also $AD = [2/3\ 1/3\ -1]$.

[Step 2. Update the dual variables. Let $\mathbf{u}^{k+1}$ be the solution of the system of linear equations,

$$AD^2A^T\mathbf{u}^{k+1} = AD^2\mathbf{c}(v^k), \tag{5}$$

and let $\tilde{v} = \min_j \{ (\mathbf{c} - A^T\mathbf{u}^{k+1})_j\}$. If $\tilde{v} \leq v^k$, then we have not improved our previous lower bound v^k on v^*, and we can let $v^{k+1} = v^k$. If $\tilde{v} > v^k$, then we adopt the tighter bound $v^{k+1} = \tilde{v}$, and we revise $\mathbf{u}^{k+1}$ to be the solution of the system of linear equations,

$$AD^2A^T\mathbf{u}^{k+1} = AD^2\mathbf{c}(v^{k+1}). \tag{6}$$

If equation (5) is solved, say, by computing a QR factorization of A^T, then (6) can be solved with relatively little extra work. Todd and Burrell prove that when the $\mathbf{u}^k$'s are chosen in the above way, the $(\mathbf{u}^k,v^k)$'s converge to an optimal dual solution.]

[In the example problem, AD^2A^T and $\mathbf{u}^k$ are scalars. Since $\mathbf{c}(v^0) = (8/7,1/7,1/7)$, equation (5) has the solution $u^1 = 1/7$. Thus $\tilde{v} = \min\{12/7,6/7,10/7\} = 6/7$. Since $\tilde{v} = v^0$, we have not improved our lower bound on v^*, and we set $v^1 = 6/7$.]

Step 3. Find the direction of steepest descent in the transformed polytope. The objective function of (4), with $\mathbf{c}(v^{k+1})$ replacing $\mathbf{c}$, drops most rapidly in the direction of the negative gradient, which at $\mathbf{e}/n$ is parallel to $-D\mathbf{c}(v^{k+1})$ (if we ignore a component normal to the simplex and therefore irrelevant here). In Figure 3 this direction is parallel to the dashed arrow labeled $-D\mathbf{c}(v^1)$. To find the direction $-\mathbf{c}_p$ of steepest descent within the polytope (heavy line segment), we drop a perpendicular from the dashed arrow onto the subspace (here, a line) defined by $AD\mathbf{z} = 0$ and $\mathbf{e}^T\mathbf{z} = 0$. This yields its *orthogonal projection*, the solid arrow labeled $-\mathbf{c}_p$. (Do not confuse this orthogonal projection with the projective transformation T.)

Since $\mathbf{c}_p$ is an orthogonal projection, $\mathbf{z} = \mathbf{c}_p$ solves the least squares problem,

$$\begin{aligned}&\underset{\mathbf{z}}{\text{minimize}}\ \|D\mathbf{c}(v^{k+1}) - \mathbf{z}\|\\ &\text{subject to } AD\mathbf{z} = \mathbf{0}\\ &\qquad\qquad \mathbf{e}^T\mathbf{z} = 0\,.\end{aligned} \tag{7}$$

Remarkably, we have already done most of the work necessary to solve (7), because (6) is precisely the system of normal equations one would solve to solve (7) with the last constraint omitted. We can easily compute $\mathbf{c}_p$ by

$$\mathbf{c}_p = P[D\mathbf{c}(v^{k+1}) - DA^T\mathbf{u}^{k+1}], \tag{8}$$

where P is the matrix for a projection onto $\{\mathbf{z} \mid \mathbf{e}^T\mathbf{z} = 0\}$, given by $P = I - \mathbf{e}\mathbf{e}^T/n$.

In the example problem, we get $\mathbf{c}_p = P[D\mathbf{c}(v^1) - DA^Tu^1] = P(6/21,0,4/21) = (8/63,-10/63,2/63)$. Since the feasible polytope in this example is a line segment, there are only two feasible directions: $\mathbf{c}_p$ and $-\mathbf{c}_p$. A more interesting example would unfortunately require an illustration in four or more dimensions.

Step 4. We now move in the direction $-\mathbf{c}_p$ of steepest descent, but not so far as to leave the feasible set. One easy way to avoid infeasibility is to inscribe a circle of radius $r = [n(n-1)]^{-1/2}$ in the triangle of Figure 3, with its center at $\mathbf{e}/n$. Since the circle contains only feasible points, it is always safe to move across a distance αr where $0 < \alpha < r$.

Another method is simply to move as far as possible in the direction $-\mathbf{c}_p$ without reaching the boundary of the polytope — that is, without letting any z_j go to zero. This can be achieved with a simple ratio test. If the new point is $\mathbf{z}^{k+1} = \mathbf{e}/n - \gamma\mathbf{c}_p$, we want γ to be as large as possible subject to the condition that each component of $\mathbf{z}^{k+1}$ be no less than some small positive number ϵ. That is, we want to

$$\begin{aligned}&\text{maximize } \gamma\\ &\text{subject to } 1/n - \gamma c_{pj} \geq \epsilon,\ j = 1, \ldots, n.\end{aligned} \tag{9}$$

Clearly the maximum permissible value of γ is the minimum of the ratios $(1/n - \epsilon)/c_{pj}$ over all j for which $c_{pj} > 0$. (Since an overly large γ does more harm than good, it is better in practice to minimize $g(\mathbf{e}/n - \gamma\mathbf{c}_p)$ subject to the constraints in (9) rather than to maximize γ, where g is the "potential function" defined in Appendix 1.)

In the example, if we set $\epsilon = 1/30$, then (9) becomes

$$\begin{aligned}&\text{maximize } \gamma\\ &\text{subject to } 1/3 - (8/63)\gamma \geq 1/30\\ &\qquad\qquad 1/3 + (10/63)\gamma \geq 1/30\\ &\qquad\qquad 1/3 - (2/63)\gamma \geq 1/30\,.\end{aligned}$$

Clearly we want $\gamma = \min\{0.3/(8/63), 0.3/(2/63)\} = 2.3625$. The new point is $\mathbf{z}^1 = \mathbf{e}/3 - \gamma\mathbf{c}_p = (0.0333, 0.7083, 0.2583)$. In this example we could have reached the optimum by going all the way to the end of the feasible line segment (rather than just 90 percent of the way), but only because the feasible polytope happens to be a line segment.

Step 5. We now map the new point $\mathbf{z}^{k+1}$ back to its position in the original simplex, namely $\mathbf{y}^{k+1} = {}^{T-1}(\mathbf{z}^{k+1})$. If the offset objective function value $\mathbf{c}(v^{k+1})^T\mathbf{y}^{k+1}$ is not yet close enough to zero, set $k = k + 1$ and go to Step 1. Otherwise stop.

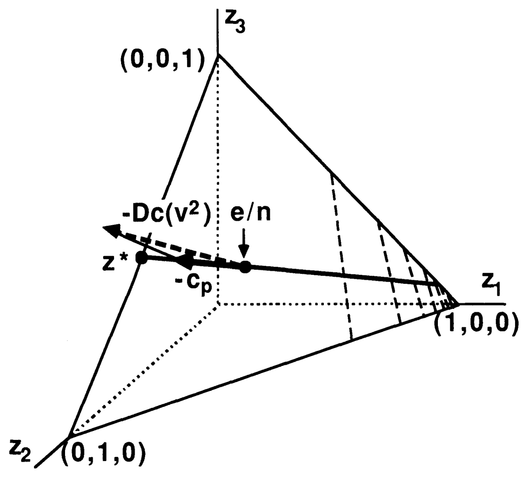

Figure 4: Illustration of the problem of Figure 3 after it has been distorted by a projective transformation in the second iteration of the algorithm. The current solution y^1 has been mapped to the center e/n of the simplex. The feasible polytope (heavy line segment) has moved but remains a polytope, and the objective function contours (dashed lines) are rearranged but remain straight lines.

In the example, $\mathbf{y}^1 = \mathbf{z}^1$, and the value of the original objective function is $\mathbf{c}^T\mathbf{y}^1 = 1.0333$, not far from the optimum of 1. But the offset objective function $\mathbf{c}(v^1)^T\mathbf{y}^1 = 0.176$ is not close enough to zero to terminate the algorithm.

If we carry the example through another iteration, the projective transformation in Step 1 converts the problem of Figure 3 to that of Figure 4. The current point $\mathbf{y}^1$ is mapped to the center $\mathbf{e}/n$ of the simplex in Figure 4. Note that straight lines remain straight lines, but the spacing of the contours is severely distorted, reflecting the nonlinearity of the objective function in (4).

[In Step 2, (5) yields $u^2 = 0.0412$, so that $\bar{v} = 0.9588$. Since $\bar{v} > v^1 = 6/7$, we have improved our lower bound on v^*, and we set $v^2 = 0.9588$, already quite close to $v^* = 1$. We solve (6) to get $u^2 = 0.0133$ and the dual feasible solution $(u^2,v^2) = (0.0133,0.9588)$, which is close to the optimal dual solution (0,1).]

The projected gradient $\mathbf{c}_p$ in Step 3 is $(0.00897, -0.00509, -0.00388)$, and the new point in Step 4 is $\mathbf{z}^2 = (0.0333,0.5035,0.4631)$. This corresponds to the point $\mathbf{y}^2 = T^{-1}(\mathbf{z}^2) = (0.0023,0.7471,0.2506)$ in the original space, at which $\mathbf{c}^T\mathbf{y}^2 = 1.0023$ and $\mathbf{c}(v^2)^T\mathbf{y}^2 = 0.044$.

Putting the Problem in the Required Form

Suppose we are given an arbitrary linear program,

$$\begin{aligned}&\text{minimize } \tilde{\mathbf{c}}^T\mathbf{x}\\ &\text{subject to } \tilde{A}\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq 0\ .\end{aligned} \tag{10}$$

We wish to convert (10) to the form (1) so that it can be solved by the projective scaling method. I will present essentially the conversion proposed by Tomlin [1985], which has the advantage that it yields an interior starting point as well.

We first define m and n such that $\mathbf{x} \in R^{n-3}$ and $\tilde{A}$ is $(m-1) \times (n-3)$. We define $\tilde{\mathbf{y}} = (y_1, \ldots, y_{n-3})$ and rescale the problem by replacing $\mathbf{x}$ with $\tilde{\mathbf{y}} = \mathbf{x}/\sigma$. The scale factor σ is chosen large enough so that we can be sure any feasible solution $\mathbf{x}$ satisfies $\Sigma_j x_j < \sigma$ unless (10) is (for all practical purposes) unbounded. After rescaling, (10) becomes min $\sigma\tilde{\mathbf{c}}^T\tilde{\mathbf{y}}$ subject to $\tilde{A}\tilde{\mathbf{y}} = \mathbf{b}/\sigma$, $\tilde{\mathbf{y}} \geq 0$. Rather than solve this problem, we solve the related problem,

minimize

$$[\sigma\tilde{\mathbf{c}}^T \quad \mathbf{0} \quad \mathbf{0} \quad M\,] \begin{bmatrix}\tilde{\mathbf{y}}\\ y_{n-2}\\ y_{n-1}\\ y_n\end{bmatrix} \tag{11}$$

subject to

$$\begin{bmatrix} \tilde{A} & \mathbf{0} & -n\mathbf{b}/\sigma & n\mathbf{b}/\sigma - \tilde{A}\mathbf{e} \\ \mathbf{0} & 0 & n & 0 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{y}} \\ y_{n-2} \\ y_{n-1} \\ y_n \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}$$

$$\mathbf{e}^T\tilde{\mathbf{y}} + y_{n-2} + y_{n-1} + y_n = 1, \mathbf{y} \geq \mathbf{0} .$$

Here y_{n-2} is a slack variable that absorbs the difference between 1 and the sum of the other variables; y_{n-1}, which is constrained to be equal to $1/n$, is introduced so that $\mathbf{b}/\sigma$ may be brought to the left hand side; and a "Big M" cost M is given to artificial variable y_n to force it to zero when (10) has a feasible solution.

The formulation (11) is contrived so that if $\mathbf{y}$ solves (11), then $\mathbf{x} = \sigma\tilde{\mathbf{y}}$ solves (10) if (10) has a feasible solution. Also the interior point $\mathbf{y} = \mathbf{e}/n$ is a feasible solution of (11). Finally, (11) is in the desired form (1) except for a 1 on the right hand side. This can be corrected by subtracting the last constraint from the next to last constraint:

minimize

$$[\sigma\tilde{\mathbf{c}}^T \quad 0 \quad 0 \quad M] \begin{bmatrix} \tilde{\mathbf{y}} \\ y_{n-2} \\ y_{n-1} \\ y_n \end{bmatrix}$$

subject to (12)

$$\begin{bmatrix} \tilde{A} & \mathbf{0} & -n\mathbf{b}/\sigma & n\mathbf{b}/\sigma - \tilde{A}\mathbf{e} \\ -\mathbf{e}^T & -1 & n-1 & -1 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{y}} \\ y_{n-2} \\ y_{n-1} \\ y_n \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}$$

$$\mathbf{e}^T\tilde{\mathbf{y}} + y_{n-2} + y_{n-1} + y_n = 1, \mathbf{y} \geq \mathbf{0} .$$

Now we have the problem in the desired form.

A "Big M" causes numerical problems in the simplex method, but not here. In a feasible problem y_n soon vanishes, so that the Big M is multiplied by a very small y_n in the product AD in (3).

If, upon solving (12), we find that the artificial variable $y_n > 0$, we know (10) is infeasible. If the slack variable $y_{n-2} = 0$, we know that (10) is unbounded.

As an example I will transform the problem

minimize $x_1 + 2x_2$

subject to $x_1 + x_2 - x_3 = 2$

$3x_1 - x_2 = 0,\ x_1, x_2, x_3 \geq 0$.

Here $m = 3$ and $n = 6$, and if we let $\sigma = 10$, (12) becomes,

minimize $10y_1 + 20y_2 + My_6$

subject to

$$\begin{aligned} y_1 + y_2 - y_3 - 1.2y_5 + 0.2y_6 &= 0 \\ 3y_1 - y_2 - 2y_6 &= 0 \\ -y_1 - y_2 - y_3 - y_4 + 5y_5 - y_6 &= 0 \\ y_1 + y_2 + y_3 + y_4 + y_5 + y_6 &= 1 \end{aligned}$$

$y_1, y_2, y_3, y_4, y_5, y_6 \geq 0$.

The solution is $\mathbf{y} = (0.05, 0.15, 0, 19/30, 1/6, 0)$, which corresponds to $\mathbf{x} = (0.5, 1.5, 0)$ in the original problem.

A variation of this approach, used to solve the problem in Figure 1, is to employ a two-phase technique reminiscent of that generally used for the simplex method. Phase 1 achieves a feasible solution for (10) by setting $\tilde{\mathbf{c}} = \mathbf{0}$ and $v^k = 0$ in every iteration until y_n essentially vanishes in, say, iteration k'. Then an initial value of $v^{k'}$ is got as in Step 0 (replacing A and $\mathbf{c}$ with AD and $D\mathbf{c}$), and Phase 2 proceeds with the algorithm in its original form (that is, using the original $\tilde{\mathbf{c}}$ and calculating v^{k+1} as indicated in Step 2). Figure 1 depicts the simplex and projective iterations for Phase 2 only. The projective iterations are based on minimization of $g(\mathbf{e}/n - \gamma\mathbf{c}_p)$, rather than maximization of γ, in (9).

Complexity of the Algorithm

Much of the interest in the projective scaling algorithm is due to Karmarkar's ingenious proof that its running time is a polynomial function of the problem size even in the worst case. He showed that if n is the number of variables in problem (1) and L is the number of bits used to represent numbers in the computer, the theoretical worst-case running time is $O(n^{3.5}L^2)$. That is, as the problem size increases, the running time tends to be a constant multiple of $n^{3.5}L^2$. This is substantially better than the ellisoid algorithm's worst-case running time of $O(n^6L^2)$.

Appendix 1 explains why the algorithm has complexity $O(n^{3.5}L^2)$. Although this material is relegated to an appendix, it is only a little more technical than the foregoing, and one cannot appreciate the ingenuity of Karmarkar's contribution without understanding it.

Recent Developments

The projective scaling algorithm has sparked an impressive amount of research. One of the most significant developments has been the discovery by Gill et al. [1985] that the projective scaling algorithm belongs to a class of solution methods that have been known for some time, the "projected Newton barrier methods." They show that if one applies a projected barrier method to an LP in the homogeneous form (1) and uses just the right "barrier parameter," one gets an algorithm that is identical to Karmarkar's (Appendix 2). Apparently no one has ever tried using this particular parameter before, however, and no one has shown any barrier method other than Karmarkar's to have polynomial complexity.

Vanderbei et al. [1985], Chandru and Kochar [1986], and others have discussed an interesting modification of the projective scaling algorithm that might be called the "affine scaling algorithm." It dispenses with projective transformations of the solution space and merely rescales the variables so that the current point becomes the point $\mathbf{e} = (1, \ldots, 1)$. In this way, it achieves Karmarkar's objective of keeping the point away from the walls of the feasible polytope. It also has the advantage that the objective function need not be offset to achieve a minimum valve of zero. But convergence is not guaranteed when the optimum is degenerate in the primal problem, and there is no proof of polynomial complexity. Gill et al. remark that this method is related to a barrier method applied to an LP in the form (10).

Curiously, the simplex method performs quite well in practice, despite its dreadful worst-case behavior.

In other developments, Anstreicher [1986] has shown how the projective scaling algorithm relates to fractional programming, and Kojima [1985] has described a test one can perform in the midst of the algorithm to determine whether a variable is going to be basic in the optimal solution. Toné [1985] has proposed a hybrid algorithm that uses the reduced rather than the projected gradient as a search direction whenever possible. D. Bayer of Columbia University and J. Lagarias of AT&T Bell Labs are developing a technique, inspired by differential geometry, that moves through the polytope along a curve rather than along a straight line.

Computational Experience

Karmarkar himself has not released a paper containing computational results. Other investigators have reported some preliminary testing. There seems to be a consensus that the number of iterations grows very slowly with problem size, as Karmarkar predicted. But each iteration poses a nasty least-squares problem. It has become clear that an efficient least-squares routine is key to the success of the projective scaling method.

Tomlin [1985] used versions of the well-known QR method (employing both Householder transformations and Givens rotations) to solve the least-squares problems. In both cases the projective scaling algorithm was substantially slower than the simplex method. The main reason is that whereas the simplex method deals with the sparse matrix $\tilde{A}$ in (10), the projective scaling method must solve the normal equations with a dense matrix AD^2A^T. This matrix is dense because the last two columns of A, as constructed in (10), are dense and propagate a large number of nonzeros in the product AD^2A^T.

There are ways to try to circumvent the density problem. Gill et al. [1985] implemented their projected barrier method by computing a Cholesky factorization of AD^2A^T with dense columns removed from A and using the result as input to the LSQR method of Paige and Saunders [1982]. Their method was competitive with the simplex method on some problems. Shanno [1985] has tried using Fletcher-Powell updates of a similar Cholesky factorization with encouraging results.

Shanno and Marsten [1985] found that they had to solve the normal equations very accurately, or else the current point would become infeasible and convergence lost. They tried to avoid this with a conjugate gradient version of the projective scaling algorithm, as well as an "inexact" projection algorithm, both without success. Aronson et al. [1985] found that the projective scaling algorithm (using the LSQR method) took 14 times longer than the simplex method to solve small, dense, randomly-generated assignment problems.

Vanderbei et al. [1985] report that their affine scaling algorithm requires fewer than one-half as many iterations as the projective scaling method on small, dense problems, with about the same amount of work per iteration. They also say that its computation time is comparable to that of the simplex method on such problems.

A common experience is that the least-squares problem becomes ill-conditioned as the optimum is approached. The problem is especially acute when the optimal point is degenerate, and the reason is clear. At a degenerate extreme point fewer than m of the variables z_j are nonzero, which means that fewer than m columns of AD in (6) and (7) are nonzero. Thus AD is not of full row rank, so that AD^2A^T is singular. Perhaps z_j's destined to vanish can be identified and removed from the problem before AD^2A^T becomes ill-conditioned. Shanno and Marsten found that it is risky simply to remove variables that approach zero, but Kojima's basic variable test may prove useful. Also, Karmarkar has pointed out in conversation that although $(AD^2A^T)^{-1}$ "blows up" as $\mathbf{y}$ approaches a degenerate solution, $\mathbf{u} = (AD^2A^T)^{-1}AD\mathbf{c}$ does not. In other words, the problem is not ill-conditioned, and it should be possible to design a numerically stable algorithm.

Just before going to press I received documentation of some very encouraging test results obtained at Berkeley by I. Adler et al. [1986]. This public domain implementation was written by Adler and several graduate students with Karmarkar's assistance. It solved 30 real-world LP's an average of three times faster than the state-of-the-art simplex routine in MINOS 4.0. Problem sizes ranged from 27 rows, 51 columns to 1151 rows, 5533 columns. Run times varied from 70 percent as fast as simplex to 8.6 times faster, and the ratio shows a clear tendency to increase with problem size. MINOS is not the fastest simplex code, but the clearly faster ones, such as IBM's MPSX, have the unfair advantage of being written in assembly language. E. R. Barnes has reported similar results at IBM (see Kozlov [1985]), but I have yet to receive a technical paper documenting them.

Adler's implementation is a variation of the affine scaling algorithm. It maximizes $\mathbf{c}^T\mathbf{x}$ subject to the inequality constraints $A\mathbf{x}\leq\mathbf{b}$, where $\mathbf{x}$ is not restricted to be nonnegative. (This form rarely occurs in practice, but it is precisely the dual of the standard form (10); Adler therefore solves a practical LP by solving its dual). The implementation adds a vector $\mathbf{s}$ of slack variables to obtain the constraints $A\mathbf{x}+\mathbf{s}=\mathbf{b}$, $\mathbf{s}\geq\mathbf{0}$, and applies the affine transformation only to the slacks. This makes it possible to compute the orthogonal projection only approximately without losing feasibility.

Concluding Remarks

I have seen no evidence that the projective scaling method can beat the simplex method by a factor of 50, as originally claimed. But there is little doubt that it or its variations can outperform the simplex method on a large class of problems. Already one implementation of it (actually an affine scaling method that differs substantially from Karmarkar's original) runs several times faster than the simplex routine in MINOS on problems having a few thousand variables. More importantly, its speed relative to the simplex method increases with problem size. There is every reason to believe that implementations will continue to improve. After all, experts honed the simplex method for decades, and similar attention should benefit its rival.

Whatever the eventual outcome, it is clear that one can't spend an afternoon writing a straightforward implementation of the projective scaling method and get something that beats the simplex method. Sophisticated numerical mathematics is as important as the underlying method.

People often ask how one can get a basic optimal solution, a dual solution, and sensitivity analysis out of the projective scaling method. One approach is to use a simplex postprocessor. It would begin by applying a routine available in many mathematical programming systems (called BASIC in MPSX, for instance) to convert the projective scaling method's solution to an equally good basic solution; for a description of the method see section 2.7 of Benichou et al. [1977]. Then one or more simplex iterations could be carried out. This approach has two advantages. Since convergence may be slow in the last few projective iterations, it may pay to let the simplex method finish the job. Also, the final simplex iteration supplies the dual variables and sensitivity analysis to which we are accustomed. Tomlin [1985] outlines some difficulties one may encounter.

Karmarkar has suggested that the projective scaling algorithm may have other advantages, yet untested. It may be useful for nonlinear programming; even in the linear case it minimizes a nonlinear function. It may perform well when tailored to the special structure of certain linear programming problems, such as multicommodity flow problems. Finally, it may lead to a decomposition approach more effective than Dantzig-Wolfe decomposition. Such an approach would presumably use decomposition to solve the normal equations in each iteration. It may therefore escape the slow convergence that often characterizes Dantzig-Wolfe decomposition.

Whatever may be its practical value, the projective scaling algorithm represents a substantial theoretical contribution, both for the novelty of the idea and its improvement over the worst-case complexity of the ellipsoid algorithm. The discovery that it is formally equivalent to a projected Newton barrier method does not, in my opinion, mitigate this contribution. The equivalence holds only when one packs into the barrier parameter a good deal of problem-solving strategy that is unrelated to the motivation underlying barrier methods. It is unlikely, after all, that anyone would have soon discovered a barrier method of polynomial complexity without the projective scaling method as a guide.

Beyond this, Karmarkar's work has inspired a flurry of research papers. Something akin to a general rethinking of mathematical programming may grow out of this activity and lead to better methods, whether or not Karmarkar's original algorithm survives.

The projective scaling method has made another sort of contribution. Ours is an age when technical research is dissipated into minute specialties. It is refreshing and exciting to find a topic that engages an entire technical community, and Karmarkar has provided one.

Acknowledgments

In writing this paper I benefited from conversations with M. Haimovich, J. K. Ho, N. Karmarkar, J. Lagarias, R. D. McBride, M. J. Todd, and L. E. Trotter. I also wish to thank G. L. Lilien and four anonymous referees for very helpful suggestions.

APPENDIX 1: Proof of Polynomial Complexity

I wish to explain why the projective scaling algorithm has $O(n^{3.5}L^2)$ complexity. (See Padberg [1986] for another proof.) I will consider Karmarkar's original algorithm, which assumes that the objective function has a minimum value of zero. Todd and Burrell [1985] have modified the proof to show that their method, which puts no restriction on the value of the objective function, likewise has polynominal complexity.

Karmarkar showed that each iteration of the algorithm has theoretical running time $O(n^{2.5}L)$, which is essentially the time required to solve the least-squares problem (7). To get an overall running time of $O(n^{3.5}L^2)$, he must therefore establish that the number of iterations is at most $O(nL)$.

A problem is considered solved when the original value $\mathbf{c}^T\mathbf{y}^\circ$ of the objective function in (1) is reduced by a factor of 2^L, since 2^{-L} is the precision of the computer. Thus the problem is solved in iteration k if $\mathbf{c}^T\mathbf{y}^k/\mathbf{c}^T\mathbf{y}^\circ \leqslant 2^{-L}$. Karmarkar must show that the problem is solved when $k = O(nL)$.

It would be nice if one could show that the algorithm shrinks the objective function $\mathbf{c}^T\mathbf{y}^k$ to at most $\mathbf{c}^T\mathbf{y}^k(e^{-\delta/n})$ each iteration, where $\sigma > 0$ is some constant. This is equivalent to showing that it reduces $n\ ln\ \mathbf{c}^T\mathbf{y}^k$ to $n\ ln\ \mathbf{c}^T\mathbf{y}^k - \delta$ each iteration (where $ln\ x = log_e x$). Then after k iterations we would have $n\ ln\ \mathbf{c}^T\mathbf{y}^k \leq n\ ln\ \mathbf{c}^T\mathbf{y}^\circ - k\delta$, or $\mathbf{c}^T\mathbf{y}^k/\mathbf{c}^T\mathbf{y}^\circ \leq e^{-k\delta/n} = 2^{-k(ln 2)/n}$. To make the exponent of 2 equal to $-L$ and get the desired precision, we could set $k = nL/\delta ln2 = O(nL)$. This would verify that $k = O(nL)$ iterations are enough.

But Karmarkar could not prove that the algorithm reduces $n\ ln\ \mathbf{c}^T\mathbf{y}$ by a constant amount δ each iteration. Instead, he ingeniously suggested that the algorithm does reduce the following "potential function" by δ each iteration:

$$f(\mathbf{y}) = \sum_{j=1}^{n} ln\ \mathbf{c}^T\mathbf{y}/y_j$$

$$= n\ ln\ \mathbf{c}^T\mathbf{y} - \sum_{j=1}^{n} \ln y_j .$$

This is all we need, because if we reduce $f(\mathbf{y})$ by δ each iteration and hence by $k\delta$ over k iterations, then we simultaneously reduce $n\ ln\ \mathbf{c}^T\mathbf{y}$ by at least $k\delta$ over the k iterations. To see why, note that the second term $-\Sigma_j\ ln\ y_j$ of $f(\mathbf{y})$ takes its minimum value in the very first iteration, when $\mathbf{y} = \mathbf{e}/n$. Thus if we reduce $f(\mathbf{y})$ by $k\delta$ over k iterations, we reduce $n\ ln\ \mathbf{c}^T\mathbf{y}$ by even more than $k\delta$. It is enough, then, to show that we reduce $f(y)$ by at least δ in each iteration.

To show this, Karmarkar first observes that when the problem is transformed by T, the potential function assumes exactly the same form except for the addition of a constant; it becomes $g(\mathbf{z}) = n\ ln\ \mathbf{c}^TD\mathbf{z} - \Sigma_j ln\ z_j +$ constant, where $g(\mathbf{z}) = f(\mathbf{y})$ when $\mathbf{z} = T(\mathbf{y})$. This means that if $n\ ln\ \mathbf{c}^TD\mathbf{z} - \Sigma_j ln\ z_j$ drops by δ in the transformed space, then the original $f(\mathbf{y})$ drops by δ in the original space.

Now we reach the heart of the argument. At any given iteration we begin with the current point at the center $\mathbf{e}/n$ of the transformed simplex. We noted in Section 4 that we can always move it a distance equal to αr without forcing any variable to zero, where r is the radius of a sphere inscribed in the simplex and $0 < \alpha < 1$. The optimal point lies somewhere in the simplex and therefore no further away from the center than the vertices of the simplex. The distance of the vertices from the center is equal to the radius nr of a sphere circumscribed about the simplex. So, we can always move at least $\alpha r/nr = \alpha/n$ of the way to the optimal point in each iteration. Thus we can reduce the linear function $\mathbf{c}^TD\mathbf{z}$ (which is zero at the optimal point) to at most $\mathbf{c}^TD\mathbf{z}(1 - \alpha/n) \leq \mathbf{c}^TD\mathbf{z}(e^{-\alpha/n})$ each iteration. This means that we can reduce the first term $n\ ln\ \mathbf{c}^TD\mathbf{z}$ of $g(\mathbf{z})$ by at least a constant α each iteration.

But will the second term $-\Sigma_j ln\ z_j$ of $g(\mathbf{z})$ increase enough to offset the reduction in the first term? It will not if α is small enough, say $\alpha = 1/3$. To see this, note that when we move from point $\mathbf{e}/n$

to $\mathbf{z}^{k+1}$, the increase in $-\Sigma_j ln\ z_j$ is $-\Sigma_j ln\ z^{k+1} + \Sigma_j ln(1/n) = -\Sigma_j ln(z_j^{k+1})$. But $\mathbf{z}^{k+1}$ satisfies $\|\mathbf{z}^{k+1} - \mathbf{e}/n\| \leq \alpha r$ and $\mathbf{e}^T\mathbf{z}^{k+1} = 1$, and Karmarkar showed (using calculus, and so forth) that any such point also satisfies $-\Sigma_j ln(nz_j^{k+1}) \leq \beta^2/2(1-\beta)$, where $\beta = \alpha[n/(n-1)]^{1/2}$. For large n we have $\alpha \approx \beta$, so that the increase in $-\Sigma_j ln(nz_j)$ is at most about $\alpha^2/2(1-\alpha)$. Thus in each iteration we can guarantee a reduction in $f(\mathbf{y})$ of at least $\delta \approx \alpha - \alpha^2/2(1-\alpha)$. As expected, δ is positive for small α; for instance, $\delta \approx 1/4$ when $\alpha = 1/3$.

There are two keys to the success of Karmarkar's argument. One is the geometrical fact that the radii of spheres circumscribing and inscribing a simplex bear a ratio n equal to the dimension of the space. This allows the algorithm to reduce the objective function by about $1/n$ of its current value, on the average, each iteration, so that the number of iterations is proportional to n. To demonstrate this Karmarkar uses the other key to his success, the ersatz objective function $f(\mathbf{y})$.

APPENDIX 2: Interpretation as a Barrier Method

It turns out that the projective algorithm belongs to a class of solution methods that have been known for some time, the "projected Newton barrier methods." This was demonstrated by Gill et al. [1985] in one of the more significant developments since Karmarkar introduced his algorithm.

One can solve the homogeneous linear program (1) with a barrier method by writing it in the following way:

$$\begin{aligned} &\text{minimize } c^T\mathbf{y} - \mu \sum_j ln\ y_j \qquad (13)\\ &\text{subject to } A\mathbf{y} = 0\\ &\qquad \mathbf{e}^T\mathbf{y} = 1,\ \mathbf{y} \geq \mathbf{0} \end{aligned}$$

Here $\Sigma_j\ ln\ y_j$ is a barrier function, and μ is the barrier parameter. Since $ln\ y_j$ becomes very negative as y_j approaches zero, the barrier function has a built-in incentive to observe the nonnegativity constraints $y_j \geq 0$. To ensure convergence, μ must go to zero as one nears the optimum.

Since now all of the constraints are equality constraints, we can solve problem (13) with a variant of Newton's method. In each iteration we orthogonally project the usual Newton search direction onto the space satisfying the constraints, so as to obtain a feasible search direction.

Because Karmarkar's algorithm involves a similar orthogonal projection (Step 3) and a "potential function" that closely resembles the objective function in (13), one might suspect a connection between the two algorithms. The connection is not obvious, but Gill et al. demonstrated that for a particular choice of barrier parameter, namely

$$\mu = (\mathbf{y}^k)^T(\mathbf{c}_p - \mathbf{c}^T\mathbf{y}^k\mathbf{e}/n), \qquad (14)$$

the barrier and projective methods are identical.

Here we see, incidentally, that if the objective function $\mathbf{c}^T\mathbf{y}$ has a minimum value of zero (as Karmarkar requires), μ as defined in (14) will go to zero, as it must to guarantee convergence.

References

Adler, I.; Resende, M. G. C.; and Veiga, G. 1986, "An implementation of Karmarkar's algorithm for linear programming," Dept. of Industrial Engineering and Operations Research, University of California, Berkeley, California 94720.

Anstreicher, K. M. 1986, "A monotonic projective algorithm for fractional linear programming," Yale School of Organization and Management, New Haven, Connecticut, to appear in a special issue of *Algorithmica.*

Aronson, J.; Barr, R.; Helgason, R.; Kennington, J.; Loh, A.; and Zaki, H. 1985, "The projective transformation algorithm by Karmarkar: A computational experiment with assignment problems," Department of Operations Research Technical Report 85-OR-3 (August revision), Southern Methodist University, Dallas, Texas 75275.

Benichou, M.; Gauthier, L. M.; Hentges, G.; and Ribiere, G. 1977, "The efficient solution of large-scale linear programming problems — Some algorithmic techniques and computational results," *Mathematical Programming*, Vol. 13, No. 3, pp. 280–322.

Chandru, V. and Kochar, B. S. 1986, "A class of algorithms for linear programming," Research Memorandum 85–14, School of Industrial Engineering, Purdue University, West Lafayette, Indiana 47907.

Dantzig, George B. 1963, *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey.

Gill, P. E.; Murray, W.; Saunders, M. A.; Tomlin, J. A.; and Wright, M. A., ca. 1985, "On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method," Report SO6 85-11, Systems Optimization Laboratory, Stanford University, Stanford, California, to appear in *Mathematical Programming*.

Hačijan, L. G. 1979, "A polynomial algorithm in linear programming," *Doklady Akademii Nauk SSSR*, Vol. 244, pp. 1093–1096, translated in *Soviet Mathematics - Doklady*, Vol. 20, No. 1, pp. 191–194.

Karmarkar, N. 1984a, "A new polynomial-time algorithm for linear programming," *Proceedings of the 16th Annual ACM Symposium on the Theory of Computing*, pp. 302–311.

Karmarkar, N. 1984b, "A new polynominal-time algorithm for linear programming," *Combinatorica*, Vol. 4, No. 4, pp. 373–395.

Karmarkar, N. 1984c, "A new polynomial-time algorithm for linear programming," presentation at Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Karmarkar, N. 1985, "Further developments in the new polynomial time algorithm for linear programming," presented at the 12th International Symposium on Mathematical Programming, Cambridge, Massachusetts.

Kojima, M. 1985, "Determining basic variables of optimum solutions in Karmarkar's new LP algorithm," Research Reports on Information Sciences No. B-164, Department of Information Sciences, Tokyo Institute of Technology.

Kolata, Gina 1984, "A fast way to solve hard problems," *Science*, Vol. 225, No. 4668 (21 September), pp. 1379–1380.

Kozlov, A. 1985, "The Karmarkar algorithm: Is it for real?", *SIAM News*, Vol. 18, No. 6 (November), pp. 1–4.

Padberg, M. 1986, "A different convergence proof of the projective method for linear programming," *Operations Research Letters*, Vol. 4, No. 6, pp. 253–257.

Paige, C. C. and Saunders, M. A. 1982, "Algorithm 583 LSQR: Sparse linear equations and least-squares problems," *ACM Transactions on Mathematical Software*, Vol. 8, No. 2, pp. 195–209.

Shanno, D. F. 1985, "Computing Karmarkar projections quickly," Graduate School of Administration Working Paper 85-10, University of California, Davis, California.

Shanno, D. F. and Marsten, R. E. 1985, "On implementing Karmarkar's method," College of Business and Public Adminstration Working Paper 85-01 (September revision), University of Arizona, Tucson, Arizona.

Todd, M. J. and Burrell, B. P. 1985, "An extension of Karmarkar's algorithm for linear programming using dual variables," School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York, to appear in a special issue of *Algorithmica*.

Tomlin, J. A. 1985, "An experimental approach to Karmarkar's projective method for linear programming," Ketron, Inc., Mountain View, California, to appear in *Mathematical Programming Studies*.

Toné, K. 1985, "A hybrid method for linear programming," working paper, Graduate School for Policy Science, Saitama University, Urawa, Saitama 338, Japan.

Vanderbei, R. J.; Meketon, M. S.; Freedman, B. A., ca. 1985, "A modification of Karmarkar's linear programming algorithm," AT&T Bell Laboratories, Holmdel, New Jersey, to appear in a special issue of *Algorithmica*.