Ann. Rev. Comput. Sci. 1987. 2: 119–45

LINEAR PROGRAMMING (1986)

Nimrod Megiddo

IBM Research, Almaden Research Center, San Jose, California 95120-6099, and Statistics Department, Tel Aviv University, Tel Aviv, Israel

1. INTRODUCTION

Linear programming has enormous practical importance. Perhaps this is why it is frequently the subject of news reports. Existing methods for solving linear programming problems have been applied successfully in many areas. Improvements by orders of magnitude in the efficiency of solution methods would open up many new application areas with substantial economic gains.

The traditional and still most widely used tool for solving linear programming problems is the simplex method of George Dantzig (1963). This method is in fact a whole family of algorithms. Commercial software packages for linear programming consist of auxiliary procedures for handling particular types of problems. The efficiency of the simplex method can be appreciated by anyone who tries to solve a linear programming problem on any computer or even by hand. Thus the search for alternative algorithms for linear programming is motivated not by frustration but by theoretical considerations. It has been observed in practice that the number of steps performed by several variants of the simplex method grew somewhat linearly with the number of rows in the matrix, and much more slowly with the number of columns. In highly degenerate problems the numbers of steps was larger, and "anti-degeneracy" features were developed. Theoreticians attempted to prove nontrivial upper bounds on the number of steps until Klee & Minty (1970) constructed a sequence of problems where several variants of the simplex method required an exponential number of steps. Following Klee & Minty, several additional variants were shown by other people (see Megiddo 1986a for references) to require exponential time in the worst case. The notion of a "variant of the simplex method" is not well-defined, and it is still not known whether every variant (in any reasonable sense of the word) requires exponential

8756–7016/87/1115–0119$02.00

time in the worst case. Anyway, practical experience suggests that the bad cases are extremely rare.

Computational complexity theory is concerned mostly with the worst-case performance of algorithms. The worst-case measure has theoretical advantages, but its main deficiency is that it is one-sided: If an algorithm works well in the worst case then it always works well; however, proof that an algorithm is slow in the worst case does not tell against its performance in practice. The simplex method is perhaps the best counterexample. Nevertheless, it is interesting from a theoretical point of view to classify problems according to their worst-case complexity.

It was not known until 1979 whether the linear programming problem was in the class P of problems solvable in polynomial time in terms of the input size. The size of the input for a problem with integer coefficients is measured as the total number of bits in the binary representation of the problem. Khachiyan (1979) proved that an algorithm by Yudin & Nemirovsky (1976), originally proposed for nonlinear optimization, had a polynomial upper bound. The algorithm has been tried without much success on linear programming problems. It performs relatively better on nonlinear optimization problems.

In the fall of 1984 the news media reported that a new algorithm for linear programming by N. Karmarkar performed much better than the simplex method. The reports stirred up a great interest in applying nonlinear methods for solving linear programming problems. Extensions and variations on Karmarkar's algorithm have been proposed. The relation of the new algorithms to the classical methods of nonlinear optimization is still under investigation. Apparently, the idea of applying such methods in linear programming has not been seriously pursued until recently. Most of the development of nonlinear programming occurred before complexity theory started to flourish. Because the functions involved lack well-defined properties, it is usually hard to analyze algorithms for nonlinear optimization. On the other hand, if all the functions are linear one has a nice framework for such an analysis. The work of Khachiyan (1979) pioneered in this direction. Karmarkar's work drew attention to methods of nonlinear programming.

At present it is not clear whether nonlinear methods will eventually replace the simplex method as the standard tool for solving linear programming problems in practice. However, for certain classes of problems, with sparse matrices of favorable structures, such methods are superior.

This paper focuses on work in linear programming from a nonlinear point of view. I do not discuss here other areas of linear programming in which there has been active research in recent years. Probabilistic analysis of simplex-type algorithms became popular with the work of Borgwardt

(1982a,b) and Smale (1983). Further work has also been done (Adler & Megiddo 1985; Todd 1986; Adler et al 1983a,b; Megiddo 1986c; see also Megiddo 1986a). Another interesting area in linear programming is the search for strongly polynomial algorithms (Tardos 1985; Frank & Tardos 1985; see also Megiddo 1986a).

The organization of this survey is as follows: In Section 2 I describe Karmarkar's algorithm; in Section 3 discuss the linear rescaling algorithm with observations on its induced vector field; in Section 4 review some methods of nonlinear programming applied to linear programming; and in Section 5 survey recent algorithms for linear programming that involve Newton's method.

2. KARMARKAR'S ALGORITHM

The algorithm published by Karmarkar (1984), is stated with respect to the linear programming problem given in form

$$\begin{aligned} &\text{Minimize } \boldsymbol{c}^T\boldsymbol{x} \\ &\text{subject to } A\boldsymbol{x} = 0 \\ &\qquad\qquad \boldsymbol{e}^T\boldsymbol{x} = 1 \\ &\qquad\qquad \boldsymbol{x} \geq 0, \end{aligned}$$

where $A \in R^{m\times n}$ $(1 \leq m \leq n)$; $\boldsymbol{x}, \boldsymbol{c} \in R^n$; and $\boldsymbol{e} = (1, \ldots, n)^T \in R^n$. It is assumed that $A\boldsymbol{e} = 0$ so the point $\boldsymbol{x}^0 = (1/n)\boldsymbol{e}$ is interior relative to the linear subspace $\{A\boldsymbol{x} = 0\}$. It is also assumed that the optimal value of the objective function is zero. The idea of the algorithm is explained with a projective transformation as follows. Denote

$$\bar{A} = \begin{pmatrix} A \\ \boldsymbol{e}^T \end{pmatrix}.$$

If $\boldsymbol{x} \in R^n$ is such that $A\boldsymbol{x} = 0$, $\boldsymbol{e}^T\boldsymbol{x} = 1$, and $\boldsymbol{x} > 0$ then the algorithm would move from $\boldsymbol{x}$ to a new point $\boldsymbol{x}'$ which is computed as a function of $\boldsymbol{x}$ as follows. Denote by

$$D_x = \text{Diag}(x_1, \ldots, x_n)$$

a diagonal matrix whose diagonal entries are the components of $\boldsymbol{x}$. Consider a transformation of space

$$T_x: R^n \to R^n$$

given by

$$T_x(y) = \frac{1}{e^T D_x^{-1} y} D_x^{-1} y.$$

Thus, $T_x(x) = (1/n)e$. The mapping T_x is a *projective rescaling transformation* characterized by the property that it leaves all the vertices of the unit simplex in their places and moves the point x to the center of the simplex.

Karmarkar used a "potential function" in the design and analysis of his algorithm. The function is the following:

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

Given the transformation T_x, the transformed potential function $\psi_x(z)$ is the following:

$$\begin{aligned}\psi_x(z) &= \psi(T_x^{-1}(z)) \\ &= n \ln c^T D_x z - \sum_{j=1}^{n} \ln z_j - \sum_{j=1}^{n} \ln x_j.\end{aligned}$$

The algorithm moves in the transformed space in a steepest descent direction relative to the transformed potential function. The gradient of $\psi_x(z)$ at the point $z = (1/n)e$ is equal to $D_x c$. In the transformed space, the direction of movement, that is, the direction of the vector $T_x(x') - T_x(x)$, is opposite to the direction of the gradient of the potential function, projected into the space of the problem. This direction is obtained by projecting the vector $-D_x c$ orthogonally into the nullspace of the matrix

$$\bar{A} = \begin{pmatrix} AD_x \\ e^T \end{pmatrix}.$$

The result of this projection is the following vector:

$$\eta = -(I - \bar{A}^T(\bar{A}\bar{A}^T)^{-1}\bar{A})Dc.$$

Thus, the new point x' has the form

$$x' = T_x^{-1}\left(\frac{1}{n}e + \frac{\gamma}{\|\eta\|}\eta\right)$$

where γ is a positive scalar. The size of the actual step in practice is subject to variations. In theory, the choice

$$\gamma = \frac{1}{4\sqrt{n(n-1)}}$$

leads to a polynomial-time algorithm. Other results on the choice of γ are known (Blair 1986; Padberg 1985a,b; Todd & Burrell 1986).

The role of the potential function is crucial in the theoretical analysis of the algorithm. It was shown that in each step the value of this function decreases by at least some fixed amount δ independent of the data. The initial value of the potential function is $O(nL)$ where L is bounded by the length of the representation of the problem in binary encoding. The algorithm can be terminated when the value of this function is less than $-O(nL)$ since then an optimal basis can be computed directly from the current interior point. Thus, the algorithm terminates in $O(nL)$ iterations. Interestingly, the constant reduction of potential corresponds to linear convergence, which is considered dissatisfactory in the traditional theory but "good" from the point of view of contemporary complexity theory.

If the iterations are performed using standard methods of linear algebra then each takes $O(n^3)$ arithmetic operations. Thus, the total number of arithmetic operations is $O(n^4L)$. However, to determine the complexity of the algorithm in bit operations, we still have to determine the precision in which the arithmetic operations need to be performed. Karmarkar (1984) claimed that precision of $O(L)$ bits suffices. However, Renegar (1986) pointed out the incompleteness of the argument in Karmarkar's paper, suggesting that the required precision is actually $O(nL)$. Of course, in practice one only works with fixed precision. Karmarkar also showed how to perform the iterations of his algorithm in an average of $O(n^{2.5})$ arithmetic operations per iteration. This theoretical improvement does not seem to have a practical value since on medium-size problems this average is not realized within the first few dozens of iterations (the typical number of iterations required for solving practical problems).

3. THE LINEAR RESCALING ALGORITHM

After the publication of Karmarkar's algorithm many people independently suggested the idea of using a *linear* rather than projective rescaling transformation. We mention here only two papers (Barnes 1985 and Vanderbei et al 1986), where convergence proofs of the resulting algorithms were provided. Consider the linear programming problem in the standard form (*SF*):

$$\begin{aligned} &\text{Maximize } \boldsymbol{c}^T\boldsymbol{x} \\ (SF)\quad &\text{subject to } \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ &\boldsymbol{x} \geq 0. \end{aligned}$$

Also, assume a point x^0 is known such that $Ax^0 = b$ and $x^0 > 0$. Given $x \in R^n$ such that $Ax = b$ and $x > 0$, consider the following linear transformation

$$T_x : R^n \to R^n$$

given by

$$T_x(y) = D_x^{-1} y.$$

Obviously, $T_x(x) = e = (1, \ldots, 1)^T$, and T_x is actually *characterized* as a linear transformation with this property. In the transformed space the problem becomes

$$\text{Maximize } c^T D_x y$$

$$\text{subject to } AD_x y = b$$

$$y \geq 0.$$

The algorithm moves in the transformed space in the direction of the gradient projected into the space of the problem. The gradient is $D_x c$. The projection of this vector on the nullspace of the matrix AD_x is

$$\eta = (I - D_x A^T (AD_x^2 A^T)^{-1} AD_x) D_x c.$$

The new point x' is of the form

$$x' = D_x(e + \gamma(x)\eta)$$

where $\gamma(x)$ is a scalar.

The linear rescaling algorithm was stated by both Barnes (1985) and Vanderbei et al (1986) with respect to problems in standard form. For reasons discussed below, it seems that there is an advantage to work with problems in the inequality form (the dual of the standard form):

$$\text{Minimize } c^T x$$

$$(D) \quad \text{subject to } Ax \geq b$$

where $A \in R^{m \times n}$; $b \in R^m$; and $c, x \in R^n$ $(m \geq n)$. One can develop an analogous algorithm for (D), based on the principle suggested by Barnes (1985). Suppose x is an interior feasible point. Let

$$E = \left\{ y : \sum_{i=1}^{m} \left(\frac{A_i y - b_i}{A_i x - b_i} - 1 \right)^2 \leq 1 \right\}$$

be an ellipsoid contained in the feasible domain. Note that

$$E = \{y : \|D_s^{-1}(Ay - b) - e\| \leq 1\},$$

where

$$D_s = D_s(x) = \text{Diag}(A_1x - b_1, \ldots, A_mx - b_m).$$

The algorithm picks a direction v of movement towards the minimum of the function c^Ty over E. The vector v can thus be found by solving the following optimization problem:

Minimize $c^T(x+v)$

subject to $\|D_s^{-1}[A(x+v) - b] - e\| = 1.$

Since $D_s^{-1}(Ax - b) = e$, it follows that this problem is equivalent to

Minimize c^Tv

subject to $\|D_s^{-1}Av\| = 1.$

However, we are interested only in the direction of the vector v, so we can write the following set of equations for the optimality conditions:

$$(A^TD_s^{-2}A)v = c.$$

Thus

$$v = (A^TD_s^{-2}A)^{-1}c.$$

The latter looks simpler than the formula for the problem in standard form, even though the effort involved in computing the direction is not much different. The advantage of the inequality form is that it is numerically much easier to satisfy inequalities than equalities. Specifically, if x satisfies $Ax > b$ and we move in a direction $\tilde{v}$ rather than v, then we still maintain feasibility. On the other hand, in the standard form the approximate direction has to satisfy $A\tilde{v} = 0$ with relatively high accuracy, or else the point becomes infeasible.

It is interesting to observe that the same search direction can be obtained for (D) by transforming it to standard form with surplus variables replacing the original x variables (Megiddo & Shub 1986). Also, both in standard form and the form (D), this direction corresponds to the choice of $\mu = 0$ in the logarithmic barrier technique (Gill et al 1985; Megiddo & Shub 1986) (see Section 4).

Let us return to problems in standard form (SF). As pointed out in (Vanderbei et al 1986), one can associate dual values $w = w(x)$ with any feasible point x so that (under nondegeneracy assumptions) when x tends to the optimal solution, $w(x)$ tends to the dual optimal solution. Specifically,

$$w(x) = (AD_x^2A^T)^{-1}AD_x^2c.$$

It is worthwhile to note that this vector arises naturally as the vector of Lagrange multipliers in the projection problem in the transformed space:

$$\text{Minimize } \frac{1}{2}\|D_x c - \eta\|^2$$

subject to $AD_x\eta = 0$.

If λ is the vector of multipliers then

$$\begin{aligned} \eta \quad & -D_x A^T\lambda = D_x c \\ AD_x\eta \quad & = 0 \end{aligned}$$

so, by eliminating η we get

$$(AD_x^2A^T)\lambda = AD_x^2c.$$

The same vector is also used by Todd & Burrell (1986), who derive dual variables in extensions of Karmarkar's algorithm.

It is not known whether the linear rescaling algorithm runs in polynomial time. Although it was suggested as a variation on Karmarkar's algorithm, it is not clear that it shares similar properties—e.g. a guaranteed amount of progress in a certain precise sense in each iteration. Moreover, there are indications that in certain cases the behavior of this algorithm is drastically different from the projective rescaling algorithm. Megiddo & Shub (1986) analyze the differences by considering trajectories in the vector fields corresponding to both algorithms. First, they show that there exist linear rescaling trajectories that visit all the vertices of the Klee-Minty cube within any prescribed $\varepsilon > 0$. Suppose we let a point on a trajectory tend to a boundary point. The limit of a linear rescaling trajectory may have an exponential number of breakpoints. On the other hand, projective rescaling trajectories may have only a linear number of such breakpoints in the limit. Intuitively, the projective rescaling trajectories can in the worst case behave as follows. Suppose the starting point x^0 of a trajectory tends to a point in the relative interior of a face Φ of the feasible polyhedron. If Φ contains the optimum then the limit of the trajectory stays in Φ and converges to the optimal solution. If Φ does not contain the optimum then there exists a point $\sigma(\Phi)$ in the relative interior of Φ with the following property: For every x interior to Φ and every $\varepsilon > 0$, if the starting point x^0 is sufficiently close to x then the trajectory visits the ε-neighborhood of $\sigma(\Phi)$. The only possible breakpoints in the limit of projective rescaling trajectory are the points $\sigma(\Phi)$. Moreover, if $\sigma(\Phi_1), \ldots, \sigma(\Phi_r)$ are breakpoints in the limit of one trajectory then necessarily

$$\dim \Phi_1 < \dim \Phi_2 < \cdots < \dim \Phi_r < n.$$

It follows that there can be no more than n such breakpoints on one trajectory.

Current implementation efforts seem to concentrate on the linear rescaling algorithm. Encouraging computational experiences with the linear rescaling algorithm were reported recently (Chen 1986; Adler et al 1986). Adler et al implemented the algorithm in its dual version. Remarkable results were also reported by Karmarkar & Sinha (1985), although it was not clear exactly what algorithm they implemented.

4. APPLICATIONS OF TRADITIONAL METHODS OF NONLINEAR PROGRAMMING

The notion of a potential function is of course well known in mathematical physics. Many algorithms for nonlinear programming (Fiacco & McCormick 1967) were developed with intuition drawn from physics. The use of the function $\psi(x)$ in Karmarkar's algorithm is very much related to barrier function techniques in nonlinear programming. A formal equivalence to the projected Newton method with a logarithmic barrier function was pointed out by Gill et al (1985). When this technique is applied to the linear programming problem, the following nonlinear optimization problem is considered:

$$\text{Minimize } F_\mu(\boldsymbol{x}) = c^T\boldsymbol{x} - \mu \sum_j \ln x_j$$

$$\text{subject to } A\boldsymbol{x} = \boldsymbol{b}$$

$$\boldsymbol{x} > 0$$

where $\mu > 0$ is a scalar. Given $\boldsymbol{x}$ (such that $A\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{x} > 0$) and μ, the Newton direction is the direction of the vector

$$\boldsymbol{v} = (\nabla^2 F_\mu(\boldsymbol{x}))^{-1} \nabla F_\mu(\boldsymbol{x}).$$

This direction is projected orthogonally into the nullspace of the matrix A to obtain a direction of movement from $\boldsymbol{x}$ along which the points stay in the flat $\{A\boldsymbol{x} = \boldsymbol{b}\}$. The parameter μ is chosen either as the current element in a predetermined sequence or as a function of $\boldsymbol{x}$, tending to zero during the execution of the algorithm. The method is attributed to Frisch (1955). A related method, the "method of centers" proposed by Huard (1967), chooses the "parameter" with respect to the current point as follows. Suppose the problem is

$$\text{Maximize } f(\boldsymbol{x})$$

$$\text{subject to } g_i(\boldsymbol{x}) \geq 0 \quad (i = 1, \ldots, m),$$

and assume a point x^0 is available such that $g_i(x^0) > 0$ $(i = 1, \ldots, m)$. Suppose the algorithm has reached the point ξ. It then attempts to maximize the function

$$F_\xi(x) = \ln(f(x) - f(\xi)) + \sum_i \ln g_i(x).$$

Here the constraints are represented by logarithmic barriers and the objective function is represented through the logarithm of the improvement relative to the current point. When the current point gets close to optimal, the latter becomes the dominant term of F_ξ. When this idea is applied to problems in the form (SF), the iterative step attempts to solve

$$\text{Maximize } \ln(c^T x - c^T \xi) + \sum_j \ln x_j$$

$$\text{subject to } Ax = b.$$

In practice the method of local optimization can be chosen with respect to the particular instance. Newton's method is usually mentioned in this context (see Section 5 for recent applications of Newton's method). Renegar (1986) recently developed a polynomial-time algorithm related to this method of centers (see Section 5).

Betke & Gritzmann (1986) establish a formal relation between linear programming and unconstrained convex optimization as follows. They consider the linear programming problem in the feasibility form

$$Ax \leq b$$

$$x \geq 0,$$

where $A \in R^{m \times d}$. For any $x \in R^d$, the vector of "violations" at x is the following:

$$F(x) = (x_1^-, \ldots, x_d^-, (A_1 x - b_1)^+, \ldots, (A_m x - b_m)^+)^T$$

where $\xi^+ = \max\{\xi, 0\}$ and $\xi^- = \max\{-\xi, 0\}$. For any p $(p \geq 2)$, a convex function $g(x) = g_p(x)$ is then constructed in terms of the ℓ_p-norm of $F(x)$:

$$g(x) = (\|F(x)\|_p)^p.$$

General conditions on algorithms for minimizing convex functions are stated which imply polynomial time when applied to the function g for solving the linear programming problem. The proposed algorithms generalize the ellipsoid algorithm. They attempt to minimize $g(x)$ by taking gradient steps in a transformed space. I first explain the transformations involved and their relation to the ellipsoid method. Consider a matrix

$$M = I + \lambda uu^T$$

where $\boldsymbol{u}^T\boldsymbol{u} = 1$ and $\lambda > -1$. Obviously, the linear transformation associated with M is a dilatation of space by a factor of $1+\lambda$ in the direction of $\boldsymbol{u}$. An iterate of the ellipsoid method consists of an infeasible point $\boldsymbol{x}^k$ together with an ellipsoid E_k centered at $\boldsymbol{x}^k$ and guaranteed to contain all the basic feasible solutions if any. By a suitable affine transformation, the point $\boldsymbol{x}^k$ is mapped to the origin and the ellipsoid E_k is transformed into a ball. In the transformed space the algorithm then moves from the origin in a direction $\boldsymbol{u}$ perpendicular to one of the violated constraints, and a new ellipsoid is constructed. The new ellipsoid has one short axis in the direction $\boldsymbol{u}$ whereas all the other axes are of equal length. Thus, to transform this ellipsoid back into a ball, we first translate the new point to the origin and then apply a dilatation transformation of the form $I+\lambda\boldsymbol{u}\boldsymbol{u}^T$. This implies that the current ellipsoid in the original space of the problem can be represented as the image of a ball (centered at the origin) under a chain of "contractions" (that is, dilatations with negative λs) followed by a translation.

The proposed generalization of Betke & Gritzmann is to move in the transformed space (that is, where the current ellipsoid is represented by a ball) in the direction of the gradient of $g(\boldsymbol{x})$, construct a new ellipsoid, apply a dilatation that maps it into a ball, and so on. The original ellipsoid method is reproduced by choosing the norm to be ℓ_∞. The claim is that for any p this method yields a polynomial-time algorithm. On the more practical side, Betke & Gritzmann have applied the variable metric method (Davidon 1959; Fletcher & Powell 1963) with exact line searches for minimizing the function $g_2(\boldsymbol{x})$. The exact line searches are possible since $g_2(\boldsymbol{x})$ is piecewise quadratic. The reported computational experience is promising.

5. RECENT ALGORITHMS BASED ON NEWTON'S METHOD

Newton's method is a classical tool for minimizing convex functions. Essentially, to minimize a convex function $f(\boldsymbol{x})$ (with no constraints) one simply attempts to solve the equation $g(\boldsymbol{x}) \equiv \nabla f(\boldsymbol{x}) = 0$. Newton's iteration is based on first-order approximation of the function $g(\boldsymbol{x})$, so the next point is determined by the equation $g(\boldsymbol{x}^k)+\nabla g(\boldsymbol{x}^k)(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k) = 0$, which is equivalent to

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - (\nabla^2 f(\boldsymbol{x}^k))^{-1}\nabla f(\boldsymbol{x}^k).$$

Intuitively, Newton's method of optimization is more powerful than the gradient method since it takes into account information about the curvature. Traditional methods of constrained nonlinear optimization apply

Newton's method to unconstrained functions that incorporate the constraints of the given problem. The barrier function technique is one such way of incorporating constraints. Interestingly, when the current point tends to the boundary of the feasible domain, the gradient direction of the barrier function becomes, as desired, perpendicular to the boundary, whereas Newton's direction becomes parallel to it (see Megiddo & Shub 1986).

The news reports about Karmarkar's algorithm suggested that his ideas were revolutionary. Following these claims, a number of people wrote papers suggesting that algorithms based on traditional principles should yield similar results.

Iri & Imai (1986) use a barrier function

$$\phi(\boldsymbol{x}) = \frac{(c^T\boldsymbol{x})^{m+1}}{\prod_{i=1}^{m}(A_i\boldsymbol{x}-b_i)}$$

closely related to Karmarkar's potential function, for solving the problem

Minimize $c^T\boldsymbol{x}$

subject to $A\boldsymbol{x} \geq b$.

A nice property of Iri & Imai's function and Karmarkar's potential function is that both are free of parameters (such as μ in the traditional barrier method) that need to be updated by the algorithm. Assuming the optimal value is zero, Iri & Imai simply minimize the function ϕ using Newton's search direction method. This approach is justified by the proven convexity of the barrier function, since Newton's method has been known to work well on such functions. The interesting feature in their work is that they analyze the effect of performing line searches, rather than just taking standard steps in the chosen direction. Although it is not clear that their complete algorithm converges quadratically, the sequence of relative minima (i.e. minima with respect to the search line) converges quadratically to the optimum. Also, it is not known whether this algorithm runs in polynomial time.

De Ghellinck & Vial (1985, 1986) describe another algorithm that can be interpreted as a Newton method. They set the problem in a homogeneous feasibility form:

$$A\boldsymbol{x} = 0$$

$$\boldsymbol{x} \geq 0$$

$$x_0 \neq 0$$

(where $A \in R^{m\times(n+1)}$ and the columns are indexed $0, 1, \ldots, n$). The algo-

rithm works inside the positive orthant of R^{n+1}, attempting to find a feasible point by driving the following quantities to zero:

$$\psi_i(\boldsymbol{x}) = \frac{A_i \boldsymbol{x}}{\boldsymbol{e}^T \boldsymbol{x}} \quad (i = 1, \ldots, m).$$

Given any point $\boldsymbol{x} \in R_+^{n+1}$ $(\boldsymbol{x} > 0)$, one would ideally like to find a translation $\boldsymbol{u} \in R^{n+1}$ defined by the following optimization problem:

$$\text{Maximize } \boldsymbol{e}^T(\boldsymbol{x}+\boldsymbol{u})$$

$$\text{subject to } A\boldsymbol{u} = 0$$

$$\boldsymbol{x}+\boldsymbol{u} \geq 0$$

where the optimization is with respect to $\boldsymbol{u}$. The difficulty of this problem is in the nonnegativity constraints. Since $\boldsymbol{x} > 0$, one can avoid violating the nonnegativity constraints by considering the following problem instead:

$$\text{Maximize } \prod_{j=0}^{n} (x_j+u_j)$$

$$\text{subject to } A\boldsymbol{u} = 0.$$

Working in the interior of the positive orthant, the objective function can be replaced by

$$\phi(\boldsymbol{u}) = \phi_x(\boldsymbol{u}) = \sum_j \ln (x_j+u_j),$$

which is concave. Newton's method is a natural choice as a tool for optimizing the latter. Obviously,

$$\nabla\phi(0) = D_x^{-1}\boldsymbol{e}$$

and

$$\nabla^2\phi(0) = -D_x^{-2},$$

where $D_x = \text{Diag}(x)$. Thus, the Newton direction is determined by the following problem

$$\text{Maximize } -\frac{1}{2}\boldsymbol{u}^T D_x^{-2}\boldsymbol{u} + \boldsymbol{e}^T D_x^{-1}\boldsymbol{u}$$

$$\text{subject to } A\boldsymbol{u} = 0$$

or, equivalently,

$$\text{Minimize} \sum_j \left(\frac{u_j}{x_j} - 1\right)^2$$

subject to $Au = 0$.

The vector u is therefore determined together with a vector of multipliers v by the following system:

$$\begin{aligned} D_x^{-2}u + A^T v &= D_x^{-1}e \\ Au &= 0. \end{aligned}$$

The solution is

$$u = D_x[I - D_x A^T (AD_x^2 A^T)^{-1} AD_x]e.$$

This can be interpreted as follows. Standing at a point $x > 0$, we would like to move *parallel* to the nullspace of A so that $e^T x$ increases. The direction of movement is determined by first transforming the space. Let us transform the space by the linear rescaling transformation D_x^{-1}. The point x is mapped to the point e and the nullspace of A is transformed into the nullspace of AD_x. In the transformed space we move parallel to the nullspace of AD_x. We first project e orthogonally into the latter, denoting this projection by e_p. The movement in the transformed space is from e to a point of the form $e + \alpha e_p$, where α is a positive scalar. Of course, the direction of movement in the original space is that of the vector $D_x e_p$. Although the de Ghellinck-Vial algorithm is described as an exterior-point method, it does involve arguments similar to those used in Karmarkar's algorithm, and its framework is suggested by the projective nature of the latter. They also prove a polynomial-time bound.

Smale (1986) proposes to use Newton's method for solving linear programming problems in a related way. It is well known that the problem

$$\text{Maximize } c^T x$$

$$\text{subject to } Ax \leq b$$

$$x \geq 0$$

can be set as a linear complementarity problem, namely, finding z such that

$$\begin{aligned} Mz + q &\geq 0 \\ z &\geq 0 \\ z^T(Mz + q) &= 0 \end{aligned}$$

where

$$M = \begin{pmatrix} O & -A^T \\ A & O \end{pmatrix}$$

and $q = (-c, b)^T$. Equivalently, if we identify z with y^+ and $-(Mz+q)$ with y^- (denoting $\eta^+ = \max\{\eta, 0\}$, $\eta^- = \min\{\eta, 0\}$, and $y^{\pm} = (y_1^{\pm}, \ldots, y_{m+n}^{\pm})^t$) then the problem is equivalent to finding y such that $My^+ + y^- = -q$. As functions of η, the quantities η^+ and η^- are piecewise linear with a single breakpoint at the origin. They can be approximated, respectively, by hyperbolas:

$$\varphi_\mu^+(\eta) = \frac{\eta + \sqrt{\eta^2 + \mu^2}}{2}$$

and

$$\varphi_\mu^-(\eta) = \frac{\eta - \sqrt{\eta^2 + \mu^2}}{2}$$

where $\mu > 0$ is a parameter. These hyperbolas converge uniformly to the corresponding piecewise linear functions as μ tends to zero. The approximate problem with a fixed value of μ is to solve the equation

$$\Phi_\mu(y) \equiv M\Phi_\mu^+(y) + \Phi_\mu^-(y) = -q$$

where

$$\Phi_\mu^{\pm}(y) = (\varphi_\mu^{\pm}(y_1), \ldots, \varphi_\mu^{\pm}(y_{m+n}))^T.$$

Given q one can apply Newton's method for solving this equation by following the inverse image under Φ_μ of a line segment $[q^0, q]$ (starting at any point $q^0 > 0$). Lemke's algorithm (which is a generalization of Dantzig's self-dual simplex method) follows the inverse image of such a line segment under

$$\Phi_0(y) \equiv My^+ + y^-.$$

Smale argues that the Newton path converges to Lemke's path as μ tends to zero.

If one prefers Newton's method to simplex-type methods then one may apply the former with a fixed small value of μ, but it is doubtful that this idea will give rise to a more efficient algorithm. Another approach, more in the spirit of traditional methods of nonlinear optimization, is to drive μ to zero during the execution of the algorithm. This is in a sense an attempt to follow the path determined by $\Phi_\mu^{-1}(-q)$ as μ varies. Consider the mapping $\Phi_\mu(y)$. The Jacobian matrix is equal to $MD_y^+ + D_y^-$ where D_y^+ and D_y^- are the (diagonal) Jacobian matrices of $\Phi_\mu^+(y)$ and $\Phi_\mu^-(y)$,

respectively. The Newton direction $\boldsymbol{u}$ at $\boldsymbol{y}$ (relative to a fixed value of μ) is obtained as the solution of the following system of linear equations:

$$(MD_y^+ + D_y^-)\boldsymbol{u} = -\boldsymbol{q} - \Phi_\mu(\boldsymbol{y}).$$

The structure of the underlying matrix is very similar to the ones encountered in the algorithms described previously in this paper. It is worthwhile to note that the approximation by the hyperbolas can be interpreted as a penalty function method. Suppose the problem is to solve a system of linear inequalities $A\boldsymbol{x} \geq \boldsymbol{b}$ ($A \in R^{m \times n}$). Then an approximation to it is to minimize the function

$$\psi_\mu(\boldsymbol{x}) = \sum_{i=1}^{m} \varphi_\mu^+(A_i\boldsymbol{x} - b_i).$$

In the remainder of this section I consider a recent algorithm proposed by Renegar (1986). He applies Newton steps in an algorithm that is also related to the logarithmic barrier technique of Frisch (1955) and to Huard's (1967) method of centers (see Section 4). Renegar states his algorithm with respect to the problem in the inequality form:

Maximize $c^T\boldsymbol{x}$

subject to $A\boldsymbol{x} \geq \boldsymbol{b}$,

where $A \in R^{m \times n}$. The following function (where τ is a parameter) plays an important role in the development of the algorithm:

$$f(\boldsymbol{x}) = f(\boldsymbol{x}; \tau) = m \ln(c^T\boldsymbol{x} - \tau) + \sum_{i=1}^{m} \ln(A_i\boldsymbol{x} - b_i).$$

(Renegar uses a coefficient ℓ instead of m and argues that the choice $\ell = m$ is optimal in a certain sense.) The value of τ is changed throughout the execution of the algorithm.

Suppose the problem is formulated so that it has a feasible domain of full dimension and an optimal solution. The function $f(\boldsymbol{x}; \tau)$ is well-defined for $\boldsymbol{x}$ in

$$P_\tau = \{\boldsymbol{x} : c^T\boldsymbol{x} > \tau, A\boldsymbol{x} > \boldsymbol{b}\}.$$

Since $f(\boldsymbol{x}, \tau)$ is strictly concave and P_τ is bounded, it follows that there exists a unique maximizer $\boldsymbol{x}^* = \boldsymbol{x}^*(\tau)$ of $f(\boldsymbol{x}; \tau)$. This point is characterized as the solution of the following set of equations (equivalent to $\nabla f(\boldsymbol{x}; \tau) = 0$):

$$\frac{m}{c^T\boldsymbol{x} - \tau}\boldsymbol{c} + A^T\tilde{D}^{-1}(\boldsymbol{x})\boldsymbol{e} = 0,$$

where

$$\tilde{D}(\boldsymbol{x}) = \text{Diag}(A_1\boldsymbol{x} - b_1, \ldots, A_m\boldsymbol{x} - b_m)$$

and

$$\boldsymbol{e} = (1, \ldots, 1)^T \in R^m.$$

When τ varies we obtain a path through the interior of the feasible domain. For any $\boldsymbol{y}$ such that $A\boldsymbol{y} > \boldsymbol{b}$, let

$$L(\boldsymbol{y}) = \{\boldsymbol{x} : \boldsymbol{c}^T\boldsymbol{x} = \boldsymbol{c}^T\boldsymbol{y}, A\boldsymbol{x} > \boldsymbol{b}\}.$$

Obviously, $\boldsymbol{x}^*(\tau)$ is the maximizer of $f(\boldsymbol{x}; \tau)$ in the set $L(\boldsymbol{x}^*(\tau))$. Thus, $\boldsymbol{x}^*(\tau)$ is also the maximizer of $\Sigma_{i=1}^m \ln(A_i\boldsymbol{x} - b_i)$ over $L(\boldsymbol{x}^*(\tau))$. Consider, on the other hand, the function

$$g(\boldsymbol{x}) = g(\boldsymbol{x}; \mu) = \boldsymbol{c}^T\boldsymbol{x} + \mu \sum_{i=1}^{m} \ln(A_i\boldsymbol{x} - b_i),$$

used in the barrier function technique. Here too we can define a path $\boldsymbol{x} = \boldsymbol{x}^+(\mu)$, where $\boldsymbol{x}^+(\mu)$ is the maximizer of $g(\boldsymbol{x}; \mu)$ over the interior of the feasible domain. Since $\boldsymbol{x}^+(\mu)$ is obviously the maximizer of $g(\boldsymbol{x}; \mu)$ over the set $L(\boldsymbol{x}^+(\mu))$, we have that $\boldsymbol{x}^+(\mu)$ is also the maximizer of $\Sigma_{i=1}^m \ln(A_i\boldsymbol{x} - b_i)$ over $L(\boldsymbol{x}^+(\mu))$. We thus have

Proposition 5.1 *The functions* $\mathbf{x} = \mathbf{x}^*(\tau)$ *and* $\mathbf{x} = \mathbf{x}^+(\mu)$ *are just two parameterizations of the same path.*

Let us refer to the path discussed above as the *central path*. The central path converges to an optimal solution. This path has been studied in the context of nonlinear optimization by Fiacco & McCormick (1967). More recently, in the context of linear programming, the path was studied by Bayer & Lagarias (1986) and Megiddo (1986b). The barrier function attempts to "follow" this path. However, the word "follow" may be interpreted in various ways in this context. In general, a "path-following technique" would run as follows. Given a value of μ, assuming we are at a point that is not so far from the point $\boldsymbol{x}^+(\mu)$, we run a certain number of Newton iterations with this fixed value of μ, to get closer to the point $\boldsymbol{x}^+(\mu)$. When we get sufficiently close to the point $\boldsymbol{x}^+(\mu)$, we update the value of the parameter μ by some rule, and then the process is repeated.

Let us examine this procedure in a more general context. Let $F(\boldsymbol{x}; t)$ be a mapping R^{n+1} into R^n ($\boldsymbol{x} \in R^n$, $t \in R$) that implicitly defines a path $\boldsymbol{x} = \boldsymbol{x}(t)$ through the equation

$$F(\boldsymbol{x}; t) = 0.$$

Suppose we are at a point $\boldsymbol{x}^0$ and we attempt to follow this path. Let t_0 be the current value of t, so that $\boldsymbol{x}^0$ is considered sufficiently close or even

equal to $x(t_0)$. We now update the value of t to be $t_0+\Delta t$. The Newton step is computed with respect to the equation $F(x; t_0+\Delta t) = 0$ (with the current point x^0). The step Δx is given by

$$\Delta x = -\left(\frac{\partial F(x^0; t_0+\Delta t)}{\partial x}\right)^{-1} F(x^0; t_0+\Delta t).$$

Suppose, for a moment, that x^0 is a point on the path—that is, $x^0 = x(t_0)$ and $F(x^0; t_0) = 0$. We later consider the tangent to the path $x = x(t)$ at x^0. This tangent is the line

$$\{x^0+\lambda u : \lambda \in R\}$$

where

$$u = \frac{dx(t_0)}{dt} = -\left(\frac{\partial F(x^0; t_0)}{\partial x}\right)^{-1} \frac{\partial F(x^0; t_0)}{\partial t}.$$

Consider the representation of the central path with $x = x^+(\mu)$. The implicit function is

$$F(x; \mu) = \nabla g(x; \mu) = c+\mu A^T \tilde{D}^{-1}(x)e.$$

In this case we have

$$\frac{\partial F}{\partial \mu}(x, \mu) = A^T \tilde{D}^{-1}(x)e.$$

Consider a point x^0 on the path, $x^0 = x^+(\mu_0)$. By definition,

$$\nabla g(x^0; \mu_0) = 0.$$

Now, let $\mu' = \mu-\Delta\mu$ be the updated value of the parameter. We have

$$\nabla g(x^0; \mu_0-\Delta\mu) = c+(\mu_0-\Delta\mu)A^T \tilde{D}^{-1}(x^0)e = -\Delta\mu A^T \tilde{D}^{-1}(x^0)e.$$

We thus have

Proposition 5.2. *If* $\mathbf{x}^0 = \mathbf{x}^+(\mu_0)$ *then for any update* $\Delta\mu$ *of the parameter* μ, *the Newton step with respect to the logarithmic barrier parameterization* $\mathbf{x}^+(\mu)$ *is in the direction of the tangent of the path.*

Proof: Since

$$\frac{\partial F(x; \mu)}{\partial x} = -\mu A^T \tilde{D}^{-2}(x)A$$

and

$$\frac{\partial F}{\partial \mu}(x, \mu) = A^T \tilde{D}^{-1}(x)e,$$

it follows that the common direction is the direction of the vector

$$-(A^T \tilde{D}^{-2}(x^0)A)^{-1} A^T \tilde{D}^{-1}(x^0)e.$$

The situation with Renegar's algorithm is the same:

Proposition 5.3. *If* $\mathbf{x}^0 = \mathbf{x}^*(\tau_0)$ *then for any update* $\Delta\tau$ *of the parameter* τ, *the Newton step with respect to Renegar's parameterization* $x^*(\tau)$ *is in the direction of the tangent of the path.*

Proof: Here the implicit function is

$$F(x; \tau) = \nabla f(x; \tau) = \frac{m}{c^T x - \tau} c + A^T \tilde{D}^{-1}(x)e.$$

In this case we have

$$\frac{\partial F}{\partial \tau}(x, \tau) = \frac{m}{(c^T x - \tau)^2} c.$$

Consider a point x^0 on the path, $x^0 = x^*(\tau_0)$. By definition,

$$\nabla f(x^0; \tau_0) = 0.$$

Let $\tau' = \tau + \Delta\tau$ be the updated value of τ. Thus,

$$\begin{aligned} F(x^0; \tau_0 + \Delta\tau) &= \frac{m}{c^T x^0 - \tau^0 \Delta\tau} c + A^T \tilde{D}^{-1}(x^0)e \\ &= m\left(\frac{1}{c^T x^0 - \tau^0 \Delta\tau} - \frac{1}{c^T x^0 - \tau^0}\right)c \\ &\quad + \frac{m}{c^T x^0 - \tau^0} c + A^T \tilde{D}^{-1}(x^0)e \\ &= m\left(\frac{1}{c^T x^0 - \tau^0 - \Delta\tau} - \frac{1}{c^T x^0 - \tau^0}\right)c. \end{aligned}$$

On the other hand,

$$\frac{\partial F(x; \tau)}{\partial x} = -\frac{m}{(c^T x - \tau)^2} cc^T - A^T \tilde{D}^{-2}(x)A.$$

Notice that, in general, for any matrix $M \in R^n$ and scalars α, β, the solution v of the system

$$(M+\alpha cc^T)v = \beta c$$

satisfies

$$v = M^{-1}(\beta - \alpha c^T v)c$$

and hence v and $M^{-1}c$ lie on the same line. This implies that in our case, if x^0 is on the central path, then Renegar's step is in the direction of the tangent—that is, the direction of the vector

$$(A^T\tilde{D}^{-2}(x)A)^{-1}c.$$

Notice that this is the same direction as Proposition 5.2 since

$$-A^T\tilde{D}^{-1}(x^0)e = c$$

when x^0 is on the central path.

We have seen that at points on the central path both algorithms assign the same direction of movement—namely, the direction tangent to the path. Let us now consider points x off the central path. Let $\mu = \mu(x)$ be the value of the parameter associated with the point x (under a certain variant of the logarithmic barrier technique). The direction assigned by the barrier technique is given by a vector u satisfying

$$A^T\tilde{D}^{-2}(x)Au = (c+\mu A^T\tilde{D}^{-1}(x)e).$$

On the other hand, Renegar's direction is given by a vector v satisfying

$$\left(\frac{m}{(c^Tx-\tau)^2}cc^T + A^T\tilde{D}^{-2}(x)A\right)v = \frac{m}{c^Tx-\tau}c + A^T\tilde{D}^{-1}(x)e.$$

In order for $u = u(x)$ and $v = v(x)$ to have the same direction it suffices that

$$\frac{m}{(c^Tx-\tau)^2}cc^Tu = \left(\mu\frac{m}{c^Tx-\tau} - 1\right)c.$$

The latter is equivalent to

$$\mu = \mu(x) = \frac{c^Tu}{c^Tx-\tau} + \frac{c^Tx-\tau}{m}.$$

We thus have

Proposition 5.4. *The direction assigned by Renegar's algorithm at a point* x, *when the updated value of the parameter is* τ, *is the same as the one assigned by the logarithmic barrier technique where the parameter* μ *is given by*

$$\mu = \mu(\boldsymbol{x}) = \frac{\boldsymbol{c}^T\boldsymbol{u}}{\boldsymbol{c}^T\boldsymbol{x}-\tau} + \frac{\boldsymbol{c}^T\boldsymbol{x}-\tau}{m},$$

where

$$\boldsymbol{u} = (A^T\tilde{D}^{-2}(\boldsymbol{x})A)^{-1}(\boldsymbol{c}+\mu A^T\tilde{D}^{-1}(\boldsymbol{x})\boldsymbol{e}).$$

Interestingly, Gill et al (1985) showed that a certain choice of $\mu(\boldsymbol{x})$ in the projected Newton logarithmic barrier algorithm (stated for problems in standard form) assigns the same direction as Karmarkar's algorithm.

The actual algorithm proposed by Renegar is as follows. Two parameters must be fixed. First, there is constant δ whose value is chosen as

$$\delta = \frac{1}{13\sqrt{m}}$$

for obtaining a valid algorithm. Second, one must fix the number N of Newton iterations to be performed per step of the algorithm. The main result is proven with $N = 1$. The algorithm generates a sequence of interior points $\{\boldsymbol{x}^k\}$ and a sequence of lower bounds $\{\tau^k\}$ on the optimal value of the objective function. The algorithm starts with an interior point $\boldsymbol{x}^0$ and a lower bound $\tau^0 < \boldsymbol{c}^T\boldsymbol{x}^0$. The iterative step is as follows. Given $\boldsymbol{x}^k$ and τ^k ($\tau^k < \boldsymbol{c}^T\boldsymbol{x}^{k+1}$), perform N Newton iterations starting at $\boldsymbol{x}^k$, to maximize the function $f(\boldsymbol{x}) = f(\boldsymbol{x}; \tau^k)$. Note that here a Newton iteration does not involve a line search. The sequence of points generated during a step,

$$\boldsymbol{x}^k = \boldsymbol{y}^0, \ldots, \boldsymbol{y}^N = \boldsymbol{x}^{k+1}$$

is determined by the equation:

$$\nabla^2 f(\boldsymbol{y}^j; \tau^k)(\boldsymbol{y}^{j+1}-\boldsymbol{y}^j) = -\nabla f(\boldsymbol{y}^j; \tau^k) \qquad (j = 0, \ldots, N).$$

However, to make sure that these points stay in the feasible domain, one has to choose $N = 1$ and δ as above. The constant δ determines the new value of τ:

$$\tau^{k+1} = \delta\boldsymbol{c}^T\boldsymbol{x}^{k+1} + (1-\delta)\tau^k$$

and it is chosen sufficiently small that the first Newton step at $\boldsymbol{x}^{k+1}$ does not generate an exterior point.

Suppose $\boldsymbol{x}^*$ is an optimal solution. Renegar proves that objective function values $\boldsymbol{c}^T\boldsymbol{x}^k$ converge to $\boldsymbol{c}^T\boldsymbol{x}^*$ at least as fast as implied by the following inequality:

$$\boldsymbol{c}^T\boldsymbol{x}^* - \boldsymbol{c}^T\boldsymbol{x}^k \leq \left(1 - \frac{1}{28\sqrt{m}}\right)^k (\boldsymbol{c}^T\boldsymbol{x}^* - \tau^0).$$

This means that the number of steps it takes the algorithm to reach a point where the termination criterion holds is $O(\sqrt{m+n}L)$ whereas for Karmarkar's algorithm the known bound on the number of steps is $O((m+n)L)$.

6. PRACTICAL IMPROVEMENTS

The emphasis in Karmarkar's paper (1984) is on the asymptotic worst-case complexity, and therefore some of the assumptions he makes (for the ease of presentation) sound unreasonable from a practitioner's point of view. For example, the hypothesis that the optimal objective function value is zero is justified by setting the problem in the combined primal-dual form. Other people (Todd & Burrell 1986; Anstreicher 1986a,b,c; Gay 1985; Gonzaga 1985; Goldfarb & Mehrotra 1986) later proposed various algorithms of the same type where this hypothesis can be relaxed. Although these improvements do not improve the order of magnitude of the known upper bound, they are believed to have a significant practical value. The paper by Todd & Burrell was also the first to show how dual variables arise naturally in this algorithm.

Another direction, pursued by Goldfarb & Mehrotra (1985, 1986) and Kojima & Tone (1986), is to make the algorithm more efficient by computing projections only approximately. Polynomial time bounds still hold for these relaxed versions. It seems that such an approach would be even more advantageous if the problem is set in the inequality form. Notice that in standard form, although the direction does not have to be the exact projection, it still has to be in the appropriate nullspace or else the new point will not be feasible. In the inequality form, starting from an interior point, one always remains in the feasible domain provided the step size is not too large.

Karmarkar did not provide a practical stopping rule for his algorithm. In practice one must stop the algorithm at a certain point (earlier than the theoretical one) and attempt to guess what the optimal solution is. Kojima (1986) developed such rules for deducing at a relatively early stage which variables must be basic in the optimal solution.

The standard form of a linear programming problem distinguishes the nonnegativity constraints from the equality constraints. Thus, upper bound constraints (that is, inequalities of the form $x_i \leq u_i$) that may exist in the original problem are "lost" during the transformation to standard form. It is known that the simplex method can be stated with respect to problems in a more general form where $0 \leq x_i \leq u_i$ (and u_i may be infinite) so as to take advantage of such constraints. In fact, commercial codes of the simplex method are set to deal with problems in this form. The question

of how to deal with upper bounds in the context of Karmarkar's algorithm is very natural. Rinaldi (1986) developed a theory for dealing with such upper bounds in this context.

7. CONCLUSION

Encouraging experiences with different methods were recently reported also by Kojima & Tone (1986) and Kojima (1986). From a theoretician's point of view it is interesting to confirm the surprisingly low number of iterations that different nonlinear methods perform on linear programming problems. The upper bounds proven for various algorithms are not known to be tight. The analysis so far is typically based on estimating the worst that can happen during a *single* step of the algorithm. It is conceivable that one will be able to prove that such bad performance cannot occur during many steps. In a certain sense Karmarkar's potential function captures such a phenomenon. More precisely, he proves that if the improvement in terms of the *objective* function is small during a certain step then progress is made during that step in another sense—namely, the barrier part improves since the potential function does. However, it is not known whether a small improvement in terms of the *potential* function during one step implies larger improvements during other steps.

A theory is desired that would explain the differences among various algorithms and identify features that probably make one algorithm better than another one.

An interesting feature of the new algorithms is that they are invariant under certain transformations of the problem or, in other words, they are "unit free." Some variants of the simplex method are not unit free, but it is easy to design ones that are. It is interesting to pursue the connection between invariance and efficiency of an algorithm. Let us first examine this question in a rather abstract setting. More concrete examples will be given later.

By a *problem* we mean a set P of pairs (I,s) where I is an *instance* or an *input* and $s \in S(I)$ is one of its *solutions*. Let us further assume that there is some fixed equivalence relation $\sim$ over instances such that $I_1 \sim I_2$ only if $S(I_1) = S(I_2)$. Now, let us fix the model of computation so that we have a well-defined notion of an *algorithm*. Roughly, an algorithm receives an input, operates on it under the model of computation, and returns an output. By an *algorithm for* P we mean an algorithm that receives an input I and returns an output $s \in S(I)$, or else recognizes that there is no s such that $(I, s) \in P$. A *computation* is a sequence that describes the operation of an algorithm on a particular input. Thus a computation C can be specified by an algorithm A together with an input I where $C = C(A, I)$. Let $\cong$

denote an equivalence relation over computations. An algorithm A is invariant with respect to $\sim$ and $\cong$ if for every $I_1 \sim I_2$, $C(A, I_1) \cong C(A, I_2)$—that is, its computations on equivalent problems are equivalent. The invariance of an algorithm depends on the definition of equivalence of instances and computations. Here we are of course interested in the linear programming problem. Consider, as an example, the role of the center of gravity of a polytope. We do not know of useful ways of computing the center of gravity of a polytope, so we do not expect to produce good algorithms for linear programming that rely on this point. However, we can still examine the issue from the point of view of invariance and complexity. The center of gravity is *not* invariant under convex projective transformations—that is, the image of the center of gravity is not necessarily equal to the center of gravity of the image of the polytope. However, despite its not being invariant, the center of gravity *is* a very useful point to know. The "only" problem is that we do not know how to compute it efficiently. Suppose we had an oracle that could tell us the center of any polytope (given by a set of linear inequalities). We could then design a linear programming algorithm as follows. Given a (bounded) polytope, produce a second polytope by cutting the first one through its center of gravity with a hyperplane on which the objective function is constant, picking the half that contains the optimum. This step gives rise to a very good iterative algorithm since the volume of the remaining polytope is reduced by a factor of at least $1/e$ in each step. Now, consider the rationale of the invariance postulate. The representation of an instance may be chosen by an adversary. Thus, if an algorithm is not invariant, the adversary will choose a bad representation for the algorithm. On the other hand, if the algorithm is invariant (and the definition of equivalence of computations is right) then the choice of the representation has no effect and the adversary is neutralized. Given the real world instance that we need to solve, the adversary may choose a representation in which the center of gravity corresponds to an interior point close to the "anti-optimum" so that a cut through that point will not reduce the volume significantly. However, this argument holds only for the first iteration. Once the representation has been fixed, the repeated choice of the center of gravity guarantees steady progress towards the optimum, at least in the metric of the representation.

A simplification of the preceding paragraph can be made by considering the more familiar binary search. Suppose $f(x)$ is continuous over the unit interval such that $f(0) < 0 < f(1)$ and we need to approximate a point x such that $f(x) = 0$. The only information we can obtain is in the form of values of f at points of our choice. A natural equivalence relation over inputs in this problem is derived from monotone transformations: f and

g are equivalent if there is a strictly monotone function m such that $g(x) = m(f(x))$. We may further restrict the equivalence relation to cases where m is a projective transformation of the form

$$m(y) = \frac{(a+1)y}{ay+1},$$

which implies that $m(0) = 0$ and $m(1) = 1$. Even under the weaker relation, binary search is not invariant. Indeed, with binary search we believe we are chopping the interval by the maximum amount possible in the worst case, in each step. An adversary can distort the picture in such a way that our binary search probes, at least in the first step, very close to one of the endpoints. But, since the adversary cannot change the metric from step to step, we do make progress relative to the metric he had to choose before the process started.

Invariance does look like a nice theoretical requirement. However, the level of equivalence of inputs should be decided with caution. For example, in the context of graphs it seems natural to consider isomorphic graphs as equivalent from the point of view of computing graph-theoretic invariants. Taking such a point of view eliminates any interest in the graph isomorphism problem. Also, in the context of recognizing Hamiltonian graphs, suppose an adversary could choose the particular assignment of labels to nodes. Knowing that our algorithm is invariant, the adversary does not compromise anything by assigning the labels $1, 2, \ldots$, around a Hamiltonian circuit if one exists. Thus an invariant algorithm cannot take advantage of such transparent facts.

A somewhat less trivial observation applies to the linear programming problem. Instead of invariance under convex projective transformations let us consider a weaker notion—namely, invariance under linear transformations. It may seem natural to consider two linear programming problems equivalent if there is a linear transformation that maps one to the other. However, for every linear programming problem there is a linear transformation that *reveals the solution.* In fact, the simplex algorithms compute such transformations and use them as certifications for the solution.

A weaker notion of equivalence that seems more appropriate is derived from scaling: Two problems are equivalent if they differ only in the choice of units of measurement. Thus, an invariant algorithm is one that does not depend on these choices. Indeed, many variants of the simplex algorithm are invariant under scaling transformations.

Interior point algorithms can also be studied through the vector fields they define over the feasible domain. Such an approach is used by Bayer &

Lagarias (1986), Megiddo & Shub (1986), and Megiddo (1986b). Nazareth (1986) points out the relation to homotopy methods previously used in nonlinear optimization.

The new methods would probably extend to problems close to linear programming, such as convex quadratic programming. An extension of Karmarkar's algorithm to this domain has already been developed by Kapoor & Vaidya (1986). Megiddo (1986b) discusses extensions to various types of linear complementarity problems.

Literature Cited

Adler, I., Karp, R. M., Shamir, R. 1983a. *A family of simplex variants solving an m × d linear program in expected number of pivots depending on d only*. Rep. UCB CSD 83/157, Comp. Sci. Div., Univ. Calif., Berkeley

Adler, I., Karp, R. M., Shamir, R. 1983b. *A family of simplex variants solving an m × d linear program in $O(\min(m^2, d^2))$ expected number of pivot steps*. Rep. UCB CSD 83/158, Comp. Sci. Div., Univ. Calif., Berkeley

Adler, I., Megiddo, N. 1985. A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension. *J. ACM* 32: 871–95

Adler, I., Resende, M. G. C., Veiga, G. 1986. *An implementation of Karmarkar's algorithm for linear programming*. Rep. ORC 86-8, Oper. Res. Ctr., Univ. Calif., Berkeley

Anstreicher, K. M. 1986a. *Analysis of a modified Karmarkar algorithm for linear programming*. See Megiddo 1986c

Anstreicher, K. M. 1986b. *A strengthened acceptance criterion for approximate projections in Karmarkar's algorithm*. Tech. Rep., Yale Sch. Organization and Management, Yale Univ.

Barnes, E. R. 1985. *A variation on Karmarkar's algorithm for solving linear programming problems*. Res. Rep. RC 11136, IBM T. J. Watson Res. Ctr., Yorktown Heights, NY

Bayer, D. A., Lagarias, J. C. 1986. *The nonlinear geometry of linear programming I: affine and projective rescaling trajectories*. Murray Hill, NJ: AT&T

Betke, U., Gritzmann, P. 1986. *Projection algorithms for linear programming*. Presented at the 1986 AMS-IMS-SIAM Summer Res. Conf. on Discrete and Computational Geometry, Santa Cruz, Calif.

Blair, C. E. 1986. The iterative step in the linear programming algorithm of N. Karmarkar. See Megiddo 1986d

Borgwardt, K.-H. 1982a. Some distribution-independent results about the asymptotic order of the average number of pivot steps of the simplex method. *Math. Oper. Res.* 7: 441–62

Borgwardt, K.-H. 1982b. The average number of steps required by the simplex method is polynomial. *Z. Oper. Res.* 26: 157–77

Chen, S. 1986. *Computational experience with the Karmarkar algorithm*. Presented at TIMS/ORSA Meet., Los Angeles

Dantzig, G. B. 1963. *Linear Programming and Extensions*. Princeton, NJ: Princeton Univ. Press

Davidon, W. C. 1959. *Variable metric method for minimization*, Rep. ANL-5990, Argonne Natl. Lab.

de Ghellinck, G., Vial, J.-Ph. 1985. *An extension of Karmarkar's algorithm for solving a system of linear homogenous equations on the simplex*. Disc. Pap. 8538, C.O.R.E., Catholic Univ. Louvain, Belgium

de Ghellinck, G., Vial, J.-Ph. 1986. A polynomial Newton method for linear programming. See Megiddo 1986d

Fiacco, A. V., McCormick, G. P. 1967. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. New York/Toronto: Wiley

Fletcher, R., Powell, M. J. D. 1963. A rapidly convergent descent method for minimization. *Comput. J.* 6: 163–68

Frank, A., Tardos, E. 1985. An application of simultaneous approximation in combinatorial optimization. *Proc. IEEE Symp. Found. Comput. Sci., 26th, Portland*, pp. 459–63

Frisch, K. R. 1955. *The logarithmic potential method of convex programming*. Unpublished manuscript, Univ. Inst. Econ., Oslo, Norway

Gay, D. M. 1985. *A variant of Karmarkar's linear programming algorithm for problems in standard form.* Numer. Anal. Manusc. 85-10. Murray Hill, NJ: AT&T

Gill, P. E., Murray, W., Saunders, M. A., Tomlin, J. A., Wright, M. H. 1985. *On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method.* Tech. Rep. SOL 85-11, Dept. Operations Res., Stanford Univ.

Goldfarb, D., Mehrotra, S. 1985. *A relaxed version of Karmarkar's method.* Tech. Rep., Dept. Indust. Eng. Operations Res., Columbia Univ.

Goldfarb, D., Mehrotra, S. 1986. *Relaxed variants of Karmarkar's algorithm for linear programs with unknown objective value.* Tech. Rep., Dept. Indust. Eng. Operations Res., Columbia Univ.

Gonzaga, C. 1985. *A conical projection algorithm for linear programming.* Tech. Rep., Dept. Elect. Eng. Comput. Sci., Univ. Calif., Berkeley

Huard, P. 1967. Resolution of mathematical programming with nonlinear constraints by the method of centers. In *Nonlinear Programming*, ed. J. Abadie, pp. 207–19. Amsterdam: North-Holland

Iri, M., Imai, H. 1986. A multiplicative barrier function method for linear programming. See Megiddo 1986d

Kapoor, S., Vaidya, P. M. 1986. Fast algorithms for convex quadratic programming and multicommodity flows. *Proc. ACM Symp. Theory Comput., 18th, Berkeley*, pp. 147–59

Karmarkar, N. 1984. A new polynomial-time algorithm for linear programming. *Combinatorica* 4: 373–95

Karmarkar, N. K., Sinha, L. P. 1985. *Application of Karmarkar's algorithm to overseas telecommunications facilities planning.* Presented at 12th Symp. Math. Program., Cambridge, Mass.

Khachiyan, L. G. 1979. A polynomial algorithm in linear programming. *Sov. Math. Dokl.* 20: 191–94

Kojima, M. 1986. Determining basic variables of optimum solutions in Karmarkar's new LP algorithm. See Megiddo 1986d

Kojima, M., Tone, K. 1986. *An efficient implementation of Karmarkar's new LP algorithm.* Res. Rep. B-180, Dept. Info. Sci., Tokyo Inst. Tech.

Megiddo, N. 1986a. On the complexity of linear programming. In *Advances in Economic Theory*, ed. T. Bewley. New York: Cambridge Univ. Press

Megiddo, N. 1986b. *Pathways to the optimal set in linear programming.* IBM Res. Rep. RJ 5295

Megiddo, N. 1986c. Improved asymptotic analysis of the average number of steps performed by the self-dual simplex algorithm. *Math. Program.* 35: 140–72

Megiddo, N., ed. 1986d. Special issue: New Approaches to Linear Programming. *Algorithmica* 1: 4

Megiddo, N., Shub, M. 1986. *Boundary behavior of interior point algorithms for linear programming.* IBM Res. Rep. RJ 5319

Nazareth, J. L. 1986. Homotopies in linear programming. See Megiddo 1986d

Padberg, M. W. 1985a. *A different convergence proof of the projective method for linear programming.* Tech. Rep., New York Univ.

Padberg, M. W. 1985b. *Solution of a nonlinear programming problem arising in the projective method for linear programming.* Tech. Rep., New York Univ.

Renegar, J. 1986. *A polynomial-time algorithm, based on Newton's method, for linear programming.* Rep. MSRI 07118-86, Math. Sci. Res. Inst., Berkeley, Calif.

Rinaldi, G. 1986. The projective method for linear programming with box-type constraints. See Megiddo 1986d

Smale, S. 1983. On the average number of steps of the simplex method of linear programming. *Math. Program.* 27: 241–62

Smale, S. 1986. *Algorithms for solving equations.* Presented at the Int. Congr. Math., Berkeley, Calif.

Tardos, E. 1985. *A strongly polynomial algorithm to solve combinatorial linear programs.* Rep. 84360-OR, Inst. Econ. Oper. Res., Univ. Bonn

Todd, M. J. 1986. Polynomial expected behavior of a pivoting algorithm for linear complementarity and linear programming problems. *Math. Program.* 35: 173–92

Todd, M. J., Burrell, B. P. 1986. An extension of Karmarkar's algorithm for linear programming using dual variables. See Megiddo 1986d

Vanderbei, R. J., Meketon, M. J., Freedman, B. A. 1986. A modification of Karmarkar's linear programming algorithm. See Megiddo 1986d

Yudin, D. B., Nemirovsky, A. S. 1976. Informational complexity and effective methods for solving convex extremum problems. *Economica i Mat. Metody* 12