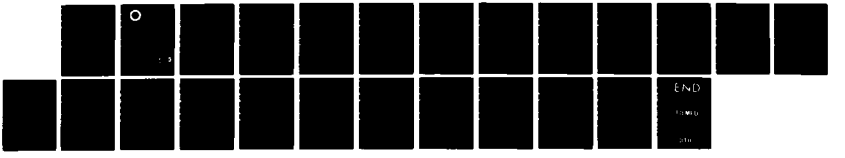
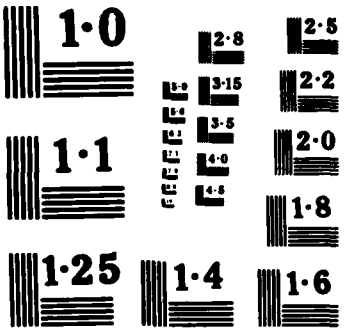
AD-A155 714 A PRACTICAL APPROACH TO KARMARKAR'S ALGORITHM(U) 1/1
STANFORD UNIV CA SYSTEMS OPTIMIZATION LAB I J LUSTIG
JUN 85 SOL-85-5 N00014-85-K-0343

UNCLASSIFIED F/G 12/1 NL





Systems
Optimization
Laboratory

A Practical Approach to Karmarkar's Algorithm

by

Irvin J. Lustig

TECHNICAL REPORT SOL 85-5

June 1985

Department of Operations Research
Stanford University
Stanford, CA 94305

2

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

A Practical Approach to Karmarkar's Algorithm

by

Irvin J. Lustig

TECHNICAL REPORT SOL 85-5

June 1985

Research of this report was supported by a National Science Foundation Graduate Fellowship.

Reproduction of this report was partially supported by the Department of Energy Contract DE-AM03-76SF00326, PA# DE-AT03-76ER72018; National Science Foundation Grants DMS-8420623, DCR-8413211 and ECS-8312142; and Office of Naval Research Contract N00014-85-K-0343.

Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do **NOT** necessarily reflect the views of the above sponsors.

A Practical Approach to Karmarkar's Algorithm

Irvin J. Lustig
Department of Operations Research
Stanford University
Stanford, CA 94305

ABSTRACT

A practical approach to implementing Karmarkar's algorithm is discussed. A variant of the algorithm is proposed which still has polynomial complexity and which eliminates the need for Karmarkar's canonical form. This method allows upper and lower bounds to be used and does not require knowledge of the objective value. Some heuristics are given which alleviate certain computational difficulties that arise when a practical implementation of the algorithm is attempted. A FORTRAN program is described that allows one to study its convergence properties.

Keywords: Linear Programming; Karmarkar's algorithm, Simplex Method; Projective Method, Least-Squares Problems

This material is based upon work supported under a National Science Foundation Graduate Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

Section 0: Introduction

Narendra Karmarkar of AT&T Bell Laboratories has described a new polynomial-time algorithm for linear programming that has generated a large amount of press coverage. The important theoretical result he presented at the Symposium on Theory of Computing in Spring of 1984 has been well received, but statements regarding the practical efficiency of the method have stirred up much controversy among experts in the field. His initial claim that the algorithm was 50-100 times more efficient than the simplex method was recently repeated by him at the Operations Research Society of America meeting in Boston in April, 1985, but the type of problems for which these results were claimed were for a very small number of test problems specially structured to favor his approach [5]. Tomlin has attempted to solve some less structured problems, and has not been able to duplicate Karmarkar's less spectacular computation times on these smaller problems [8]. Karmarkar [4] has claimed that "any undergraduate should be able to read my paper and write a program that is 10 times faster than the simplex method." This manuscript describes the efforts of the author, a graduate student, to validate this last claim.

Section 1 describes one possible way to apply Karmarkar's method to general linear programs and some of the basic difficulties that arise. Section 2 proposes a variant of the method that remains polynomial and allows general linear programs to be solved. Section 3 discusses how the method is applied in practice, the generation of interior feasible points, and the problem of null variables. Section 4 describes an iterative scheme to find the projected gradient on each iteration. Section 5 discusses an implementation of the algorithm and some computational results. Section 6 concludes the paper with a discussion of some future research.

Section 1: Solution of General Linear Programs

An initial reading of Karmarkar's paper [2] indicates that a practical implementation of the algorithm is not easy to come by. One difficulty arises because projective transformations are used to transform any general linear program to the following canonical form with a homogeneous right-hand side and a convexity constraint:

$$
\begin{array}{lll}
 & \min\ c^T x & c \in \mathbf{R}^n \\
\text{(LP0)} & \text{s.t. } Ax = 0 & A \in \mathbf{R}^{m\times n},\ b \in \mathbf{R}^m \\
 & e_n^T x = 1 & e_k = (1,1,\ldots,1)^T \in \mathbf{R}^k,\ \forall\, k \geq 1 \\
 & x \geq 0. &
\end{array}
$$

Todd and Burrell [7] and Tomlin [8] have described methods to do this transformation, but these methods either add unnecessary rows and columns to the linear program, and/or depend on the existence of some upper bound on the sum of all of the variables. For practical implementations, finding such a bound may be as difficult as solving the linear program and using this bound can cause numerical instability during the course of solution. The procedure described herein has the advantages that only one dense column in Phase I needs to be added and the bad scaling properties of previously suggested methods are not present.

Suppose we are given the following linear program:

$$
\begin{array}{lll}
 & \min\ c^T x & c \in \mathbf{R}^n \\
\text{(LP1)} & \text{s.t. } Ax = b & A \in \mathbf{R}^{m\times n},\ b \in \mathbf{R}^m \\
 & x \geq 0 &
\end{array}
$$

with known feasible point $w \in \mathbf{R}^n$, $w>0$ and known minimum objective value z^*. Karmarkar [4] indirectly suggested that a projective tranformation be used to bring this program into the canonical form of (LP0), thereby creating the following linear program:

$$
\begin{aligned}
& \min\ [c^T D, -z^*]\bar{x} \\
\text{(LP2)}\quad & \text{s.t. } [AD, -b]\bar{x} = 0 \\
& e_{n+1}^T \bar{x} = 1 \\
& \bar{x} \geq 0,
\end{aligned}
$$

where $D = \text{diag}[w_1, w_2, \ldots, w_n]$. (LP2) is in the canonical form for Karmarkar's algorithm [2] (henceforth **KA**). It also has not changed the sparsity structure of the original problem, except for the addition of the convexity constraint. If the **KA** algorithm is applied directly to this linear program, a new projected space is created on each iteration in order to move the current iterate to the center of a simplex. Therefore, a series of three linear spaces are used to solve the linear program: The first space is that of (LP1), the second is (LP2), and the third is artificially created on each iteration when the projective step is made. Todd and Burrell [7] and Tomlin [8] have not used the form of (LP2) to solve (LP1), and the equivalence of (LP1) and (LP2) is not apparent in Karmarkar's paper [2], due to the addition of the column for $\bar{x}_{n+1}$.

Iterates of the **KA** algorithm are calculated in terms of $\bar{x}$, a vector in $\mathbf{R}^{n+1}$. When the algorithm stops with a solution $\bar{x}^*$, the solution x^* to (LP1) is given by the inverse projective transformation $x^* = [D,0]\bar{x}^* / \bar{x}^*_{n+1}$. Furthermore, the knowledge of the finite minimum to (LP1) guarantees the existence of $\bar{x}^*$, with an objective value for (LP2) of 0, as the following theorem shows:

Theorem 1 Suppose (LP1) has a finite optimal solution x^*. Then (LP2) has an optimal solution at $\bar{x}^* = (D^{-1}x^*, 1)/(1 + e_n^T D^{-1} x^*)$.

Proof It is clear that $\bar{x}^*$ is feasible for (LP2) and that its objective value is 0, since $c^T x^* = z^*$. Now suppose $\exists\ \hat{x}$ that is feasible for (LP2) and that $[c^T D, -z^*]\hat{x} < 0$. There are 2 cases to consider, depending on the value of $\hat{x}_{n+1}$.

Case 1: Assume $\hat{x}_{n+1} > 0$. Let $x = [D,0]\hat{x} / \hat{x}_{n+1}$. Then x is feasible for (LP2). $[c^T D, -z^*]\hat{x} < 0 \Rightarrow c^T [D,0]\hat{x} < z^* \hat{x}_{n+1} \Rightarrow c^T x < z^* \Rightarrow x^*$ is not optimal for (LP1).

Case 2: Assume $\hat{x}_{n+1} = 0$. Let $x=x^*+[D,0]\hat{x}$. Then $[AD,-b]\hat{x}=0 \Rightarrow A[D,0]\hat{x}=0$, so that x is feasible for (LP1). $[c^TD,-z^*]\hat{x} < 0 \Rightarrow c^T[D,0]\hat{x} < z^*\hat{x}_{n+1} \Rightarrow c^T[D,0]\hat{x} < 0 \Rightarrow c^Tx=z^*+c^T[D,0]\hat{x} < z^* \Rightarrow x^*$ is not optimal for (LP1).

Hence, in either case, $\hat{x}$ cannot exist, and $\overline{x}^*$ must be optimal for (LP2). ●

In fact, one can see from the above proof that $\hat{x}_{n+1}=0$ and $[c^TD,-z^*]\hat{x} <0$ implies the existence of an unbounded optimal solution in (LP1) by simply moving in the direction $[D,0]\hat{x}$ from x^* in the feasible region for (LP1).

Difficulties with running the **KA** algorithm directly on (LP2) are due to two factors. If the optimal $\overline{x}^*$ found by the algorithm has $\overline{x}^*_{n+1} = 0$, then a nonnegative objective value for $\overline{x}^*$ only implies that the original problem has an infinite ray along which it is optimal. Unfortunately, the **KA** algorithm does not yield the endpoint of this ray for (LP1). It seems difficult to prove that this condition will not occur. The second difficulty arises when one tries to extend the basic idea of projecting (LP1) to (LP2) when the linear program has variables with upper bounds. This problem occurs because the upper bounds do not project to a simple linear ratio test when performing the iterative step. The projective transformation used on each iteration that maps (LP2) to the third space mentioned above destroys this linearity. With this motivation, we modify the basic idea of Karmarkar's algorithm to avoid these and other difficulties.

Intuitively, projecting the current iterate to the center of the feasible region is also a good way of solving linear programs. In fact, there seems good reason to consider an algorithm which simply brings each current iterate into the center of some simplex by a projective transformation, moves in the direction of optimality of the linear function over the inscribed sphere, and then uses an inverse projective transformation to bring the new point back to a point in the original space. Such an algorithm works only with iterates in the original space of the linear program and uses projective transformations only to move from the current iterate to the next one. When doing this, there is no need to bring a linear program to any canonical form like (LP0). Furthermore, there are now only two spaces used: the original linear program (LP1) and the space created on each iteration. The next section will show that this variant of the **KA** algorithm also remains polynomial.

Section 2: The New Projective Algorithm (NPA)

For this algorithm, we assume we are given a linear program in the form of (LP1), the feasible point w, and the known objective value z^*. Later, we will indicate how such a point w can be obtained and how z^* need not be given.

The **NPA** algorithm is as follows:

begin

$x^1 \leftarrow w$;

$k \leftarrow 1$;

while (not optimal) **do**

begin

$D \leftarrow \text{diag}[x_1^k, x_2^k, \ldots, x_n^k, 1]$

$$B \leftarrow \begin{bmatrix} (A, -b)D \\ \hline e_{n+1}^T \end{bmatrix}$$

$c_r \leftarrow [c, -z^*]$

$c_p \leftarrow [I - B^T(BB^T)^{-1}B]Dc_r$

$\hat{c} \leftarrow c_p / \|c_p\|$

$b' \leftarrow \frac{1}{n+1} e_{n+1} - \alpha r \hat{c}$

$\overline{b} \leftarrow (Db')/(e_{n+1}^T Db')$

$\tilde{b} \leftarrow \overline{b} / \overline{b}_{n+1}$

$k \leftarrow k+1$

$x^k \leftarrow \tilde{b}_{\{1,\ldots,n\}}$ (i.e, the first n components of $\tilde{b}$)

end

end

For theoretical purposes, the **NPA** algorithm will be considered terminated at optimality when $(c^T x - z^*) < 2^{-q}$, where $q=O(L)$, as in Karmarkar's paper. Also, r is set equal to $1/\sqrt{n(n+1)}$, which is the radius of the largest inscribed sphere in a simplex of dimension n. The parameter α is set to .25, to facilitate a polynomial proof of convergence. In a following section, it is shown how the above algorithm is modified for practical purposes by selecting α differently and using an appropriate convergence criterion.

In order to show the algorithm is still polynomial, it is necessary to invoke the use of a potential function similar to the one used by Karmarkar. In fact, this potential function will relate in a special way to that of Karmarkar. We define the potential function as:

$$g(x^k;c;z^*) = (n+1)\ln(c^T x^k - z^*) - \sum_{j=1}^{n} \ln(x_j^k) .$$

The following lemmas will show how this potential function leads to the polynomial complexity of the **NPA** algorithm.

Lemma 1 $g(x^{k+1};c;z^*) \leq g(x^k;c;z^*) - \delta$, where $\delta = 1/8$, as in [3].

Proof In each iteration of the **NPA** algorithm, the current iterate x^k is projected to the center of a simplex to create a linear program in the canonical form of (LP0) with n+1 variables, and a known minimum of 0, by Theorem 1. This linear program has the form:

$$\begin{array}{lll} & \min \ \overline{c}^T x' & \overline{c}^T = [c^T, -z^*]D \\ \text{(LP3)} & \text{s.t.} \ \overline{A}x' = 0 & \overline{A} = [A, -b]D \\ & e_{n+1}^T x' = 1 & \\ & x' \geq 0 & \end{array}$$

Applying Theorem 2 of [3], one finds that $f'(b') \leq f'(a_0) - \delta$, where b' is as in the **NPA** algorithm, $a_0 = e_{n+1}$, $\delta = 1/8$, and f' is Karmarkar's potential function for the transformed linear program (LP3), as found in [3]:

$$f'(x')=(n+1)\ln(\bar{c}^T x')-\sum_{j=1}^{n+1}\ln(x'_j)\ .$$

Hence, the reduction in Karmarkar's potential function implies that

$$(n+1)\ln(\bar{c}^T b')-\sum_{j=1}^{n+1}\ln(b'_j)\le(n+1)\ln(\bar{c}^T a_0)-\sum_{j=1}^{n+1}\ln\frac{1}{n+1}-\delta\ .$$

Since $\tilde{b}=(Db')/b'_{n+1}$, effectively scaling $\tilde{b}$ so that it is feasible, we can write

$$(n+1)\ln(c^T\tilde{b}-z^*)-\sum_{j=1}^{n+1}\ln(\tilde{b}_j/D_{jj})\le(n+1)\ln(Dc^T e_n-z^*)-\delta\ .$$

Now, $\tilde{b}_{n+1}=1$ and $D=\mathrm{diag}[x_1^k,x_2^k,\ldots,x_n^k,1]$, and $x^{k+1}=\tilde{b}_{\{1,\ldots,n\}}$ together imply

$$(n+1)\ln(c^T x^{k+1}-z^*)-\sum_{j=1}^{n}\ln(x_j^{k+1})\le(n+1)\ln(c^T x^k-z^*)-\sum_{j=1}^{n}\ln(x_j^k)-\delta\ ,$$

which yields the final result, by the definition of $g(x^k;c;z^*)$. ●

Lemma 2 $g(x^k;c;z^*)\le g(w;c;z^*)-k\delta$.

Proof Obvious, by Lemma 1. ●

Lemma 3 After k iterations of the modified algorithm,

$$(c^T x^k-z^*)\le\exp(q-\tfrac{k\delta}{n+1})(c^T w-z^*)(\det\mathrm{diag}[w_1,w_2,\ldots,w_n])^{(1/(n+1))}\ .$$

Proof Lemmas 2 and 1 together imply

$$(n+1)\ln(c^T x^k - z^*) - \sum_{j=1}^{n} \ln(x_j^k) \leq (n+1)\ln(c^T w - z^*) - \sum_{j=1}^{n} \ln(w_j) - k\delta .$$

Since we assume there to be an optimal solution to the linear program, each component of x^k is bounded above by e^q. Hence,

$$\ln(c^T x^k - z^*) \leq \frac{n}{n+1} q + \ln(c^T w - z^*) - \sum_{j=1}^{n} \ln(w_j) - \frac{k\delta}{n+1} .$$

After exponentiating both sides, the desired result is obtained. ●

One may argue that the exponential term in the result of Lemma 3 is quite large. However, as k increases, this term will eventually become small, and this yields a proof that a polynomial number of iterations will occur. Furthermore, the other two terms in the product are of the same order as the exponential term (i.e. their logarithms are polynomial in the size of the input) and these terms do not effect the overall complexity of the **NPA** algorithm.

Section 3: Applying the algorithm in practice

The first major modification to the **NPA** algorithm is in the choice of α. Karmarkar [4] had originally suggested that the following ratio test be performed:

$$\alpha^* = \max \{ 1/(n+1)r\hat{c}_j , \ \hat{c}_j > 0 \}$$

This would find a blocking variable x_j. (We know that $\alpha^* \geq 1$ because $\alpha^*=1$ corresponds to a distance of one radius of the inscribed sphere.) One would then use $\alpha=\rho\alpha^*$ as the step length in the algorithm, where $\rho=.85$, .90, or .95, say. Then ρ represents a fraction of how close one would get to any face of the positive orthant in $\mathbf{R}^n$. Tomlin's initial experimentation indicated that this was a good approach, as values of $\alpha^*>1$ were often observed, which indicated that the algorithm was moving in profitable directions to points far outside the sphere inscribed in the simplex. The cost of computing the value α^* is simply n ratio tests. Todd and Burrell [7] have suggested that a line search be performed in the direction $\hat{c}$ to minimize the potential function. It is not clear how beneficial this search would be, versus the above heuristic, which seems to work well in practice.

Given a general linear program in the form of (LP1), the first problem that arises is that of finding an interior point. As suggested by Karmarkar in [2], one may set up the following linear program to get an interior point:

$$\begin{aligned} & \min \ \lambda \\ \text{(LP4)} \quad & \text{s.t.} \ Ax+(b-Ae_n)\lambda = b \\ & \quad x \geq 0, \ \lambda \geq 0 \ . \end{aligned}$$

This linear program has an optimal value of 0 if and only if (LP1) is feasible. Furthermore, the vector $x=e_n$, $\lambda=1$, is feasible for this linear program. We can then apply the **NPA** algorithm directly to this linear program to yield a feasible point. If the program (LP1) is infeasible, we will probably take many iterations to detect that condition. However, if (LP1) has a strictly interior point, λ will become a blocking variable in relatively few iterations, with the iteration count depending upon how far e_n lies from any solution of (LP4). When λ blocks in the ratio test mentioned above, α is set to α^*, and the algorithm obtains a strictly interior point.

A heuristic argument can be given as to why the above procedure yields an interior point, if one exists. In each iteration of the algorithm, the variable λ is projected to a simplex as one of the variables of the linear program. Since $\lambda=0$ corresponds to all feasible solutions of (LP1) and to all optimal solutions of (LP4) and also to a face of the simplex artificially created in each iteration, one can see that the linear programs (LP3) and (LP4) have multiple optimal solutions. The **KA** and **NPA** algorithms both tend to go to the "center" of such a face of multiple optima, when starting with a point sufficiently far away from any boundary. This is a result of minimizing over the sphere inscribed inside the simplex, and the fact that this sphere is tangent to the face $\lambda=0$ of that simplex.

In Phase I, problems arise in the presence of null variables. For (LP1), x_j is said to be a null variable if $x_j=0$ in all feasible solutions for (LP1). Hence, a linear program with any null variables has no strict interior. It can be shown that a linear program has one or more null variables if and only if every basic feasible solution of (LP1) is degenerate. One should also note that randomly generated linear programs rarely have null variables, but null variables seem to occur quite often in practical problems.

The behavior of both the **KA** and **NPA** algorithms on (LP4) is quite interesting. The solution of (LP4) still lies on the face $\lambda=0$, but this face is not n-dimensional, as in the case when no null variables are present. It has been observed on various test problems that λ does not become the blocking variable in a small number of iterations; instead, the algorithm tends to "tail off" as it converges to a solution with $\lambda=0$. However, as iterations progress the algorithm seems to identify the null variables in an interesting way. Since the algorithm is converging, the change in those variables that are not null is very small compared to those that are null. If x_j^k is small, it must be projected relatively far to the center of the next simplex in the next iteration. It has been observed that for null variables that are converging to 0, the value of x_j^k is very small compared to that of its projected gradient $\hat{c}_j$.

A heuristic which takes advantage of this observation has been implemented. If a variable x_j is the blocking variable in the ratio test and $(x_j/(\hat{c}_j/x_j)) < \epsilon$, then α is set to α^*, driving x_j to zero. Note that the test divides $\hat{c}_j$ by x_j so that the values compared are related to each other within the space of (LP1). For practical testing, a value of $\epsilon=10^{-6}$ was found to be

suitable. After that variable has been set to 0, it is effectively ignored, and the algorithm proceeds until λ becomes blocking in a ratio test. During that time, other null variables may be identified and set to 0 as well. As soon as an interior point (ignoring any null variables that have been identified) has been generated, one can then solve (LP1).

The final difficulty in solving the linear program is the requirement that z^*, the optimal objective value of (LP1), be input to the algorithm. It does not seem practical to solve the combined primal-dual program as suggested by Karmarkar in [3]. Yinyu Ye has used in [9] a "cutting objective" method to solve this problem. This method is extremely easy to apply to the **NPA** algorithm, as it simply sets $c_r=[c,-c^Tx^k]$, effectively cutting off the objective value of (LP3) at 0. Ye has proved that this version of the algorithm will converge, but is polynomial only if the starting point w is "reasonably close" to the optimal solution. Ye's algorithm, in fact, may always be polynomial, but this seems to be extremely difficult to prove. One can show that $c^Tx^{k+1} < c^Tx^k$ for the cutting objective method, which implies that c_p is a descent direction. In practice, his method works well and was the method used in my implementation. The method also admits an elegant convergence criterion: $(\|c_p\|/|c^Tw|)<\epsilon$. For my implementation, a value of $\epsilon=10^{-6}$ was used. Later it will be shown why this convergence criterion has special significance in some other way.

Given a linear program with upper bounds, the **NPA** can be modified to handle them. Assume the linear program is in the form:

$$\begin{array}{lll} & \min & c^Tx \\ \text{(LP5)} & \text{s.t.} & Ax=b \\ & & 0 \le x \le u \end{array}$$

If lower bounds are present as well, the above form can be obtained by a simple translation of the variables.

Karmarkar has suggested using a flipping technique to handle the upper bounds. As the algorithm progresses, if $x_j^k > (u_j/2)$, then x_j is flipped by setting $x_j^k=u_j-x_j^k$, negating the $j^{\underline{th}}$ columns of c^T and A, and adjusting the value of b, so that the new x remains feasible. The upper bound for x_j is tested

using the ratio test:

$$\alpha = \max_j \left\{ \frac{u_j - x_j^k}{(u_j \hat{c}_{n+1} - x_j^k \hat{c}_j)(n+1)r} \text{, provided } (u_j \hat{c}_{n+1} - x_j^k \hat{c}_j) > 0 \right\}.$$

As the algorithm converges, the "flipping" of the variables should become less frequent. Note that this procedure preserves the upper bounds and is easy to implement. It should be noted that a proper proof of convergence has not yet been exhibited. The procedure has worked well on a small number of test problems.

Section 4: The calculation of the projected gradient

The largest computational burden in any algorithm based on Karmarkar's original method is in the calculation of the projected gradient $c_p=(I-B^T(BB^T)^{-1}B)Dc_r$ during every iteration. The computation involved can be viewed in many ways, and, in particular, as the following least-squares problem:

(LS1) $\min \|Dc_r - B^T(\frac{\pi}{\mu})\|_2$,

whose solution satisfies the normal equations $(BB^T)(\frac{\pi}{\mu})=BDc_r$. It follows that $c_p=Dc_r-B^T(\frac{\pi}{\mu})$ is the residual vector for the least-squares problem (LS1). If (LP1) has a non-degenerate optimal solution, then π converges to the optimal dual solution. In any case, μ always converges to zero. In the case of the cutting objective method of Ye, one can show that $\mu=0$ on every iteration. With that assumption, one can then write $c_p=D([c,-c^Tx^k]^T-[A,-b]^T\pi^k)$ on the $k^{\underline{th}}$ iteration. One can then see that the first n components of c_p are related to the complementary slackness conditions that hold at optimality of (LP1), because of the construction of D. Furthermore, the last component of c_p corresponds to the duality gap between (LP1) and its dual. So if D is converging to a non-degenerate solution, the size of the norm of c_p compared to the initial estimate (with $\pi=0$) is a reasonable convergence criterion. In the case of a degenerate optimal solution, the convergence criterion is still valid, but there is no reason to believe that π will converge to a specific dual solution. Since a degenerate optimal solution corresponds to multiple dual optima, the method used to compute c_p will determine the nature of the convergence of π. However the norm of c_p will still converge to zero.

M. A. Saunders has suggested an incremental scheme to find c_p by solving for a correction term to the vector π. This method will be presented in the context of the **NPA** algorithm presented in Section 2.

Before the first iteration, we set

$d \leftarrow c_r$ and

$\pi \leftarrow 0$.

On each iteration, we solve the least-squares problem

(LS2) $\min \|Dd - B^T(\begin{smallmatrix}\delta\\ \mu\end{smallmatrix})\|_2$

for the vector $(\begin{smallmatrix}\delta\\ \mu\end{smallmatrix})$.

We then update our past solutions:

$\pi \leftarrow \pi+\delta$,

$d \leftarrow d-[A,-b]^T\delta$,

and calculate the final projected gradient:

$c_p \leftarrow Dd-\mu e_{n+1}$.

Note that $Dd=D(c_r-[A,-b]^T\pi)$ should be converging to zero, by the complementary slackness condition. Hence, we are solving for smaller correction terms to π as the algorithm proceeds.

When Ye's cutting objective method is used, it is also necessary to update the last component of d on each iteration before solving the least-squares problem (LS2).

Section 5: Computational Results

The new projective algorithm described above was implemented with the cutting objective method of Ye using FORTRAN on an IBM 3081 Model KX2. Since the cutting objective method was used, no knowledge of the final objective value was needed by the program to obtain a solution. The majority of the test problems were the same as those run by Tomlin [8] in his tests. An additional test problem was created from an exponential counterexample to the simplex method and came from Blair [1].

In order to calculate c_p, we used the iterative algorithm LSQR of Paige and Saunders [6]. The incremental method described in the previous section was used, since the diminishing vectors Dd in the least-squares problem (LS2) allow LSQR to converge more rapidly than it does when applied to (LS1). A minimal preconditioner which scaled the columns of each least-squares problem to have a unit Euclidean norm was also implemented. Without this, the LSQR algorithm failed to converge in reasonable time on some problems.

The computational results are presented in Table 1. All CPU times represent the time in seconds to read in the data from MPS format, solve the linear program, and write out a solution. Columns labeled MINOS refer to the iteration counts and CPU times to solve the same problems using software based on the Simplex Method developed by M.A. Saunders of the Systems Optimization Laboratory at Stanford. The times to input the data and write out the solution for the **NPA** algorithm and for MINOS are equal because MINOS subroutines were used for these purposes.

While the CPU times reflect the difficulty of using LSQR directly in the **NPA** algorithm, the iteration counts indicate much promise for **NPA**. In order to reduce the work done on each iteration, a very good preconditioner would have to be implemented. It should be noted that the problems BRANDY, E226, and BANDM all have null variables and result in higher iteration counts in Phase I. The problem ISRAEL requires a large amount of time because of severe ill-conditioning. This problem was a notable omission from the results presented by Karmarkar in Boston [5].

It is interesting to note the amount of work done by this rather naive application of LSQR to find c_p. The number of iterations performed by LSQR can be linked to how well-conditioned the least-squares problems are. This conditioning can be poor on either the first few iterations of Phase I or the

final iterations of Phase 2, depending on the nature of the problem. Table 2 summarizes the results on the amount of work performed by LSQR. On most problems, the number of iterations performed by LSQR on (LS2) seemed to stabilize as the projective algorithm converged. This usually occurred when the algorithm had few degeneracies at optimality.

From these results, it is apparent that success or failure of Karmarkar's algorithm and its variants will depend solely on how fast c_p can be calculated. The ill-conditioning of the problems may cause difficulties at the beginning of Phase I or the end of Phase 2. Most likely, any method that finds c_p will find difficulties because of this ill-conditioning. It is clear that further research is necessary to understand how Karmarkar's algorithm creates these conditioning problems.

It is also important to note the low iteration counts of the **NPA** algorithm, when the cutting objective method of Ye is used. The promise provided by these low counts is encouraging enough to warrant further research into Karmarkar's algorithm and its variants.

Section 6: Conclusions and Acknowledgements

Karmarkar's algorithm has variants which are easier to understand in the context of classical linear programming theory. These variants allow one to implement computer programs that run with low iteration counts and solve linear programs with upper bounds.

Karmarkar claims there are some problems which will be solved much faster by his algorithm than by the standard simplex method, but it is probable that an adapatation of the simplex method which does some interior probing may also do well on these same problems. For most linear programs, however, the cost of doing each iteration might well outweigh any low iteration counts obtainable. It is still questionable whether Karmarkar's algorithm or some variant will become "the method of choice" for linear programming in the near future.

The author would like to thank George Dantzig, Narendra Karmarkar, Edward Klotz, Nimrod Megiddo, and Michael Saunders for suggestions, insights and encouragement as this work progressed.

This material is based upon work supported under a National Science Foundation Graduate Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

Table 1: Computational Results of New Projective Algorithm.
All times in CPU seconds on an IBM 3081 Model KX2.

Problem Name	Rows	Columns	MINOS (Simplex)		New Projective Algorithm		
			Total Itns.	CPU Time	Phase I Itns.	Phase II Itns.	CPU Time
EXP1	10	17	52	0.6	6	11	0.4
AFIRO	28	32	6	0.5	3	11	0.8
ADLITTLE	57	97	126	1.0	9	20	12.3
SHARE2B	99	79	91	1.0	7	14	67.4
ISRAEL	175	142	338	4.2	9	24	636
BRANDY	221	249	292	4.1	16	19	215
E226	226	282	572	7.9	17	42	644
BANDM	306	472	362	6.4	18	37	771

Table 2: LSQR Iteration Counts for **NPA** algorithm.
NS = Never Stabilized, continuing to increase.

Problem Name	LSQR Iteration Counts			Max Attained Phase/Itn
	Minimum	Maximum	"Stable"	
EXP1	12	22	12	1/1
AFIRO	26	55	30	1/1
ADLITTLE	83	329	~160	1/1
SHARE2B	607	786	~775	2/10
ISRAEL	270	8853	~650	1/1
BRANDY	136	1008	NS	2/18
E226	130	1332	~1100	1/1
BANDM	175	1679	NS	2/34

References

[1] C. Blair, "Some Linear Programs Requiring Many Pivots," Faculty Working Paper No. 867, College of Commerce and Business Administration, University of Illinois At Urbana-Champaign, (May 1982).

[2] N. Karmarkar, "A New Polynomial-Time Algorithm for Linear Programming," Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, p. 302-311 (May 1984).

[3] N. Karmarkar, "A New Polynomial-Time Algorithm for Linear Programming," Combinatorica, Vol. 4, No. 4, p. 373-395 (Nov. 1984).

[4] N. Karmarkar, Seminar Presentation at Stanford University, (Jan. 1985), and private communication (Nov. 1985).

[5] N. Karmarkar, Seminar Presentation at ORSA/TIMS, Boston, MA, (May 1985)

[6] C.C. Paige and M.A. Saunders, "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares," ACM Transactions on Mathematical Software, Vol. 8, p. 43-71 (1982).

[7] M.J. Todd and B.P. Burrell, "An Extension of Karmarkar's Algorithm for Linear Programming Using Dual Variables," Technical Report No. 648, School of Operations Research and Industrial Engineering, College of Engineering, Cornell University, Ithaca, NY, (Jan 1985).

[8] J.A. Tomlin, "An Experimental Approach to Karmarkar's Projective Method for Linear Programming," Ketron, Inc., Mountain View, CA., (Jan. 1985).

[9] Y. Ye, "A Large Group of Projections for Linear Programming - Cutting and K-Projective Methods," Ph.D. Dissertation, Department of Engineering and Economic Systems, Stanford University, (in progress).

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER SOL 85-5	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Practical Approach to Karmarkar's Algorithm		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Irvin J. Lustig		8. CONTRACT OR GRANT NUMBER(s) N00014-85-K-0343
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Operations Research - SOL Stanford University Stanford, CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR-047-064
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research - Dept. of the Navy 800 N. Quincy Street Arlington, VA 22217		12. REPORT DATE June 1985 13. NUMBER OF PAGES 20 pp.
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

linear programming
Karmarkar's algorithm
simplex method
projective method
least-squares problems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A practical approach to implementing Karmarkar's algorithm is discussed. A variant of the algorithm is proposed which still has polynomial complexity and which eliminates the need for Karmarkar's canonical form. This method allows upper and lower bounds to be used and does not require knowledge of the objective value. Some heuristics are given which alleviate certain computational difficulties that arise when a practical implementation of the algorithm is attempted. A FORTRAN program is described that allows one to study its convergence properties.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

END

FILMED

7-85

DTIC