SOME MODIFICATIONS AND EXTENSIONS OF KARMARKAR'S MAIN ALGORITHM WITH COMPUTATIONAL EXPERIENCES

by

BYEONG-SOO KIM

Bachelor of Science

Seoul National University

Seoul, Republic of Korea

1981

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1987

Thesis
1987
K49s
cop.2

OKLAHOMA STATE UNIVERSITY LIBRARY

SOME MODIFICATIONS AND EXTENSIONS

OF KARMARKAR'S MAIN ALGORITHM

WITH COMPUTATIONAL

EXPERIENCES

Thesis Approved:

Donald W. Grace
Thesis Adviser

J P Chandler

L. E. Hedrick

Norman N. Durham
Dean of the Graduate college

1275676

PREFACE

The Karmarkar algorithm and its modifications are studied in this thesis. A modified line search algorithm with extended searching bound to the facet of the simplex is developed and implemented. Using this modification, a modified row partition method is tested. Both algorithms are coded in Fortran 77 and compared their performances with the original Karmarkar algorithm. The modifications are promising and other extensions are encouraged.

I wish to express my sincere appreciation to my major adviser, Dr. D.W. Grace, for his guidance, concern, and invaluable help. I am also thankful to my committee members, Dr. J.P. Chandler and Dr. G.E. Hedrick for their advice. A special thank goes to Mr. Yoon-sik Kim for his kind help in English correction.

My parents deserve my deepest appreciation for their patience, encouragement, and financial support.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

From Dantzig to Karmarkar

The Simplex Method

For solving a linear programming (LP) problem, the well-known simplex method was developed by Dantzig [4]. The procedure of the simplex method is summarized as follows :

A linear programming problem has the form

$$\begin{array}{lll} \text{Min} & C^T X & C , X \in R^n \\ \text{s.t.} & AX = b & A \text{ is an } m \times n \text{ matrix, } m < n \\ & X >= 0 & \end{array}$$

If an extreme point is X', then the matrix A can be divided into [B,N], where B is the basis with an m x m full rank matrix comprised of the columns of coefficients of the non-zero variables, and N is the non-basis with an m x (n - m) matrix.

Now, by decomposing X into (X_B, X_N), AX = b can be written as $BX_B + NX_N = b$. Therefore, $C^T X = C_B^T X_B + C_N^T X_N$ $= C^T X' + (C_N^T - C_B^T B^{-1} N) X_N$. If $C_N^T - C_B^T B^{-1} N$ is non-negative, then X is an optimal extreme point.

The main algorithm proceeds as follows :

1. Find a starting solution X with basis B.

2. If $C_N^T - C_B^T B^{-1} N$ is nonnegative, then stop.

   Else pick the most positive component $C_B^T B^{-1} a_j - c_j$.

3. Let $B^{-1} b = b'$ and $\pi = \min \{ b_i'/y_{ij} , y_{ij} > 0 \}$, where y_{ij} is the i_th component of $y_j = B^{-1} a_j$.

4. Get the new extreme point X by calculating

   $X_{Bi} = b_i' - \pi y_{ij}$ for i=1,...,m

   $X_j = \pi$

   other X_i's are equal to zero.

5. Go to 2.

Figure 1 shows that the simplex method which traverses the boundary of the feasible polytope.

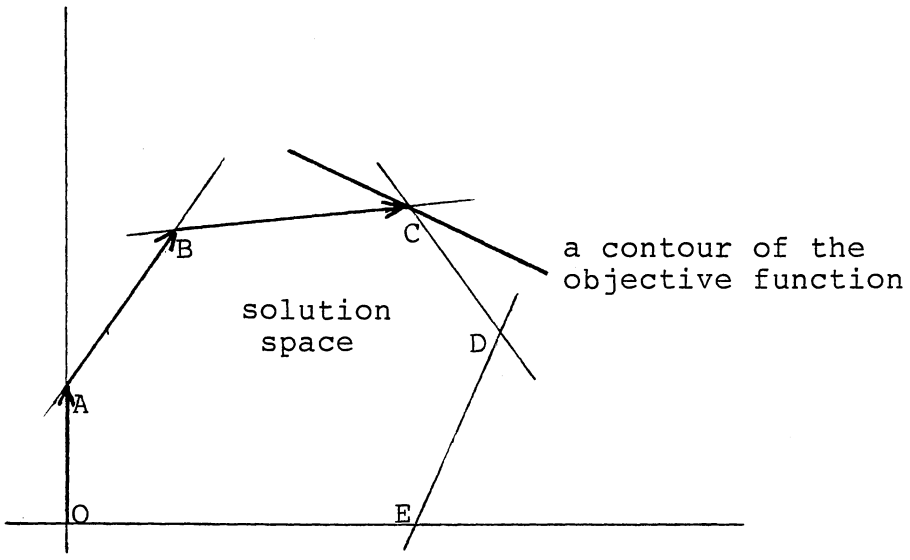


Figure 1. The Simplex Method

This algorithm solves the underdetermined system $AX = b$ by traversing the edges of the solution space from one extreme point to another one in a systematic manner, driven by the criterion that each move improves the objective function.

The Ellipsoid Algorithm

In order to find a better algorithm than the commonly used Dantzig simplex method [4], diverse linear programming (LP) ideas have been essayed. One such attempt is the ellipsoid algorithm developed by Khachiyan [9]. His algorithm runs in polynomial time, whereas the simplex method runs in exponential time only. Also, the geometric interpretation of the ellipsoid algorithm is totally different from the simplex method, in that it circumscribes the solution space with a shrinking ellipsoid.

The ellipsoid algorithm is summarized as follows :

1. A feasible region S is defined as $S = \{X \mid AX <= b\}$, where A, X, and b are defined as before.
2. Construct an initial ellipsoid E^0 which contains S.
3. Construct a new ellipsoid E^{k+1} which fully contains the half-ellipsoid $(0.5E^k)$. If the center of E^{k+1} is feasible, then stop.
4. The procedure will eventually terminate since the volume of E^k will be contained in S as k goes to infinity; The center of E^k is contained in S. Otherwise there is no solution.

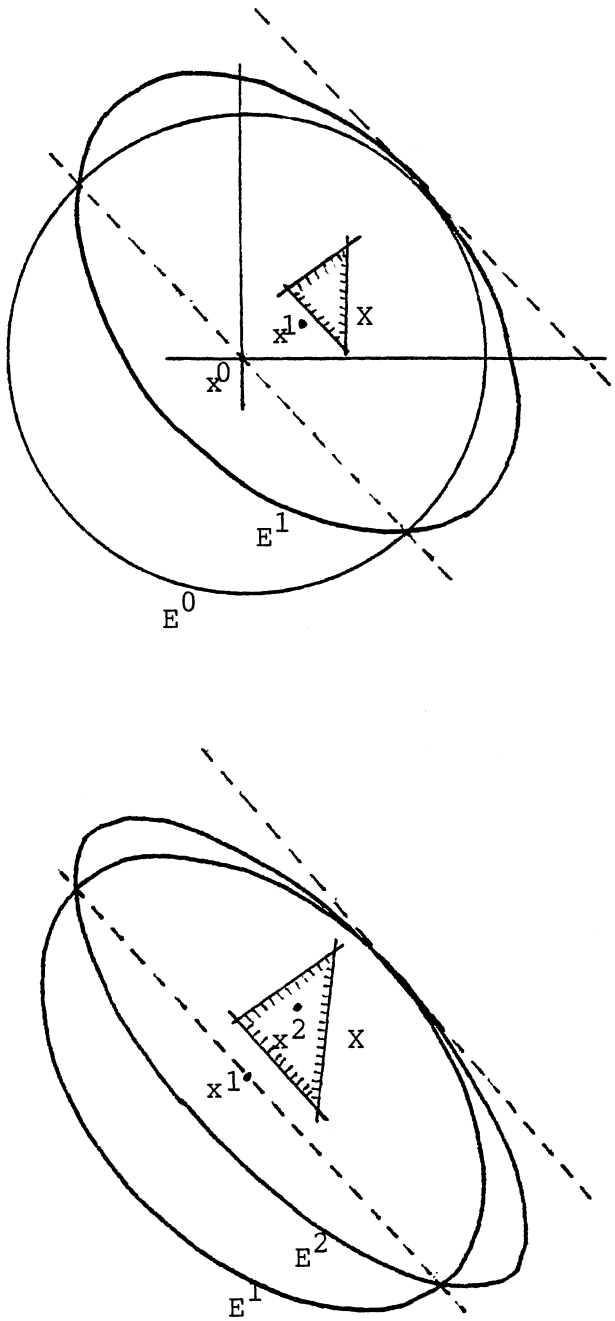


Figure 2. The Ellipsoid Algorithm

Figure 2 shows Khachiyan's ellipsoid algorithm which starts with an initial hypersphere and successive ellipsoids.

After the feasible solution (the center of E^k) is found, one should formulate a feasible-point problem where the feasible region is an arbitrary small volume containing the feasible solution. This causes the number of iterations to be very large, thereby making this procedure much more expensive than the simplex method.

The Karmarkar Algorithm

Finally, in 1984, Narendra Karmarkar [8] introduced a new polynomial-time algorithm for complex LP problems whose method requires less time-complexity than that of Khachiyan's ellipsoid algorithm [9].

As a major advantage, Karmarkar's algorithm runs within a polynomial-time bound; whereas, the well-known simplex method requires exponential-time in the worst-case. Because of this polynomial-time bound, the benefit of Karmarkar's algorithm increases as the problem size grows : Karmarkar claimed that his algorithm performs 50 to 100 times faster than the simplex method on large-sized LP problems.

Briefly, the Karmarkar algorithm works in the non-negative poly-dimensional space (e.g., the 1st quadrant in a 2-dimensional problem) to find a direction toward the optimal point. This concept is in contrast to the simplex

method which traverses the vertices of the polytope boundaries to find the optimal point: unlike the conventional simplex method, the Karmarkar algorithm attempts to find an objective-improving direction instead of moving from one vertex to another in order to find the optimal solution. Instead of enumerating extremely many vertices in the worst-case large scale LP problems in the simplex method, many computational iterations can be saved if Karmarkar's algorithm is used.

However some specialists earlier argued [10] [11] that Karmarkar used his experimental inputs only to favor his approach, a view that is not shared by everyone. The controversy regarding whether his algorithm is better than the well-known simplex method has abated as the Karmarkar algorithm has developed.

Since Karmarkar's algorithm came into existence rather recently, some thought-provoking aspects of the concepts have been noticed, such as the somewhat unclear transformational process from the original LP problem to canonical form as well as the possibility for modification and improvement of the concept. Therefore, the main purpose of this thesis is to clarify Karmarkar's original idea, to design and to program a better modified algorithm, and to compare its performance with Karmarkar's original algorithm.

In Chapter II, the main idea of Karmarkar's algorithm is reviewed because it is entirely different algorithm for linear programming and somewhat difficult to understand due

to several new concepts such as projective transformation, potential function, and sliding objective function.

In Chapter III, the transformation process from general LP problems to Karmarkar's main algorithm is explained because he did not show exactly how to transform from the general LP problems to the canonical form from which his algorithm starts. For the next modification, a line search (the Fibonacci method is used) for the "potential function" is performed explicitly as Todd and Burrell [13] suggested. Here, the searching bound is extended to the facet of the simplex rather than limiting the search to the inscribed sphere of the simplex to observe how beneficial this modification would be.

Chapter IV contains the comparison between the original Karmarkar algorithm and the modified methods. Their performance on relatively small LP problems is compared.

In the next chapter, duality is chosen as an extension of Karmarkar's algorithm because duality is deeply related with postoptimal analysis (infeasibility can be recovered by duality) and it has great economic significance. Here, Todd and Burrell's duality method [13] is examined.

The final chapter contains a summary, conclusions, and suggestions for future work.

Literature Review

Since Dantzig [4] developed the well-known and popular simplex method, a number of researchers before Karmarkar [8]

have tried to design a polynomial-time algorithm for linear programming. One of these, Khachiyan [9], developed an ellipsoid algorithm which has polynomial-time convergence. Although his ellipsoid method was mathematically attractive, it was not practical because it cost much more than the simplex method to implement: the structure per each iteration is totally different from the simplex method, and unfortunately, the computation associated with each iteration is more costly than that of the simplex method. Also, the iteration count is usually very large.

Since Karmarkar [8] published his new polynomial-time algorithm, some research regarding step size has been reported. Kalantari [7] reported that faster convergence is possible with a modified algorithm in which the step size α varies with an improved step at each iteration. Todd and Burrell [13] suggested a line search along the negative gradient of the potential function, instead of fixing α at every iteration.

Vandervei, et al. [15] proposed a linear transformation of a feasible solution, which sets the transformed feasible solution to be uniformly rescaled. They also proved a convergence of this modified algorithm and provided a stopping rule. On the other hand, Cavalier and Soyster [2] considered the same modification and showed a better result with small size LP problems. Also they examined ill-conditioned problems which caused some difficulties in calculating the inverse of the matrix BB^T, where B is the

matrix of the constraints in Karmarkar's canonical form.

Cavalier and Schall [3], with the above modification, proposed yet another algorithm for maintaining feasibility, and made an efficient implementation with the row partitioning scheme. But no convergence criterion was provided and their method was restricted to inequality constraints.

On the other hand, Gill et al. [5] presented an application of a barrier transformation to a linear program and pointed out an equivalence between the barrier method and Karmarkar's algorithm with a suitable choice of a barrier parameter. Also, they derived a formal equivalance between the projected Newton search direction and the direction of the projected gradient in Karmarkar's algorithm.

Various research results have been reported with regard to the unknown optimal objective value, C* (Min $C^T X$.) Lustig [11] proposed the "cutting objective method" to update the value of the objective function at each iteration and gained a good result. Todd and Burrell [13] devised the duality algorithm by continually updating a lower bound $z <= C^*$ and had dual optimal solutions as a by-product. Also they proposed a method to transform the general LP problem into Karmarkar's canonical form in which their duality algorithm could be applied. While quoting from Tomlin [14], Hooker [6] not only suggested the conversion from a given arbitrary LP problem to Karmarkar's canonical

form but also gave a general survey of Karmarkar's main algorithm with numerical examples. Meanwhile, Anstreicher [1] provided a totally different method for finding C* based on the geometric viewpoint, and established a convergence criterion.

CHAPTER II

REVIEW OF KARMARKAR'S MAIN ALGORITHM

In this chapter, the Karmarkar algorithm is reviewed in order to provide a general understanding of the concept.

Let us consider the linear programming problem

$$\begin{array}{lll} \text{Min} \quad C^T X & & C \,,\, X \in R^n \\ \text{s.t.} \quad AX = b & & A \text{ is an } m \times n \text{ matrix.} \end{array} \qquad (1)$$

This can be transformed into Karmarkar's canonical form of

$$\begin{array}{lll} \text{Min} \quad C^T X' & & C \,,\, X' \in R^{n+1} \\ \text{s.t.} \quad AX' = 0 & & A \text{ is an } m \times (n+1) \text{ matrix.} \\ \qquad e^T X' = 1 & & e^T = (1,1,\ldots,1) \end{array} \qquad (2)$$

This transformation will be shown further in the next chapter by using a projective transformation.

Now, by taking a new transform T such that

$$T(X') = D^{-1}X' \,/\, e^T D^{-1} X' \qquad \text{where } D = \begin{vmatrix} x'_1 & & 0 \\ & x'_2 & \\ & & \cdot \\ 0 & & x'_{n+1} \end{vmatrix}$$

is a diagonal matrix with a feasible solution X^0, where $X^0 = (x^0_1 \,,\ldots,x^0_{n+1})$, the transformation T maps the

initial feasible point into the center of the simplex. Figure 3 of the next page shows the transformation on the surface of $\Omega^2 = \{ X \in R^3 \mid \sum_{i=1}^{3} xi = 1 \}$ for this centering scheme.

Using the inverse mapping of T : $T^{-1}(Y) = DY / e^T DY$, where $Y = T(X')$, the above problem can be reorganized as

$$\begin{aligned} \text{Min} \quad & C^T DY / e^T DY \\ \text{s.t} \quad & ADY / e^T DY = 0 \\ & e^T DY / e^T DY = 1 \end{aligned} \qquad (3)$$

by substituting $T^{-1}(Y)$ in (2) for X'.

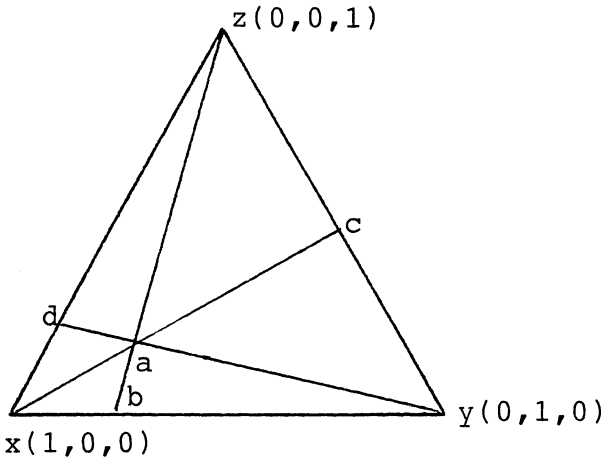
By maintaining $e^T Y = 1$, (3) can be rewritten as

$$\begin{aligned} \text{Min} \quad & C^T DY \\ \text{s.t.} \quad & ADY = 0 \\ & e^T Y = 1 \\ & Y >= 0 \end{aligned} \qquad (4)$$

because of strict positivity of $e^T DY$.

But, optimizing $C^T DY$ is an approximation of $C^T DY / e^T DY$ even if C affects only the numerator. Also it seems very hard to optimize with respect to a rational function $C^T DY / e^T DY$.

To preserve the linearity of the objective function, Karmarkar introduced the " potential function " $f : \Omega^n \dashrightarrow R$, $f(X') = \sum_{j=1}^{n+1} \ln (C^T X' / x_j')$. Then it is easy to show that $f'(Y) = \sum_{j=1}^{n+1} \ln (C^T DY / y_j) - \sum_{j=1}^{n+1} \ln x_j'$,



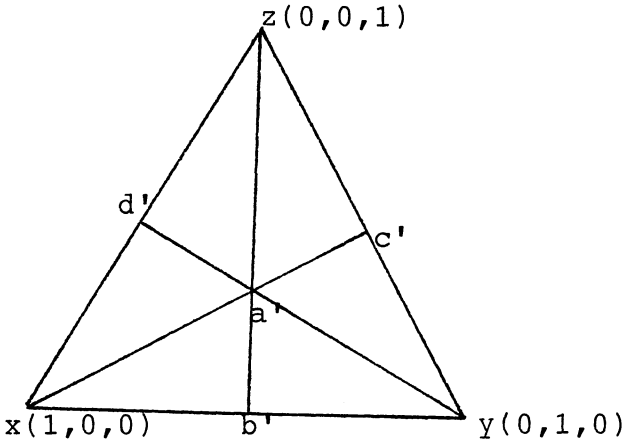


Figure 3. Transformation via T, where
a' = (1/3,1/3,1/3)

where f' is the transformed potential function, where $Y = T(X')$, and $y_j \in Y$.

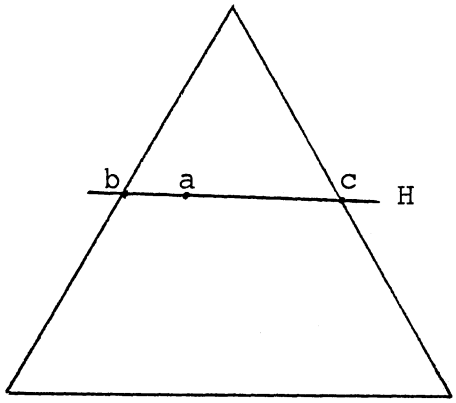
Clearly, f is the sum of ratios of linear functions which are transformed into another ratios of linear functions via T. Also it transforms the potential function into a new one. This is very important to show that $f(X')$ is decreased by a constant $d > 0$ in each iteration of Karmarkar's main algorithm. Formal description is summarized as follows :

Theorem 1 : Let Y' be the point that minimizes $C^T DY$ over $B(a_0, r) \cap \Omega'$, where $0 < \alpha < 1$, and B is a sphere centered at $a_0 = (1/n+1, \ldots, 1/n+1)$ with radius r ($r = 1/\sqrt{n(n+1)}$ is the radius of the largest inscribed sphere of the $\Omega' = \{ Y \mid ADY = 0 \} \cap \Omega^n$.) Then either (i) $C^T DY' = 0$ or (ii) $f'(Y') <= f'(a_0) - d$.

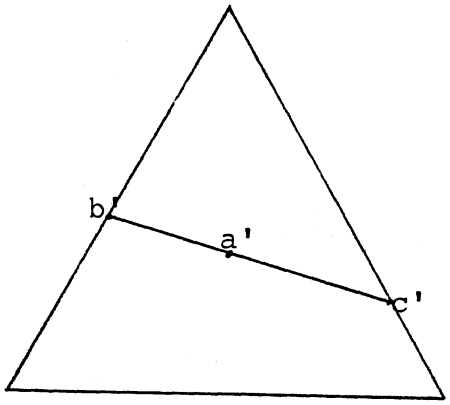
(proof) See Karmarkar's theorems.

Main algorithm

The procedure works with the system of (4). Any feasible point in (3) is mapped into the center of the simplex in (4), so that the direction of the negative projected gradient of $C^T D$ over the intersection of the polytope $H = \{ Y \mid ADY = 0 \}$ and the simplex $\Omega^n = \{ Y \mid \sum_{j=1}^{n+1} y_j = 1 \}$ can be searched.



a=(1/3,1/6,1/2) b=(1/2,0,1/2) c=(0,1/2,1/2)



a'=(1/3,1/3,1/3) b'=(3/5,0,2/5) c'=(0,3/4,1/4)

Figure 4. Transformation from the Equation (2) to the Equation (4) via T

Figure 4 shows the intersection between H and Ω^2, where $H = \{ (x, y, z) \mid x + y - 2z = 0 \}$ and $\Omega^2 = \{(x, y, z) \mid x + y + z = 1 \}$. Here, $D=diag(1/2,1/6,1/3)$ is assumed as the initial feasible solution. In Figure 4, the initial feasible point $a=(1/2,1/6,1/3)$ is transformed into the center of the simplex Ω^2, and the boundary points b and c are transformed into b' and c', respectively ($b' = \dot{T}(b) = D^{-1}b/e^T D^{-1}b = (4/3,0,1)/(7/3) = (4/7,0,3/7)$, and $c' = T(b) = D^{-1}c/e^T D^{-1}c = (0,4,1)/5 = (0,4/5,1/5)$.)

Having done with this transformation, the following Rosen's "Gradient Projection" method [11] is used in order to find the projected gradient vector over the $\Omega' = H \cap \Omega^n$.

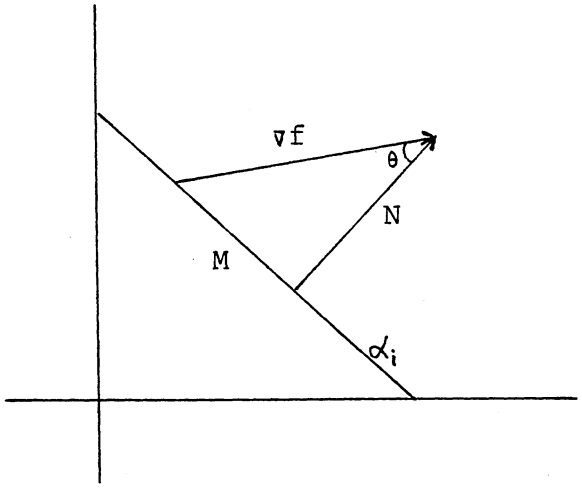


Figure 3. Rosen's Gradient Projection Method

Since the gradient ∇f is followed uphill in Figure 5 -- because ∇f cannot be followed without passing out of the solution space -- ∇f should be projected onto the boundary, giving an uphill direction (M on the Figure 5.)

When a_i is the outward pointing normal to the constraint, then the magnitude of $N = |\nabla f| \cos\theta = a_i^T \nabla f / |a_i|$, and the normalized direction of $N = a_i / |a_i|$. And M can be calculated by using ∇f and N.

$$M = \nabla f - N$$

$$= \nabla f - \frac{a_i}{|a_i|} \frac{a_i^T \nabla f}{|a_i|}$$

$$= [\, I - a_i (a_i^T a_i)^{-1} a_i^T \,] \nabla f \qquad (5)$$

For multiple active constraints, if (5) is applied to the matrix B in which each column is the outward pointing normal of an active constraint, then a_i can simply be replaced with B in the projection operator, giving $(I - B(B^T B)^{-1} B^T)$, and the desired result can be obtained.

Now, the above method can be summarized by the following algorithm (It is assumed that the minimization problem is worked here) :

step 1 : Start with a feasible solution X_0', then X_0' goes the center of the simplex by T.

step 2 : Project $C^T D$ onto the affine null-space of the feasible region.

$$B = \begin{vmatrix} AD \\ \hline e \end{vmatrix}$$, where A is an m x (n+1) matrix, and

$D = \mathrm{diag}(x_1, \ldots, x_{n+1})$.

Then the projected gradient c_p is

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

step 3 : By taking a step of length αr from the center, the objective function can be improved ($\alpha \in (0,1)$ is a fixed parameter.)

$$Y = (1/n+1,\ldots,1/n+1) - \alpha r \frac{c_p}{|c_p|}$$

step 4 : Obtain next X' by using inverse of T.

$$X' = DY / e^T DY$$

step 5 : If $C^T X' = 0$ or $C^T X' / C^T X^0 <= 2^{-L}$ then stop

$$(L=[\sum_{j=1}^{m}\sum_{i=1}^{n+1} \ln 2\ (|a_{ji}|+1) + \sum_{j=1}^{m} \ln 2\ (|b_j|+1)+\ln 2\ nm]+1$$

is an input parameter defined in Khachiyan's ellipsoid algorithm, where $a_{ji} \in A$, and $b_j \in b$.)

Else go to step 2.

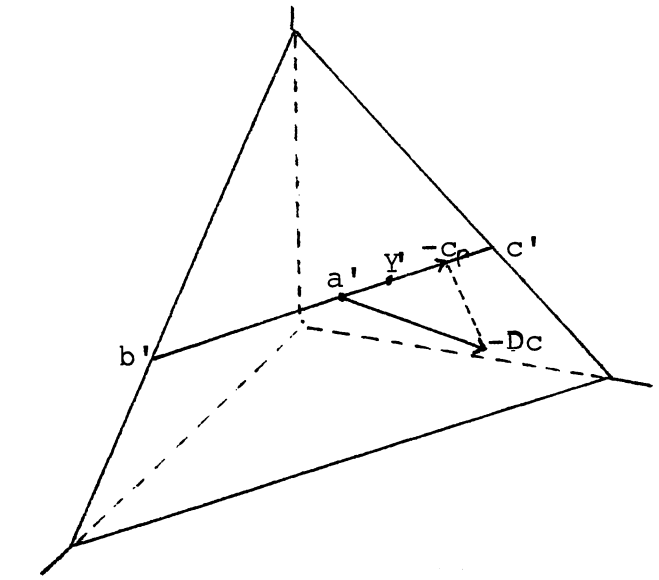
An example problem which has Karmarkar's canonical form is :

$$\text{Min} \quad x_1 - x_2 - 2x_3$$

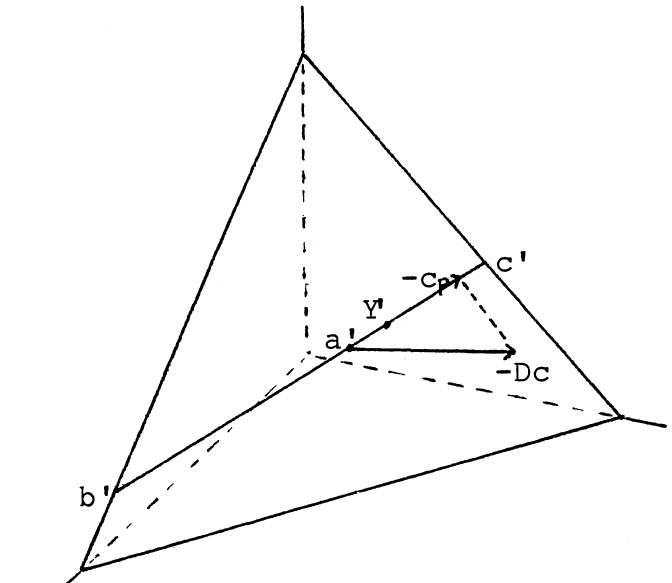
$$\text{s.t.} \quad x_1 + x_2 - 2x_3 = 0$$

$$x_1 + x_2 + x_3 = 1$$

Figure 6 shows the procedure of Karmarkar's main algorithm in the transformed space. The triangle is the simplex Ω^2, and the line segment (b'-a'-c') is the feasible polytope. In the first figure, the initial feasible solution $X^0 = (1/3,1/3,1/3)$, -DC is the negative gradient of the transformed objective function, and Y^1 is the improved solution at the end of the first iteration.



a'= (1/3,1/3,1/3) b'= (2/3,0,1/3)
c'= (0,2/3,1/3) Y'= (.282,.384,.333)

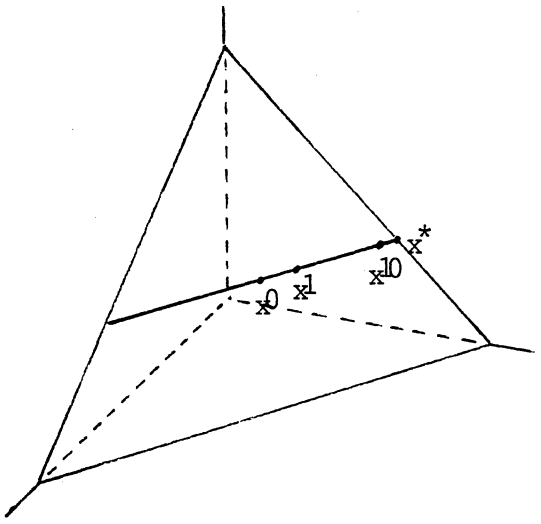


a'= (1/3,1/3,1/3) b'= (.941,.0,.059)
c'= (.0,.519,.484) Y'= (.274,.366,.360)

Figure 6. Illustrations of Three Variables Problem in the Transformed Space

Second figure shows the transformed space after 10_th iteration. The feasible polytope has been distorted by T, and -DC is also reorganized. Y^{10} is the improved solution at the end of the 10_th iteration.

Figure 7 shows the sequence of Y^i transformed back to the corresponding X^i values in the (normalized) original space. After 10_th iteration, the solutions are $X = (0.032, 0.635, 0.333)$, and $C^T X = 0.063$. They are considerably close to the optimal solutions $X^* = (0.0, 0.667, 0.333)$ and $C^T X^* = 0.0$.



$x^0=(1/3,1/3,1/3)$ $x^1=(.282, .384, .333)$
$x^{10}=(.032, .635, .333)$

Figure 5. Illustration of the Sequentially Improved Solutions X

The optimal solution of the above algorithm is only for (2) not (1). Also it assumes that the starting feasible solution is known.

Two Major Problems in Karmarkar's Algorithm

Feasibility

The initial feasible solution can be found if the size of the matrix A in (2) is small. But in large LP problems, the initial feasible point cannot be found easily. Therefore, the following two-phase problem can be used :

phase 1 : Min $C^T X$

s.t $AX = b$

phase 2 : Min μ

s.t $AX = b + \mu(AX_0 - b)$, where $\mu >= 0$

According to Khachiyan and Karmarkar, it is known that phase 1 has a feasible solution if μ'(min μ) satisfies the condition $\mu' <= 2^{-L}$.

First, by setting μ as x_{n+1} -- the (n+1)th variable of X', phase 2 can be formulated as :

phase 2' Min x_{n+1}

s.t $A'X' = b$, where $A' = [\ A \mid -(AX_0 - b)\ ]$,

and $X' = (\ X\ ,\ x_{n+1}\ )$.

Here, $X = X_0$, x_{n+1} ($=\mu$) = 1 can be taken as a feasible solution in phase 2. By solving the system phase 2' with Karmarkar's algorithm until the condition $\mu' <= 2^{-L}$ is met, the initial feasible solution for phase 1 can be found.

Sliding Objective Function

In the main idea, it is assumed that the value of the objective function at the optimum was zero i.e. $C^T X^* = 0$. Here, Karmarkar's original algorithm is extended for the unknown optimal value of the objective function. The algorithm is summarized as follows :

step 1. Start with l (-2^L) and u (2^L) as the lower and upper bound, respectively.

step 2. Set tentative lower and upper bounds.

$l' = l + 1/3(u - l)$

$u' = u - 1/3(u - l)$

$c' = c - l'e$

step 3. Assume that l' is the minimum value of the objective function. And run the algorithm over c'.

step 4. (a) If the value of the objective function is less than u', then set u = u', and determine new l', u', and c' as in step 2.

(b) If a solution X with $c'X < u$ has not been reached within $n(k + \ln(n))$ steps, then $l' < c'$, and set $l = l'$, reset l', u', and c'.

step 5. Continue optimizing over c'.

In a total of O(nL) times, the optimal objective value is reduced from 2^L to 2^{-L}. Therefore, the complexity for this method is within a polynomial-time bound.

CHAPTER III

SOME MODIFICATIONS

In this chapter, some practical modifications of Karmarkar's main algorithm are considered.

Practical Method for Transforming LP Problem into Karmarkar's Form

The general LP problem is of the form :

$$\begin{aligned} &\text{Min} \quad C^T X && C \in R^n \ , \quad X \in R^n \\ &\text{s.t} \quad a_j X \leq b_j \ , \\ &\qquad a_j X \geq b_j \ , \text{ or} && (6) \\ &\qquad a_j X = b_j \ , \quad \text{where } a_j \in A \ , \ b_j \in b \in R^M \ , \end{aligned}$$

and A is an M x n matrix.

By adding (or subtracting) slack variables ,

$a_j X + s_j = b_j$ and $d_j = 1$ if $a_j X \leq b_j$

$a_j X - s_j = b_j$ and $d_j = -1$ if $a_j X \geq b_j$, where

s_j is a slack variable and d_j is the coefficient of s_j.

By setting $A' = [\ A \mid I'_m\ ]$, $I'_m = \text{diag}(d1,\ldots,dm)$, $X' = (X\ ,\ S) \in R^{n+m}$, $S = \{\ s_j\ \} \in R^m$, and $C'^T = (c_1\ ,\ \ldots\ ,\ c_n, 0, 0, \ldots, 0) \in R^{n+m}$ ($m \leq M$), (6) can be transformed into :

$$\begin{aligned} &\text{Min} \quad C'^T X' \\ &\text{s.t.} \quad A'X' = b \end{aligned} \qquad (7)$$

Now, a projective transformation T such as

$$T_{x'} : R^{n+m} \longrightarrow \Omega^{n+m}, \quad \text{where}$$

$$\Omega^{n+m} = \{ y \in R^{n+m+1} \mid \sum_{k=1}^{n+m+1} y_k = 1 \}$$

has the following properties.

1. If $X^0 = (x^0_1, \ldots, x^0_{n+m})$ is a feasible solution to the above (7), Then

$$T_x(x'_1, \ldots, x'_{n+m}) = \frac{1}{1 + \sum_{k=1}^{n+m} (x'_k / x^0_k)} \left(\frac{x'_1}{x^0_1}, \ldots, \frac{x'_{n+m}}{x^0_{n+m}}, 1 \right)$$

2. The inverse of T is :

$$T_y^{-1}(y_1, \ldots, y_{n+m+1}) = \frac{1}{y_{n+m+1}} (x^0_1 y_1, \ldots, x^0_{n+m} y_{n+m}),$$

where $y_i = (x_i'/x^0_i) / (1 + \sum_{k=1}^{n+m} x_k'/x^0_k)$, and

$$y_{n+m+1} = 1 - \sum_{k=1}^{n+m} y_k \qquad (8)$$

Especially, $T_x(x^0) = (1/n+m+1, \ldots, 1/n+m+1)$ which maps the initial feasible point to the center of the simplex.

By combining (7) and (8), (7) can be rewritten as

$$\text{Min} \quad \sum_{i=1}^{n+m} c_i x^0_i y_i / y_{n+m+1} \qquad (9)$$

$$\text{s.t.} \quad \sum_{i=1}^{n+m} a_{ji} x^0_i y_i / y_{n+m+1} = b_j \qquad (10)$$

$$\sum_{i=1}^{n+m+1} y_i = 1 \qquad (11)$$

If Min $C^T X' = C^*$ for some optimum value C^* in (7), then,

$$\text{Min} \quad C^T X' - C^* = 0$$

$$\Longrightarrow \quad \text{Min} \left(C^T X' - C^* \right) = 0$$

$$\Longrightarrow \quad \text{Min} \left(\sum_{i=1}^{n+m} c_i x^0_i y_i \; / \; y_{n+m+1} - C^* \right) = 0$$

$$\Longrightarrow \quad \text{Min} \; \frac{1}{y_{n+m+1}} \left(\sum_{i=1}^{n+m} c_i x^0_i y_i - C^* y_{n+m+1} \right) = 0$$

$$\Longrightarrow \quad \text{Min} \left(\sum_{i=1}^{n+m} c_i x^0_i y_i - C^* y_{n+m+1} \right) = 0$$

$$\Longrightarrow \quad \text{Min} \sum_{i=1}^{n+m+1} c_i x^0_i y_i = 0 \;, \text{ where } c_{n+m+1} x^0_{n+m+1} = -C^*$$

By the same method, (11) is equivalent to

$$\sum_{i=1}^{n+m} a_{ji} x^0_i y_i - b_j y_{n+m+1} = 0 \quad \text{which implies}$$

$$\sum_{i=1}^{n+m+1} a_{ji} x^0_i y_i = 0 \;, \text{ where } a_{j(n+m+1)} x^0_{n+m+1} = -b_j$$

By letting $C''^T = [\, C'^T \mid C^* \,]$,

$D^0 = \text{diag}\,(x^0_1 \;, \ldots \;, x^0_{n+m} \;, -1)$, and $A'' = [\, A' \mid b \,]$;

finally, the general LP problem (6) is transformed into the following Karmarkar's canonical form :

$$\text{Min} \quad C''^T D^0 Y$$

$$\text{s.t.} \quad A'' D^0 Y = 0$$

$$\sum_{i=1}^{n+m+1} y_i = 1$$

which is equivalent to (2) in Chapter II by setting $C' = C''^T D^0$ and $A' = A'' D^0$ in (2).

Algorithm

step 1 : Add slack variables if necessary in (6).

step 2 : Start with a feasible solution $X^0 \in R^{n+m}$ in (7). The starting feasible solution can be found by using phase II in two-phase problem.

step 3 : $C'^T = C''^T D^0$

$A' = A''D^0$

step 4 : Let $D = \text{diag}(y_1, \ldots, y_{n+m+1})$.

Originally, set $D = (1/n+m+1, \ldots, 1/n+m+1)$ as the center of the simplex Ω^{n+m} .

step 5 : Same as from step 2 to step 5 in Karmarkar's main algorithm. This time, A' is changed to A and C' to C.

step 6 : After exiting from step 5, calculate

$$x_i = \frac{y_i x^0{}_i}{y_{n+m+1}}$$

to find the optimal solution $X \in R^n$ in (6).

Line Search for the Potential Function on the Tranformed Feasible Region

In Karmarkar's main algorithm, a new improved point is found after moving along the negative gradient vector from the center of the simplex.

Unfortunately, the best step size α in Karmarkar's main algorithm is not known. In his theorem, he showed that the potential function --

$f(X) = \sum_{j=1}^{n+1} \ln (C'^{T}X'/x_j)$ -- could be improved by a constant δ, where $\delta = \alpha - \alpha^2/2 - \alpha^2(n+1)/[\, n(1 - \alpha\sqrt{(n+1)/n}\,]$, and $\delta \dashrightarrow \alpha - \alpha^2/2 - \alpha^2/(1-\alpha)$ as $n \dashrightarrow \infty$ (if $\alpha = .25$, then $\delta = 1/8$ as Karmarkar suggested).

But his theorem only shows that the potential function can be decreased at least by a constant. This decrease does not necessarily indicate that the maximum improvement can be gained at each iteration if α is fixed. Instead of fixing α, the whole iteration count can be reduced if the best step size α can be found at each iteration.

Therefore, as suggested by Todd and Burrell [13], a line search is performed for the potential function f' with a negative projected gradient on the transformed space by T. By Rosen's gradient projection method, a new gradient direction c_p can be set as $c_p = [\, I - B^T(BB^T)^{-1}B\,]\,\nabla f'(Y)$ for the potential function f', where $\nabla f'$ is the gradient vector of f', and $Y^0 = (1/n+1 , \ldots , 1/n+1)$ is the center of the simplex Ω^n.

The important fact is that along the line of the gradient vector c_p from Y, f' has one stationary point, which is a minimizer.

(lemma) Let $g(d) = f'(Y + dc_p)$

$$= \sum_{i=1}^{n+1} \ln C^T D(Y + dc_p)/(y_i + dc_p) - \sum_{i=1}^{n+1} \ln x_i',$$

where $d \in H \cap \Omega^n$, and $D = \mathrm{diag}(x_1', \ldots, x_{n+1}')$.

If Y and d are not proportional, then g(d) has at most one stationary point, which is a minimizer.

The above lemma directly follows from Todd and Burrell's lemma in which they follow c_p from a feasible point X' for the potential function f instead of f'.

In the original Karmarkar algorithm, the maximum searching bound is the largest inscribed radius r of the feasible region. Here, instead, the maximum searching point is extended to the boundary of the region $H \cap \Omega^n$ i.e. the facet of the simplex Ω^n. And it is not difficult to find the intersection point where the facet and the gradient vector c_p meet.

For example, if the searching point follows the negative gradient from the center of the simplex, one y_i, which contains the largest positive element a_{max} of c_p, goes to zero first (if more than one y_i have the same maximum positive element a_{max} , then they go to zero together) for a positive constant t which satisfies

$t\, a_{max} - 1/n+1 = 0$.

This means that the searching point is on the facet of the simplex when the above y_i is zero.

Therefore, it is enough to solve the equation

$$Y^0 - t\, c_p = Y_{facet}$$

$$\text{If } t = \frac{y_i{}^0}{a_{max}}$$

$$= \frac{1}{(n+1)a_{max}} \quad , \quad \text{then}$$

$$y_i^{facet} = \frac{1}{n+1} - \frac{1}{(n+1)a_{max}} a_i$$

$$= \frac{1}{n+1} \left(1 - \frac{a_i}{a_{max}} \right)$$

Figure 8 shows the largest inscribed radius r, circumscribed radius R, and the negative gradient vector $- c_p = (-1/3, 0, 1/3)$ which meets the facet of the simplex at the point $Y^{facet} = (0, 1/3, 2/3)$. The maximum searching bound from the center is $Y^0 - Rc_p$ which can have the step size $\alpha = R/r = \sqrt{n/n+1} \,/\sqrt{1/n(n+1)} = n$ if the minimum value is attained on the vertex (In Figure 8, the step size $\alpha = 1.16$.)

In performing a line search for the above extended bound, the Fibonacci algorithm is used -- which is originally used to find the minimum value of a single variable for a nonlinear function. Here, the Fibonacci algorithm is extended to the multi-variable function f by following the given negative gradient vector from the center to the facet of the simplex such that every variable changes proportionally along the search line.

In this Fibonacci algorithm , the accuracy parameter is set to be 0.001 and the final value Y is chosen to be a strictly interior point of the simplex ; thereby Y is in the feasible region.

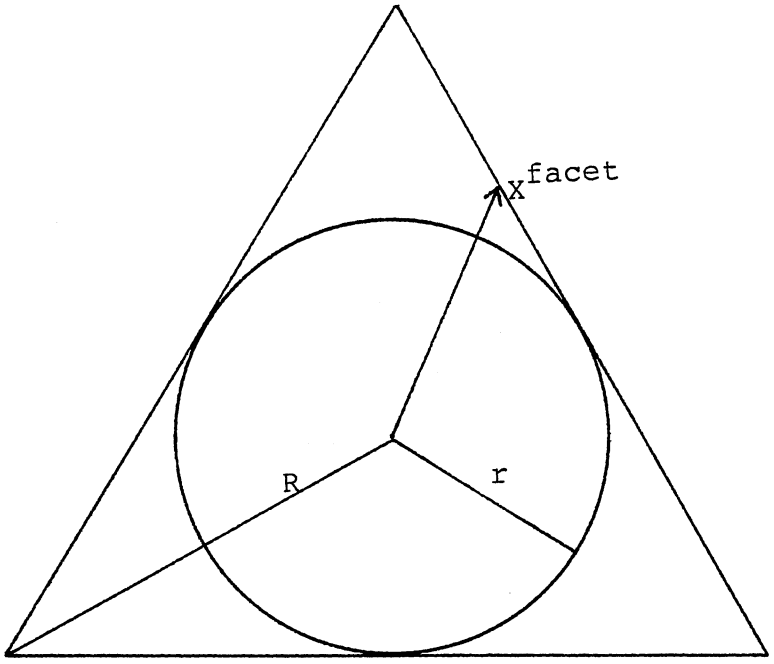


Figure 8. The Extended Bound to the Facet of the Simplex

On the other hand, Kalantari [8] has rather different views for the development of the step size without finding the minimum value of the potential function. Instead, he devised better reductions in the potential function with a suitable step size α, as it appears in the following descriptions.

The relaxation of the original Karmarkar's form is :

$$\begin{aligned} &\text{Min} \quad C^T X && C \in R^{n+1},\ X \in R^{n+1} \\ &\text{s.t.} \quad AX = 0 && A \text{ is an } m \times n+1 \text{ matrix} \\ &\qquad e^T X = n+1 \\ &\qquad x_i >= 0 \end{aligned} \tag{12}$$

Then the potential function for (12) is

$$f^r = n \ln c^T X / c^T e + \sum_{i=1}^{n+1} \ln 1/x_i$$

Kalantari [8] developed the step size α to be $\alpha^* = n / [\ (n-1) + n\beta\ ]$ with $f^r < \ln (\beta + 1/\beta) - 1/\beta$, where $\beta = c^T e / R\,|c_p|$, and $1/n <= \beta <= 1$. Then, because of the monotonicity of α^* in β,

$$1/2 < n / 2n-1 <= \alpha^* <= 1$$

Now, the transformed problem by T is :

$$\begin{aligned} &\text{Min} \quad c^T D X' \\ &\text{s.t.} \quad ADX' = 0 \\ &\qquad e^T X' = n+1 \quad , \quad X' >= 0 \end{aligned} \tag{13}$$

In (13), the step size $\alpha_d = n / [\ (n-1) + n\beta_d\ ]$, where $\beta_d = C^T De / R|c_p|$.

Since (12) and (13) have the same optimal value (for the proof, see Kalantari's lemma), it is enough to

substitute αd for α at each iteration in Karmarkar's main algorithm.

numerical example

Let us consider the following standard LP problem :

$$\begin{aligned} \text{Max} \quad & x_1 + x_2 + x_3 + x_4 \\ \text{s.t.} \quad & x_1 + 2x_2 - x_3 + 3x_4 \leq 12 \\ & x_1 + 3x_2 + x_3 + 2x_4 \leq 8 \\ & 2x_1 - 3x_2 - x_3 + 2x_4 \leq 7 \\ & x_i \geq 0 \end{aligned}$$

Table I shows the different results of the above problem with a starting point $X^0 = (1.5, 1, 1, 1)$. Of course, the maximization problem is changed into the minimization problem by changing C^TX to $-C^TX$, and transformed into Karmarkar's canonical form.

First, $\alpha = 0.25$ is fixed as Karmarkar suggested. The objective function C^TX converges to zero very slowly. After 10 iterations, C^TX has the value less than 1.0 E-2. Next, the line search method is tested with the same starting point. As the table shows, the objective function goes to zero dramatically faster than Karmarkar's. Only after iteration 3, the value of C^TX is less than 1.0 E-2, and after 10_th iteration, it converges to zero within 1.0 E-7. It is noticed that only one α has the value less than 1.0, which simply indicates that the potential function has the minimum value beyond the inscribed sphere for most

TABLE I

THE COMPARISON WITH DIFFERENT STEP SIZES

	Karmarkar		line search	
iter #	objective value	α	objective value	α
1	.396664	.25	.105362	2.03
2	.354513	.25	.033346	1.02
3	.312053	.25	.008762	1.13
4	.270596	.25	.002113	1.16
5	.231554	.25	.000531	1.12
6	.196071	.25	.000139	1.09
7	.164762	.25	.000036	1.09
8	.137706	.25	.000010	1.10
9	.114643	.25	.000002	1.17
10	.095155	.25	.0000003	1.18

	Kalantari	
iter #	objective value	α
1	.309404	.784
2	.183344	.823
3	.094867	.846
4	.046508	.851
5	.022158	.853
6	.010436	.854
7	.004891	.854
8	.002287	.854
9	.001068	.854
10	.000499	.854

iterations.

Final work is with Kalantari's α^* with the same starting solution point. The result shows that his method works reasonably well compared with Karmarkar's, but still shows a slower progress of convergency than the one from the line search method.

The one important and interesting fact is that the above problem shows different optimal solutions for each method (here, "different" means not from the roundoff errors, but from the different searching directions.)

TABLE II

THE DIFFERENT OPTIMAL SOLUTION X*

optimal X	Karmarkar	Modified Line Search	Kalantari
x_1	3.07037	3.74375	3.221551
x_2	0.00017	0.00000	0.000906
x_3	4.92812	4.25625	4.770294
x_4	0.00033	0.00000	0.001812

Table II shows the different optimal solutions X* for the above three methods.

From Table II, it is concluded that each method has different solutions respectively. This means that the above problem has infinite optimal solutions for X because the objective function meets not a vertex but a line or facet of the feasible region.

Actually, if the well-known simplex method is used, the solution will be either (5,0,3,0) or (0,0,8,0), which means the objective function meets the hyperplane between (5,0,3,0) and (0,0,8,0). Also, it shows different optimal solutions if several different starting points are used. This aspect is a good contrast to the simplex methods.

Some Methods for Calculating $(BB^T)^{-1}$

Rank-One Modification

During matrix calculations on each iteration, the matrix

$$(BB^T)^{-1} = \begin{vmatrix} (AD^2A^T)^{-1} & 0 \\ 0 & 1/n+1 \end{vmatrix} \qquad (14)$$

must be updated in Karmarkar's algorithm. The matrix inversion requires on the order of $O(n^3)$ computation in the iteration of both Karmarkar's main algorithm and the simplex method. But in the simplex method, only one column is changed from one iteration to the next -- only a rank-one update is needed. Therefore the order is reduced to $O(n^2)$.

Only the diagonal matrix D in (14) changes from step to step. If only one element of D is changed , then the order of (AD^2A^T) is $O(n^2)$ like that of simplex method; whereas, changing all n diagonal elements of D requires $O(n^3)$. Therefore the following "rank-one" strategy is needed if some elements of D are to be changed.

When the inverse matrix M^{-1} is already obtained, the following Sherman-Morrison formula (or rank-one update) is derived in order to change some elements in M of the form $M + uv^T$ for some vectors u and v.

$$\begin{aligned}(M + uv^T)^{-1} &= (1 + M^{-1}uv^T)^{-1}M^{-1} \\ &= (1 - M^{-1}uv^T + M^{-1}uv^TM^{-1}uv^T \ . \ . \ .)M^{-1} \\ &= M^{-1} - M^{-1}uM^{-1}v^T(1 - \pi + \pi^2 - \ . \ . \ .) \\ &= M^{-1} - \frac{(M^{-1}u)(M^{-1}v^T)}{1 + \pi} \ , \end{aligned} \tag{15}$$

where $\pi = u^TM^{-1}v$.

The whole procedure of (15) requires $3n^2$ computations because it is needed only to calculate $M^{-1}u$, $M^{-1}v^T$, and π -- each requires order of $O(n^2)$.

If (15) is applied to the equation : $AD''^2A^T = (AD'A^T) + (D''_{ii}{}^2 - D'_{ii})a_i a_i^T$, where D' and D" differ only in the i_th entry, and a_i is the i_th column of A, then the following equation which will be used in "rank-one" algorithm is obtained.

$$\begin{aligned}&[AD'^2A^T + da_i a_i^T]^{-1} \\ &= (AD'^2A^T)^{-1} - \frac{d[(AD'^2A^T)^{-1}a_i][(AD'^2A^T)^{-1}a_i]^T}{1 + da_i(AD'^2A^T)^{-1}a_i^T} \ , \end{aligned} \tag{16}$$

where $d = D''_{ii}{}^2 - D'_{ii}{}^2$.

In performing rank-one update, the following two steps are needed instead of just setting $D_{ii}{}^{k+1} = x_i{}^{k+1}$ at k+1_th iteration.

step 1. $D'^{(k+1)} = \sigma^{(k)} D'^{(k)}$, where

$\sigma^{(k)} = 1/n \Sigma_j x_j^{(k+1)}/x_j^{(k)}$

This "appropriately" scales $D'^{(k+1)}$.

step 2. for each i = 1, ... ,n

$$\text{if } \left| \frac{D'_{ii}{}^{(k+1)}}{D_{ii}{}^{(k+1)}} \right| \notin [1/2,2]$$

set $D'_{ii}{}^{(k+1)} = D_{ii}{}^{(k+1)}$, and make rank-one update using the equation (16).

Figure 9 shows the best direction $d(-\nabla f)$, and the modified direction d' after "rank-one" operations.

Karmarkar proved that the order of total number of updating operations N in m steps is $O(m\sqrt{n})$, thereby reducing the order $O(n^3)$ to $O(n^{2.5})$ for calculating BB^T.

Row Partition Scheme

In Karmarkar's algorithm, the time required in performing an iteration is dominated by the calculation of the projective gradient vector $c_p = [\ I - B^T(BB^T)^{-1}B\ ]DC$. The bigger the program size grows, the more computational effort is required in calculating c_p (in fact, the calculation time depends more on the number of constraints than the number of variables.)

Therefore, the computation time can be reduced if some

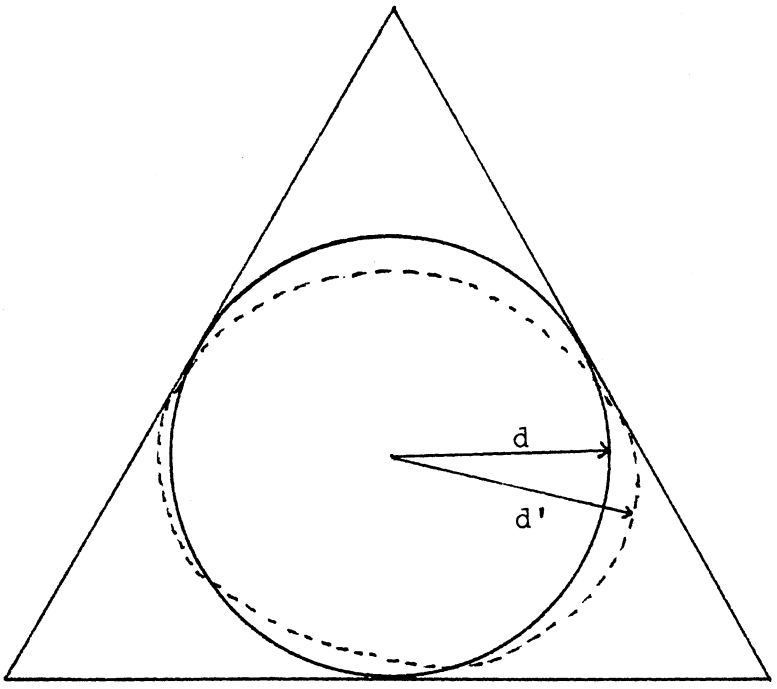


Figure 9. The Best Direction d and the Modified Direction d' After Rank-One Operations

constraints are extracted at each iteration. To do this, Cavalier and Schall [5] proposed the "row partitioning scheme." They divided the constraints into two sets such as $S^- = \{ i \mid s_i^{k+1} < s_i^k \}$ and $S^+ = \{ i \mid s_i^{k+1} >= s_i^k \}$, where s_i^{k+1} and s_i^k are slack variables at the k+1_th and the k_th iteration, respectively. Then, the constraints for S^- are getting more binding at the k+1_th iteration than those at the k_th iteration. Thus, only the constraints for S^- are attempted at the k+1_th iteration instead of the whole set of constraints. This is the main idea of the row partitioning scheme.

But, their method is restricted to the space in which a linearly transformed solution for Karmarkar's algorithm is uniformly rescaled; however, the convergence criterion has not been proved. Also, the whole constraint set has only the inequality form of $AX <= b$. Thus, the following modified algorithm can be easily applied to Karmarkar's original algorithm without any restrictions mentioned above.

1. Let S be the set of the slack variables which are obtained from the transformation scheme described in Chapter III.
2. Run the phase II with the modified line search algorithm. Then the initial feasible solution is obtained.
3. Set $S^- = S$.
4. Run the phase I with S^- .
5. Let $S' = \{ i \mid s_i < 0 \}$ where $i \in S^+$.

6. If $S' = \phi$ then

   $S^+ = S^+ - S'$, $S^- = S^- \cup S'$, goto 4.

   Else goto 7.

7. Calculate S^+ . If $CX' < 1.0E-4$ then stop, where X' is the variables in the canonical form.

8. Calculate X^0 in the original (general) LP form.

9. Let X^0 be the new starting feasible solution for the next iteration, and set $S^- = S - S^+$.

10. Goto 4.

In the above algorithm, since the dimension of X' in Karmarkar's canonical form varies after each iteration, the old canonical form is no longer available in the next iteration. Therefore, the new solution $X' \in R^{n+m+1}$ in the old canonical form is converted to $X \in R^n$ in the original LP problem. Now, a new feasible solution is updated at the beginning of every iteration to get a new canonical form.

CHAPTER IV

NUMERICAL RESULTS

In this chapter, the computational results for Karmarkar's original algorithm and the modified method (a line search for the potential function described in Chapter III) are summarized.

The program, which represents the Karmarkar algorithm and the modified version, is characterized as follows :

- The program is coded in Fortran 77 using the double precision option.
- The main body consists of two parts. The first part is for solving the phase II problem to find the initial feasible solutions; and the second part is for solving the phase I problem to reach the optimal solution.
- It is intended to solve the general LP problem. Therefore, the transformation scheme described in Chapter III (a conversion from a general LP form into the Karmarkar's canonical form) is used.
- For practical purposes, μ = 1.0E-4 in the phase II and cx = 1.0E-4 in the phase I are chosen for the stopping rules.
- Arbitrary LP problems can be solved with no prior information about the optimal value of the objective

function.

- The maximum pivot strategy is used in getting the inverse matrix $(BB^T)^{-1}$ at each iteration.
- The program has an option to follow the original Karmarkar algorithm or the modified algorithm.
- In Karmarkar's original algorithm, the algorithm uses the step size $\alpha = 0.99$ in the direction of the negative gradient. Therefore maximization problems should be changed to minimization problems by negating the objective function.

All LP problems which were used as test problems are in the form (6) in Chapter III. Table III contains the details of the problems.

TABLE III

THE TEST LP PROBLEMS FOR KARMARKAR'S ALGORITHM

problem	rows	columns	slack	total	density
dat1	9	2	9	11	18.1
dat2	6	10	6	16	27.1
dat3	7	12	7	19	23.3
dat4	27	32	19	51	7.4
dat5	42	62	29	91	4.9
dat6	36	33	36	69	5.3
dat7	25	101	17	118	5.2
dat8	56	97	41	138	5.5

"Rows" means the number of general constraints, "columns" represents the number of variables before the transformation into Karmarkar's canonical form, and "total" is equal to "columns" plus "slack," where "slack" is the number of slack variables. Finally, "density" refers to the percentage of non-zero variables in the matrix A.

Karmarkar's and the modified algorithm with the known optimal value C* are compared in Table IV. The number of iterations for the two phases is listed as phase II, phase I, and total iterations. Under each heading, the numbers on both sides of the same column indicate Karmarkar's and the modified algorithm respectively. Here "condition" means the condition number of the matrix BB^T at the final iteration defined as $|B||B^{-}|$, where $|B|$ is the matrix norm in Karmarkar's main algorithm. It is often observed that the number of iterations is excessive in ill-conditioned problems; and "*" signifies that the optimal solutions cannot be reached within 60 iterations.

Since the optimal value C* is generally unknown, the LP problems should be solved without the preinformation of C*. From the practical implementation viewpoint, the "cutting objective method" suggested by Lustig [10] is used instead of the "sliding objective method" which is primarily of theoretical interest. Table V shows the results where no knowledge of final optimal solutions is required.

But one critical problem arises when the "modified line search" algorithm is applied to the "cutting objective

TABLE IV

THE COMPARISON BETWEEN THE KARMARKAR ALGORITHM AND THE MODIFIED LINE SEARCH ALGORITHM WITH KNOWN C*

problem	phase II iter		phase I iter		total iter		cond #
dat1	7	3	8	2	15	5	5.9E3
dat2	1	1	5	3	6	4	3.5E3
dat3	6	2	43	12	49	14	2.8E01
dat4	12	4	39	10	51	14	1.1E11
dat5	10	3	54	10	64	13	2.8E7
dat6	33	11	*	*	*	*	5.4E21
dat7	7	2	40	8	47	10	4.0E5
dat8	25	4	*	20	*	24	7.5E22

TABLE V

THE COMPARISON BETWEEN THE KARMARKAR ALGORITHM AND THE MODIFIED LINE SEARCH ALGORITHM WITH UNKNOWN C*

problem	phase II iter		phase I iter		total iter		simplex
dat1	7	3	9	3	16	6	5
dat2	1	1	6	3	7	4	8
dat3	6	2	42	12	48	14	8
dat4	12	4	32	9	44	13	6
dat5	10	3	42	12	52	15	46
dat6	33	11	*	*	*	*	40
dat7	7	2	34	8	41	10	24
dat8	25	4	*	19	*	23	126

method." After finding the initial feasible solution from phase II, the gradient vector of the "potential function" $f(x_i) = (n+1)\left(c_i x_i / CDY - 1\right)$ at the i_th component should be calculated. But the value of CDY is zero under the "cutting objective method", and it is impossible to find the gradient vector. Therefore, the transformed objective function CDY/eDY is used instead of the "potential function." This substitution works well and is summarized in Table V.

The iteration numbers of phase II in Table V are the same as those of Table IV because the aim of phase II is only to keep μ as small as possible. Also the number of iterations in phase I indicates that there are no major differences between the iterations in Table IV and Table V. In phase II, both algorithms have relatively quick convergence. This is in contrast to phase I where slower convergence is observed in ill-conditioned problems such as "dat6" and "dat8".

Overall, the result shows that the "modified line search algorithm" has a much better convergence behavior than Karmarkar's original algorithm. Even in severely ill-conditioned problems (except "dat6"), the modified version shows a promising result -- whereas Karmarkar's reveals a poor convergence behavior.

Finally, the comparison between the "modified line search algorithm" and the "modified row partition method" is shown in Table VI. Although the "row partition method"

TABLE VI

THE COMPARISON BETWEEN THE MODIFIED LINE SEARCH ALGORITHM AND THE MODIFIED ROW PARTITION METHOD WITH UNKNOWN C*

	line search		row partition		exact
data	loop	optimal solution	loop	optimal solution	solution
dat1	4	-14.2196	3	-14.2199	-14.22
dat2	3	-0.9998	3	-1.0000	-1.0
dat3	12	0.00015	9	0.00000	0.0
dat4	11	-464.7528	11	-464.7530	-464.7531
dat5	14	-4317.996	13	-4317.997	-4318.0
dat6	*	*	*	*	156296.59
dat7	9	-890.996	9	-890.999	-891.0
dat8	19	242595.67	*	*	242594.96

fails to converge to the optimal solution in "dat8", it is not because of the algorithm itself, but because of the highly ill-conditioned problem (Actually the optimal solution was not converged within $C^TX < 1.0E-4$ in the "modified line search method": the lowest value of C^TX was 3.0E-4 at 20_th iteration. After that, the value of C^TX diverges.)

The number of iterations is almost same in both algorithms. The computation time in the "modified row partition method" must be faster than the computational time in the "modified line search algorithm" because some inactive constraints are not used at each iteration. Also, the optimal solution is found more accurately in the "modified row partition method." This may be regarded as another advantage of the row partition algorithm.

CHAPTER V

DUALITY AND ITS APPLICATION TO THE UNKNOWN OPTIMAL SOLUTION C*

The dual linear programming problem is defined directly from the original (primal) linear programming problem because the dual variables are associated with the constraints of the primal LP problem.

The original Karmarkar canonical form is

$$\begin{array}{lll} \text{Min} & C'^{T}X' & C'^{T} \in R^{n+1} \ , \ X' \in R^{n+1} \\ \text{s.t.} & A'X' = 0 & A' \text{ is an } m \times n+1 \text{ matrix} \\ & \sum_{i=1}^{n+1} x_i = 1 & \end{array} \qquad (17)$$

The dual problem for (17) is defined as

$$\begin{array}{ll} \text{Max} & z \\ \text{s.t.} & A'^{T}W + e\,z \leq C' \end{array} \qquad (18)$$

where $W \in R^m$, $z \in R$, $e = (1,1,\ldots,1) \in R^{n+1}$.

Since W is the set of unrestricted variables, w_i ($\in W$) = w_i' - w_i'' is set to be the difference between two non-negative variables w_i' and w_i''. Also z should be changed to - z' to preserve the non-negativity of variables (z always has a non-positive value because the minimum

optimal value of the primal objective function $C'^{T}X'$ is 0 .)

Therefore, (18) can be rewritten as

$$\begin{aligned} &\text{Max} && z \\ &\text{s.t.} && A'^{T}(W' - W'') - e\,z' \;<=\; C' \quad , \text{ where} \\ &W' = \{\, w_i' \mid w_i' >= 0 \,\} \in R^m \ , \text{ and} \\ &W'' = \{\, w_i'' \mid w_i'' >= 0 \,\} \in R^m \ . \end{aligned} \tag{19}$$

Now, (19) is the usual general LP problem (6) before transforming to Karmarkar's canonical form. But this method doubled the basic variables and added m slack variables from its dual form, thus, requiring much more computational effort.

Here, Todd and Burrell's duality algorithm is introduced. Their method does not require the redundant variables, and can be worked easily with the unknown optimal value C'^{*} of $C'^{T}X'$.

First, it is assumed that the optimal value of $C'^{T}X'$ is known i.e. $C'^{*} = 0$. In Todd and Burrell's method, the sequential dual solutions are directly derived from (17) by just setting

$$\begin{aligned} &z = \min \{\, (C' - A'^{T}W)_j \,\} \ , \quad j = 1,2,\ldots,n \\ &W = (A'D^2A'^{T})^{-1}A'D^2C' \end{aligned} \tag{20}$$

Todd and Burrell showed that the potential function could be decreased by a constant in each iteration in Karmarkar's main algorithm. And the above equations for z and W could be obtained as by-products in their proof.

<u>numerical example</u>

$$\text{Min} \quad -1.5\, x_1 - x_2 + 5.75\, x_7$$

$$
\begin{array}{llllllll}
\text{s.t.} & x_1 + x_2 + 2.5x_3 & & & & - 4.5x_7 & = 0 \\
& 2x_1 + x_2 & + x_4 & & & - 7x_7 & = 0 \\
& x_1 & & + 2x_5 & & - 3x_7 & = 0 \\
& x_2 & & & + 2x_6 & - 3x_7 & = 0 \\
& x_1 + x_2 + x_3 & + x_4 & + x_5 & + x_6 & + x_7 & = 1
\end{array}
$$

Then , the dual problem takes the following form :

$$\text{Max} \quad z$$

$$
\begin{array}{llll}
\text{s.t.} & y_1 + 2y_2 + y_3 & + z & \leq -1.5 \\
& y_1 + y_2 + y_4 & + z & \leq -1 \\
& 2.5y_1 + & + z & \leq 0 \\
& 4y_2 & + z & \leq 0 \\
& 2y_3 & + z & \leq 0 \\
& 2y_4 & + z & \leq 0 \\
& -4.5y_1 - 7y_2 - 3y_3 - 3y_4 & + z & \leq 5.75
\end{array}
$$

The iterative solutions with $\alpha = .99$ is shown in Table VII.

TABLE VII

THE ITERATIVE SOLUTIONS FOR THE DUALITY ALGORITHM

iter	cx	z
1	.204867	-.886141
2	.169918	-.412338
3	.085566	-.173773
4	.028854	-.079319
5	.007977	-.064546
6	.002256	-.015960
7	.000636	-.004443
8	.000179	-.001252
9	.000050	-.000353

After iteration 9, the optimal values of the primal and dual variables are

X' = (.3999, .3199, .0000, .0000, .0400, .0800, .1600)

Y = (-.4999, -.5000, .0001, .0000)

From the table, it is confirmed that both C'* of primal and z* of dual converge to the real optimal objective value 0. But this case is only for the known optimal value C'* .

Now, the assumption -- that the optimal value C'* of the objective functon $C'^{T}X'$ is known -- is dropped. Karmarkar originally suggested the "sliding objective method" to solve the case of the unknown optimal value C'* and showed a polynomial-time convergence. But the "sliding objective method" is not very attractive, especially for an implementation on the computer. The initial lower (-2^{L}) and upper (2^{L}) bound for C'* in his method is too low and high, respectively; therefore, many iterations will be required to update the tentative lower bound to isolate C'*.

Todd and Burrell' considered the equation

$$C'^{T}X' - C'^{*} = C'^{T}X' - C'^{*}\sum_{i=1}^{n+1} x_i = \sum_{i=1}^{n+1} (c_i - C'^{*})x_i .$$

Since C'* is not known, they set the lower bound $z(k) < C'^{*}$, and updated $C'^{T}X'$ as $\sum_{i=1}^{n+1} (c_i - z(k))\, x_i$ at the k_th iteration (initially, z(0) could be found by solving the equations in (20). And z(0) is not too low as the initial lower bound -2^{L} in the "sliding objective

method.")

Also, they set $W(k) = (A'D^2A'^T)^{-1}A'D^2(C' - e\,z(k))$, and $z' = \min \{ (C' - A'^TW)_j \}$ for $j = 1,2,\ldots,m$.

This updating method is plausible , since

$\sum_{i=1}^{n+1} (c_i - z(k))x_i$ converges to zero if z(k) converges to real optimum C'* .

There are two cases in determining W(k+1) and z(k+1). When z(k) >= z', the system for dual is not improved at the k_th iteration. Therefore, z(k+1) = z(k) is taken. Also, W(k+1) = W(k) -- since

$\sum_{i=1}^{n+1} (c_i - z(k))x_i <= \sum_{i=1}^{n+1} (c_i - z')x_i$ -- and W(k) has an improved solution.

If z(k) < z' , then obviously an improved dual optimal value of z is obtained. In this case, z(k+1) = z'; but, W(k+1) cannot be taken as W(k) because $\sum_{i=1}^{n+1} (c_i - z')x_i$ is a improved solution for C'^TX' than $\sum_{i=1}^{n+1} (c_i - z(k))x_i$.

Therefore, W(k) has not improved, and a new improved solution W(k+1) is obtained by solving the equation $W(k+1) = (A'D^2A'^T)^{-1}A'D^2(C' - e\,z')$.

Todd and Burrell proved that the above algorithm should generate a sequential set of primal and dual solutions with both the primal objective function C'^TX' and the dual objective function z, converging to the unknown optimum C'*.

More iterations are taken as expected if the same problem with the unknown optimal objective value is run (actually, after 10 iterations both primal and dual objective value have been reduced with 1.0E-3.) But the system works very well and both objective values converge to C'* (=z*) as desired.

After C'* is found, the canonical form of (17) should return to the general LP problem, thereby finding the original optimal solutions C* and X* in (6). Of course, Todd and Burrell described the algorithm which transforms the general LP problem to the canonical form. But their method depends on some upper bounds with the sum of input data. Also, their algorithm requires a redundant constraint, resulting in more computational effort.

Unfortunately, the modified algorithm, which is described in chapter III, cannot be applied to this duality algorithm because the C* in the general LP form must be known in advance. In finding C* in the general LP problem, another method -- which does not require the unnecessary calculations for the dual solutions -- comes from Lustig [10] (And this method is used in chapter IV.) First, he simply sets $C^T X^0$ as the optimal objective value C*, where X^0 is the initial feasible solution coming from phase II problem. In general, $C^T X^k$ is assumed as the unknown C* at the k_th iteration. After running Karmarkar's main algorithm with the updated $C^T X^k$ as c_{n+1} in the canonical form at the k+1_th iteration, $C^T X^{k+1}$, which is closer to C*,

can be calculated.

This method is easy to implement. Also, it has polynomial-time convergence if the initial feasible solution X^0 is near the optimal solution. It is not proved that the above method has polynomial-time convergence if the starting point is not close to the optimum (Lustig stated that it may be polynomial), but it works really well with some testing LP problems as already shown in chapter IV.

CHAPTER VI

SUMMARY, CONCLUSION, AND SUGGESTIONS FOR FUTURE WORK

Summary and Conclusion

In this study, Karmarkar's new polynomial-time algorithm is introduced. His algorithm runs in polynomial-time whereas the simplex method requires exponential-time for its iterative procedures on the worst-case problems.

Karmarkar's main algorithm starts from the canonical form of

$$\text{Min} \quad CX$$

$$\text{s.t.} \quad AX = 0$$

$$\sum_{i=1}^{n+1} x_i = 1$$

Although Todd and Burrell [13] devised an algorithm -- which transformed the general LP form into the above canonical form -- this method requires additional computational effort due to unnecessary constraints and variables. Since the exact transformation was not explained clearly in Karmarkar's original paper, a conversion method, which is based on Karmarkar's suggestion, is introduced in this thesis. This transformation algorithm is totally

different from Todd and Burrell's method [13], but clarifies Karmarkar's indirect suggestion for the above transformation.

A line search, which extends the range of search to the facet of the simplex, has been tested as a modification of Karmarkar's main algorithm. The modified algorithm has been coded in Fortran 77 and tested for eight LP problems. The program accepts the general LP form and converts it into the Karmarkar's canonical form. An initial feasible solution is generated in phase II and used to get the optimal solution in phase I. The program can handle the unknown optimal value C* by using the "cutting objective method."

The computational results show that the modified algorithms require fewer iterations and give faster convergence to the optimal solution than Karmarkar's original method. In degenerate problems such as "dat6", however, the inverse matrix of (BB^T) is severely ill-conditioned and the optimal solution cannot be found within low iteration counts. In this case, it is recommended to replace the maximum pivot strategy with other methods such as Cholesky factorization [5], QR factorization [13], or another least square method [10] for calculating $(BB^T)^{-1}$.

Finally, Todd and Burrell's duality for Karmarkar's canonical form is discussed. Both theoretical and computational results show that the dual variables can be generated. It also shows that their method can be applied

to the case of unknown C*.

Suggestions for Future Work

Karmarkar's new polynomial-time algorithm can be improved in many ways. The following suggested improvements for the Karmarkar algorithm should be achieved in the near future.

1. A good initial feasible solution for phase II in Karmarkar's main algorithm should be given for a fast convergence to the optimal solution. Until now, the starting point only depends on the phase II problem, and it is not known whether the starting feasible solution is a favorable one for the given system. If the initial point lies near the optimal solution, the optimum will be found with a few iterations and may not yield the large condition number for the highly ill-conditioned matrix (BB^T).
2. The best step size α is not known. Some recent research shows that the results depend upon the step size α. This implies that the step size of 0.25 in Karmarkar's original paper might be improved. Also, faster convergence occurs as the step size α increases to 1 for randomly chosen test programs. But the mathematical proof is not provided by anyone at the present time.
3. The most critical weakness in Karmarkar's algorithm is that it does not yield to postoptimal analysis. There should be more work in the area of postoptimal analysis.
4. Although the dual variables can be generated with Todd

and Burrell's duality algorithm, their method is only applied to the canonical form. In order to improve the duality theory, a direct relationship between primal and dual in the general LP form should be developed.

5. To be widely applied, Karmarkar's algorithm should be modified to handle the degeneracy problems. Cholesky factorization, least squares method, and QR factorization have been tried to avoid the matrix (BB^T), being severely ill-conditioned. Applying the above methods is another area of further work.

6. There may be possible extensions to integer programming, branch and bound problem, multi-objective functions, and even non-linear objective functions.

REFERENCES

[1] Anstreicher, Kurt M., "Analysis of a modified Karmarkar Algorithm for Linear Programming," Yale School of Organization and Management, Yale University, August 1985.

[2] Cavalier, T.M., and Soyster, A.L., "Some Computational Experience and a Modification of the Karmarkar Algorithm," Department of Industrial and Management System Engineering, The Pennsylvania State University, February 1985.

[3] Cavalier, T.M., and Schall, K.C., "Implementing a Projective Algorithm for Solving Inequality-Constrained Linear Programs," IMSE Working Paper 86-128, The Pennsylvania State University, 1986.

[4] Dantzig, George B., Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey, 1963.

[5] Gill, Phillip E.; Murray, Walter; Saunders, Michael A.; Tomlin, J.A.; and Wright, Margaret H., "On projected Newton Barrier Methods for Linear Programming and an Equivalence to Karmarkar's Projective Method," Technical Report Sol 85-11, Systems Optimization Laboratory, Department of Operations Research, Stanford University, July 1985.

[6] Hooker, J.N., "Karmarkar's Linear Programming Algorithm" Interfaces 16: 4 July-August p 75-90, 1986.

[7] Kalantari, B., "Karmarkar's Algorithm with Improved Steps," Technical Report LCSR-TR-80, Department of Computer Science, Rutgers University, 1986.

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

[9] Khachiyan, L.G., "A Polynomial Algorithm in Linear Programming," Doklady Akademiia Nauk SSSR 244:S, p 1093-1096, 1979, translated in Soviet

Mathematics Doklady 20:1, p 191-194, 1979.

[10] Kozlov, Alex., "The Karmarkar Algorithm: Is It for Real?" Siamnews, vol. 18, No. 6, Nov. 1985.

[11] Lustig, Irvin. J., "A Practical Approach to Karmarkar's Algorithm," Department of Operations Research, Stanford University, June 1985.

[12] Rosen, J.B., "The Gradient Projection Method for Non-linear Programming," SIAM J. Appli. Math., p 181-217, vol. 8, No. 1, March 1960.

[13] Todd, Michael J., and Burrell, Bruce P., "An Extension of Karmarkar's Algorithm for Linear Programming Using Dual Variables," Technical Report No. 648, School of Operational Research and Industrial Engineering, College of Engineering, Cornell University, January 1985.

[14] Tomlin, J.A., "An Experimental Approach to Karmarkar's Projective Method for Linear Programming," Ketron, Inc., Mountain View, California, 1985.

[15] Vanderbei, Robert J., Meketon, Marc S., and Freedman, Barry A., "A Modification of Karmarkar's Linear Programming Algorithm," AT&T Bell Laboratories, Holmdel, New Jersey, 1985.

APPENDIXES

APPENDIX A

THE PROGRAM LISTING OF KARMARKAR'S ALGORITHM AND THE MODIFIED ALGORITHM

```
C*************************************************************
C                                                            *
C    KARMARKAR'S ALGORITHM FOR LP PROBLEMS                   *
C                                                            *
C*************************************************************
C
C    PURPOSE
C    -------
C
C         THIS PROCEDURE FINDS THE OPTIMAL SOLUTIONS FOR LP
C    PROBLEMS OF THE FORM :
C
C    MIN    C(1)X(1) + ... + C(N)X(N)
C
C    SUBJECT TO  A(I,1)X(1) + ... + A(I,N)X(N)<=,=,>=B(I)
C
C                I = 1, ... , M     X(I) >= 0
C
C
C    METHOD
C    ------
C
C         THIS PROGRAM IS BASED ON KARMARKAR'S MAIN
C    ALGORITHM AND THE MODIFIED ALGORITHM FOR THE UNKNOWN
C    OBJECTIVE VALUE C*.
C         THE METHOD IS AN ITERATIVE TECHNIQUE  WHICH
C    CONSISTS OF TWO PHASES.  IN THE FIRST PHASE, A POSITIVE
C    INITIAL FEASIBLE SOLUTION WILL BE OBTAINED.
C         THE GLOBAL OPTIMAL SOLUTION WILL BE FOUND IN THE
C    NEXT PHASE.
C
C    1.     AFTER ENTERING THE VALUES OF C, A, AND B, THEN
C       THE SLACK VARIABLES ARE GENERATED AUTOMATICALLY.
C
C    2.     THE PROBLEM IS CHANGED INTO KARMARKAR'S
C       CANONICAL FORM.
C
C    3.     THE INITIAL FEASIBLE SOLUTION IS OBTAINED BY
C       SOLVING THE PHASE II PROBLEM.
C
C    3.     WITH THE INTIAL FEASIBLE SOLUTION, THE OPTIMAL
C       SOLUTION IS SEARCHED IN EACH ITERATION WITH THE
C       PHASE I PROBLEM.
C
C    4.     AFTER FINDING THE OPTIMAL SOLUTION IN
C       KARMARKAR'S CANONICAL FORM , THE ORIGINAL SOLUTION
C       FOR X IS FOUND.
C
```

```
C     USAGE
C     -----
C
C          THE PROGRAM CONSISTS OF A MAIN PROGRAM AND TWO
C     SUBROUTINES, KARMAR AND INVERSE.  THE MAIN PROGRAM
C     ESTABLISHES THE INTERACTIVE PART, THE GENERATED SLACK
C     VARIABLES, AND THE CONVERSION TO THE PHASE II PROBLEM.
C          SUBROUTINE KARMAR CONSISTS OF KARMARKAR'S MAIN
C     ALGORITHM AND THE MODIFIED ALGORITHM.  SUBROUTINE
C     INVERSE IS USED TO FIND THE INVERSE OF A GIVEN MATRIX.
C          THE MAXIMUM DIMENSION FOR A IS 200 BY 200.   IF
C     THE INPUT IS REQUIRED MORE THAN THIS MAXIMUM DIMENSION,
C     THE DIMENSION STATEMENT IN THE MAIN PROGRAM SHOULD BE
C     MODIFIED PROPERLY.
C
C
C     INPUT DATA
C     ----------
C
C          FIRST, THE VALUES OF THE NUMBER OF VARIABLES
C     AND CONSTRAINTS ARE ENTERED. NEXT INPUT DATA CONSISTS
C     OF C, OBJ_C, A, COND, AND B. EACH ROW HAS THE MAXIMUM
C     10 ELEMENTS OF C AND A. AFTER ENTERING THE LAST INPUT
C     OF C, THE INPUT FOR OBJ_C IS ENTERED. AFTER FINISHING
C     ENTERING THE I_TH INPUT OF A, THE CONDITION FOR THE
C     I_TH CONSTRAINT ('<=','>=',OR'==') IS ENTERED. AND
C     THE INPUT FOR B(I) IS ENTERED. ALL INPUT DATA USE FREE
C     FORMAT.
C
C     TABLEAU
C     -------
C
C     N   M
C     C(1)     C(2)  .  .  .  C(10)
C     .  .  .  .  C(N)   OBJ_C
C     A(1,1)   A(1,2) .  .  . A(1,10)
C     .  .  .  .  A(1,N)  COND  B(1)
C     .  .  .  .  .
C     .  .  .  .  .
C     A(M,1)   A(M,2) .  .  . A(M,10)
C     .  .  .  .  A(M,N)  COND  B(M)
C
C
C     OUTPUT
C     ------
C
C          THE INITIAL FEASIBLE SOLUTION IS PRINTED.  THE
C     IMPROVED SOLUTION PER EACH ITERATION IS SHOWN UNTIL
C     FINDING THE OPTIMAL SOLUTION.  THE FINAL OPTIMAL
C     SOLUTION IS PRINTED WITH THE NUMBER OF ITERATIONS
C     IN KARMARKAR'S CANONICAL FORM.  FINALLY, THE OPTIMAL
C     SOLUTION FOR THE ORIGINAL LP PROBLEM IS PRINTED BY
C     CONVERTING FROM KARMARKAR'S CANONICAL FORM INTO THE
```

```
C     GIVEN GENERAL LP PROBLEM.
C
C
C     DESCRIPTION OF MAJOR PARAMETERS
C     -------------------------------
C
C     N         THE NUMBER OF VARIABLES X(I)
C     N1        THE NUMBER OF THE TOTAL VARIABLES ( BASIC +
C               SLACK )
C     M         THE NUMBER OF CONSTRAINTS
C     XXN       THE REAL VALUE OF N1+1
C     R         RADIUS OF THE LARGEST INSCRIBED
C               SPHERE OF THE SIMPLEX
C     ALPHA     STEP SIZE
C     LINE      BIT FOR THE ALGORITHM.  IF LINE = 0 THEN
C               KARMARKAR'S ORIGINAL ALGORITHM WORKS, ELSE THE
C               MODIFIED ALGORITHM WORKS.
C     IFLAG     BIT FOR PHASE I AND PHASE II PROBLEM.  IF
C               IFLAG = 1, THEN SUBROUTINE KARMAR WORKS FOR
C               THE PHASE II, ELSE FOR THE PHASE I.
C
C
C     DESCRIPTION OF MAJOR VARIABLES
C     ------------------------------
C
C     STARTX(I)       THE INITIAL FEASIBLE SOLUTION
C     RES(I),RES1(I)  RESOURCE VECTOR b
C     A(J,I)          INPUT MATRIX A
C     ORIGIN(I)       THE SOLUTION OF THE ORIGINAL LP
C                     PROBLEM
C     CC(I)           COST VECTOR C
C     OBJ_C           KNOWN OPTIMAL OBJECTIVE FUNCTION
C     COND(I)         CONDITION FOR THE CONSTRAINTS
C                     '<=' , '>=' , OR '=='
C     NEWX(I)         NEW IMPROVED SOLUTION PER ITERATION
C     CC1(I)          MULTIPLICATION OF CC(I) AND STARTX(I)
C     A1(J,I)         MULTIPLICATION OF A(J,I) AND STARTX(I)
C     B(J,I)          MULTIPLICATION OF A1(J,I) AND NEWX(I)
C     BB(I,J)         MATRIX OF (ADDA) IN KARMARKAR'S MAIN
C                     ALGORITHM
C     DC(I)           MULTIPLICATION OF CC(I) AND NEWX(I)
C     CP1(I,J)        MATRIX OF B(BB)
C     CP(I,J)         MATRIX OF I -B(BB)B
C     GRAD(I)         GRADIENT VECTOR OF [ I - B(BB)B ]DC
C     Y(I)            NEW IMPROVED POINT IN THE TRANSFORMED
C                     SPACE
C*************************************************************

      double precision startx(200),res1(200),res(200),
     *                 a(200,200),origin(200),cc(200),obj_c
      character cond(200)*2
```

```
      print *, '==> Enter the number of variables and
     *          constraints'
      read *, n,m
      n1 = n

      j=1
      print *, 'MIN :'

      do while ( j .le. n )
            if ( j+9 .le. n ) then
              read *, ( cc(i), i=j,j+9 )
            else
              read *, ( cc(i), i=j,n),obj_c
            endif
            j=j+10
      enddo

      i=1
      print *, 'SUBJECT TO :'
      do while ( i .le. m )
            k=1
            do while ( k .le. n )
              if ( k+9 .le. n ) then
                read *, ( a(i,j), j=k,k+9 )
              else
                read *, ( a(i,j), j=k,n),cond(i),res(i)
              endif
              k=k+10
            enddo
            if ( cond(i) .eq. '>=' ) then
              n1=n1+1
              a(i,n1)= -1.
            else if ( cond(i) .eq. '<=' ) then
              n1=n1+1
              a(i,n1) = 1.
            endif
            i=i+1
            if ( i .le. m ) then
              print *, '==> Enter the next row.'
            endif
      enddo
      print *, 'Want line search? ---(1(yes)/0(no))'
      read *, line

      n0=n
      n=n1
      do 20 i=1,n+1
            startx(i) = 1.
   20 continue
      startx(n+1) = 1.
```

```
C.......SET UP THE PHASE II PROBLEM.

      do 21 j=1,m
         res1(j) = 0.
         do 22 i=1,n
            res1(j) = res1(j) + a(j,i)
   22    continue

         res1(j) = res1(j)
         res1(j) = res1(j) - res(j)
         a(j,n+1) = -res1(j)

   21 continue

C.......FIND THE INITIAL FEASIBLE SOLUTION

      call karmar(n+1,n0,m,a,startx,res,cc,0.,1,line)

C.......FIND THE INITIAL OPTIMAL SOLUTION

            obj_c = 0.
            do 30 i=1,n
                obj_c=obj_c + cc(i)*startx(i)
   30       continue

C.......MAIN ALGORITHM

      call karmar(n,n1,m,a,startx,res,cc,obj_c,0,line)

      stop

      end


C*************************************************************
C                                                            *
C          THIS SUBROUTINE SOLVES KARMARKAR'S MAIN           *
C                        ALGORITHM.                          *
C                                                            *
C*************************************************************

      subroutine
     * karmar(n,n1,m,a,startx,res,cc,obj_c,iflag,line)

      double precision startx(200),res(200),cc(200),obj_c,
     *       xxn,grad1,newx(200),bb(200,200),dc(200),
     *       b(200,200),cp(200,200)
      double precision grad(200),y(200),alpha,r,edy,
     *       origin(200) cc1(200),a1(200,200),a(200,200),
     *       cp1(200,200) f,accu,fib(200),center(200),
     *       bound(200),b1(200),all(200),w(200),f1,f2,v(200)
      double precision length(200),norm,norm1,amax,
     *       s(200,200),ss(200,200),s1
```

```
      xxn = n + 1.

      grad1 = xxn * ( xxn - 1. )
      r = 1. / sqrt(grad1)

C.....TRANSFORMATION INTO KARMARKAR'S CANONICAL FORM

      do 100 i=1,n+1
            newx(i) = 1./xxn
            cc1(i) = cc(i)*startx(i)
  100   continue

            cc1(n+1) = -obj_c

      do 200 i=1,m
            do 210 j=1,n
              a1(i,j) = a(i,j)*startx(j)
  210       continue
            a1(i,n+1) = -res(i)
  200 continue

C.....INITIALIZATION OF BB

      do 250 i=1,m
            bb(i,m+1) = 0.
            bb(m+1,i) = 0.
  250 continue

      bb(m+1,m+1) = 1. / xxn

C.....SET UP THE MAXIMUM FIBONACCI NUMBER

      if ( line .eq. 1 ) then
      accu = .001
      fib(1) = 1.0
      fib(2) = 2.0

      jj = 3
      fib1 = 0.
      do while (fib1 .lt. 1./accu)
            fib(jj) = fib(jj-1) + fib(jj-2)
            fib1 = fib(jj)
            jj = jj+1
      enddo

      jj = jj - 1
      kk = jj - 2
      ik = jj - 2
      jjj = jj

      endif
```

```
C.......MAIN LOOP

      do 9000 loop=1,500

C........  INITIALIZATION OF B(M+1)

         do 300 i=1,n+1
            b(m+1,i) = 1.
  300    continue

C........  CALCULATION OF DC

         if ( iflag .eq. 0 ) then
               do 310 i=1,n+1
                  dc(i) = cc1(i)*newx(i)
  310          continue
         else
               do 320 i=1,n+1
                  dc(i) = 0.
  320          continue
               dc(n) = newx(n)
         endif

C........  CALCULATION OF BB

         do 400 j=1,m
            do 410 i=1,n+1
              b(j,i) = a1(j,i)*newx(i)
  410       continue
  400    continue

         do 500 j=1,m
              do 510 i=1,m
                 bb(j,i) = 0.
                   do 520 k=1,n+1
                     if (b(j,k) .ne. 0. .and. b(i,k)
     *                    .ne. 0.) then
                         bb(j,i) = bb(j,i)+b(j,k)*b(i,k)
                     endif
  520              continue
  510         continue
  500    continue

C........  FIND CONDITION NUMBER

         bb(m+1,m+1) = xxn
         norm = 0.
         do 550 i=1,m+1
            do 560 j=1,m+1
               norm = norm + bb(i,j)*bb(i,j)
  560       continue
  550    continue
         norm = sqrt(norm)
```

```
C........  FIND INVERSE MATRIX OF BB BY CALLING INVERSE

          call inverse(bb,m)

C........  CALCULATION OF THE CONDITION NUMBER

          bb(m+1,m+1) = 1./xxn
          norm1 = 0.
          do 570 i=1,m+1
             do 580 j=1,m+1
                norm1 = norm1 + bb(i,j)*bb(i,j)
  580        continue
  570     continue

          norm1 = sqrt(norm1)
          norm = norm * norm1

C........  CALCULATION OF PROJECTED GRADIENT

          do 600 j=1,n+1
             do 610 i=1,m+1
               cp1(j,i) = 0.
               do 620 k=1,m+1
                  if (b(k,j) .ne. 0.) then
                    cp1(j,i) = cp1(j,i)+b(k,j)*bb(k,i)
                  endif
  620           continue
  610        continue
  600     continue

          do 700 j=1,n+1
             do 710 i=1,n+1
               cp(j,i) = 0.
               do 720 k=1,m+1
                  if (b(k,i) .ne. 0.) then
                    cp(j,i) = cp(j,i)+cp1(j,k)*b(k,i)
                  endif
  720           continue
  710        continue
  700     continue

          do 800 i=1,n+1
             do 810 j=1,n+1
               cp(i,j) = -cp(i,j)
  810        continue
  800     continue

          do 820 i=1, n+1
             cp(i,i) = cp(i,i) + 1.
  820     continue

C........  CALCULATION OF GRADIENT VECTOR
             do 920 i=1,n+1
```

```
                grad(i) = 0.
              do 930 j=1,n+1
                grad(i) = grad(i) + cp(i,j)*dc(j)
  930         continue
  920       continue

C........  CALCULATION OF THE NORM OF GRADIENT VECTOR

          grad1 = 0.
          do 1000 i=1,n+1
             grad1 = grad1 + grad(i)*grad(i)
 1000     continue
          grad1 = sqrt(grad1)

C........  CALCULATION OF Cp = Cp / |Cp|

          do 1010 i=1,n+1
             grad(i) = grad(i) / grad1
 1010     continue

C........  LINE SEARCH FOR THE POTENTIAL FUNCTION

          if ( line .eq. 1 ) then

C........  SEARCH FOR THE LARGEST ELEMENT OF Cp

              grad1 = grad(1)
              do 1020 i=1,n+1
                if (grad1 .lt. grad(i)) then
                  grad1 = grad(i)
                endif
 1020         continue

C........  MAXIMUM SEARCHING BOUNDS

              do 2100 i=1, n+1
                center(i) = 1./xxn
                bound(i) = (1.-grad(i)/grad1)/xxn
                length(i) = (bound(i) - center(i))
 2100         continue

C........  SEARCH FOR THE MINIMUM VALUE OF THE GIVEN
C........  FUNCTION

              jj = jjj
              kk = jjj - 2
              ik = jjj - 2
              do 2200 i=1,n+1
                bl(i) = length(i)
                all(i) = fib(ik) * bl(i) / fib(jj)
                w(i) = center(i) + all(i)
                v(i) = bound(i) - all(i)
 2200         continue
```

```
                call f(w,f1,dc,newx,n+1)
                call f(v,f2,dc,newx,n+1)
                ik = ik - 1
                jj = jj - 1

                do 2300 iter=2,kk+1
                  if ( f2 .le. f1 ) then
                      do 2310 i=1,n+1
                        center(i) = center(i) + all(i)
                        bl(i) = bound(i) - center(i)
                        w(i) = v(i)
                        all(i) = fib(ik) * bl(i) / fib(jj)
                        v(i) = bound(i) - all(i)
 2310                 continue
                      call f(w,f1,dc,newx,n+1)
                      call f(v,f2,dc,newx,n+1)

                  else
                      do 2320 i=1,n+1
                        bound(i) = bound(i) - all(i)
                        bl(i) = bound(i) - center(i)
                        v(i) = w(i)
                        all(i) = fib(ik) * bl(i) / fib(jj)
                        w(i) = center(i) + all(i)
 2320                 continue
                      call f(w,f1,dc,newx,n+1)
                      call f(v,f2,dc,newx,n+1)
                  endif
                  ik = ik - 1
                  jj = jj - 1
                  if ( ik .lt. 1 ) then
                    ik = 1
                  endif
 2300           continue

C.....  THE FINAL MINIMUM VALUE OF THE GIVEN FUNCTION

                do 2400 i=1,n+1
                  y(i) = w(i)
 2400           continue

          endif

C.....  END OF LINE SEARCH  .....

C.....  CALCULATION OF THE IMPROVED POINT

          if ( line .eq. 0 ) then
             do 1100 i=1,n+1
               y(i) = 1/xxn - alpha * r * grad(i)
 1100        continue
          endif
```

```
C........  FIND THE NEW SOLUTION BY USING INVERSE
C........  TRANSFORMATION

          edy = 0.
          do 1200 i=1,n+1
             edy = edy + newx(i) * y(i)
 1200     continue
          do 1300 i=1,n+1
             newx(i) = newx(i) * y(i) / edy
 1300     continue

C........  CHECK STOPPING RULE

          if ( iflag .eq. 0 ) then
               cx = 0.
               do 1400 i=1,n+1
                 cx = cx + cc1(i) * newx(i)
                 origin(i) = startx(i) * newx(i)/newx(n+1)
 1400          continue
               print *, loop,cx,'condition # = ',norm

C........  FIND THE NEXT OPTIMAL OBJECTIVE VALUE IF C* IS
C........  UNKNOWN

C........  PHASE I PROBLEM

               cc1(n+1) = 0.
               do 1430 i=1,n
                 cc1(n+1)=cc1(n+1)-cc(i)*origin(i)
 1430          continue
               print *, '  The value of CX = ',-cc1(n+1)
               write(3,1410) loop,cx,-cc1(n+1),norm
 1410          format('  loop = ',i3,'   cx = ',f13.6,'
     *                 CX = ',f13.5,'  cond # = ',e20.10)
               write(3,1420)
 1420          format(' ------------------------------------
     *               '-------------------------')

C........  STOPPING RULE FOR THE PHASE I PROBLEM

             if (abs(cx) .lt. 1.0E-4) then
                write(3,1450)
 1450           format(//' ***** The optimal solution ***')
                obj_c = 0.
                do 1700 i=1,n
                  obj_c=obj_c + cc(i)*origin(i)
 1700           continue
                write(3,1800) obj_c
 1800           format(//' The final optimum = ',f13.6)
                do 1810 i=1,n
                origin(i) = startx(i) * newx(i)/newx(n+1)
                write(3,1820) i, origin(i)
```

```
 1820           format(/'  x(',i3,') = ',f15.5)
 1810           continue
                goto 3000
              endif

C........  PHASE II PROBLEM

             else
               origin(n) = startx(n) * newx(n)/newx(n+1)
               if (origin(n) .lt. 1.0E-4) then
               goto 4000
             endif
        print *, '  loop = ',loop,origin(n),'cond # = ',norm
        write(3,9001) loop,origin(n),norm
 9001   format(/'  loop = ',i3,'  origin = ',f10.7,
     *          '  cond # = ',e20.10)
        endif

 9000   continue

C........  END THE MAIN LOOP ........

C........  THE INITIAL FEASIBLE SOLUTION

 4000   print *, '  loop = ',loop,origin(n),'cond # = ',norm
        write(3,9001) loop,origin(n),norm
        do 1500 i=1,n-1
            startx(i) = startx(i) * newx(i)/newx(n+1)
            feasi = feasi + cc(i)*startx(i)
 1500   continue
        print *, 'feasible value CX = ',feasi
        write(3,1550) feasi
 1550   format(/'  Initial value of CX = ',f15.5)
        write(3,1750)
 1750   format(//)

 3000   return

        end

C*************************************************************
C                                                            *
C     SUBROUTINE INVERSE :                                   *
C                                                            *
C          THIS SUBROUTINE CALCULATES THE INVERSE OF         *
C     A GIVEN MATRIX.                                        *
C                                                            *
C*************************************************************

      subroutine inverse(bb,m)

      double precision bb(200,200),t,tt,pivot(200)
      integer ipvot(200),index(200,2)
```

```
   do 50 j=1,m
      ipvot(j) = 0
50 continue
   do 55 i=1,m
      t = 0.
      do 56 j=1,m
         if ( ipvot(j) .ne. 1 ) then
           do 57 k=1,m
              if ( ipvot(k) .eq. 0 ) then
                if ( t .lt. abs(bb(j,k)) ) then
                   irow = j
                   icol = k
                   t = abs(bb(j,k))
                endif
              else if ( ipvot(k) .gt. 1 ) then
                print *, 'singular matrix'
                return
              endif
57         continue
         endif
56    continue

      ipvot(icol) = ipvot(icol) + 1

      if ( irow .ne. icol ) then
         do 60 l=1,m
           tt = bb(irow,l)
           bb(irow,l) = bb(icol,l)
           bb(icol,l) = tt
60       continue
      endif

      index(i,1) = irow
      index(i,2) = icol

      if ( bb(icol,icol) .eq. 0. ) then
         print *, 'singular matrix'
         return
      endif

      pivot(i) = 1. / bb(icol,icol)
      bb(icol,icol) = 1.
      do 65 l=1,m
         bb(icol,l) = bb(icol,l) * pivot(i)
65    continue
      do 70 ll=1,m
         if ( ll .ne. icol ) then
           tt = bb(ll,icol)
           bb(ll,icol) = 0.
           do 75 l=1,m
             bb(ll,l) = bb(ll,l) - bb(icol,l)*tt
75         continue
         endif
70    continue
```

```
   55 continue
      do 80 l=m,1,-1
            if ( index(l,1) .ne. index(l,2) ) then
              jrow = index(l,1)
              jcol = index(l,2)
              do 85 k=1,m
                tt = bb(k,jrow)
                bb(k,jrow) = bb(k,jcol)
                bb(k,jcol) = tt
   85         continue
            endif
   80 continue

      return

      end


C*************************************************************
C                                                            *
C          THIS SUBROUTINE IS TO CALCULATE THE               *
C     MINIMUM OF THE TRANSFORMED OBJECTIVE FUNCTION          *
C                                                            *
C*************************************************************

      subroutine f(xx,ff,dc,newx,nn)

      double precision xx(200),dc(200),newx(200),ff

      ff = 0.

      do 5000 i=1,nn
            ff = ff + dc(i)*xx(i)
 5000 continue

      return

      end
```

APPENDIX B

THE OUTPUTS FOR THE MODIFIED ALGORITHM

DAT 1

Phase II

loop	μ	cond #
1	0.9931786	3.1E06
2	0.0258428	3.2E05
3	0.0000244	1.5E04

Initial Objective Value = -4.43341

Phase I

loop	cx	CX	cond #
1	0.431350	-14.03786	7.0E02
2	0.008450	-14.21196	5.9E03
3	0.000358	-14.21934	5.9E03
4	0.000015	-14.21965	6.0E03

DAT 2

Phase II

loop	μ	cond #
1	0.0005740	5.7E03
2	0.0000003	4.8E03

Initial Objective Value = 1.00000

Phase I

loop	cx	CX	cond #
1	0.140280	-0.60111	4.4E03
2	0.033875	-0.99878	3.7E03
3	0.000091	-0.99984	3.5E03

DAT 3

Phase II

loop	μ	cond #
1	0.0016027	1.1E05
2	0.0000001	8.6E03

The Initial Feasible Solution = 145646.27381

Phase I

loop	cx	CX	cond #
1	2009.378784	100814.02887	1.7E02
2	1925.007202	49723.12023	1.0E02
3	1309.447998	9015.80799	6.1E01
4	234.125839	1588.45596	3.2E01
5	46.895611	94.81730	2.8E01
6	2.481863	15.74895	2.8E01
7	0.425820	2.18240	2.8E01
8	0.053919	0.46454	2.8E01
9	0.012593	0.06331	2.8E01
10	0.001685	0.00962	2.8E01
11	0.000274	0.00088	2.8E01
12	0.000022	0.00017	2.8E01

DAT 4

Phase II

loop	μ	cond #
1	0.9933052	5.9E06
2	0.8461444	3.2E06
3	0.0026256	3.1E06
4	0.0000016	5.8E06

Initial Objective Value = -64.53925

Phase I

loop	cx	CX	cond #
1	1.919885	-161.88980	1.3E04
2	3.319170	-321.03095	1.5E04
3	2.633654	-425.28271	4.1E04
4	0.773621	-452.62355	4.6E05
5	0.161926	-458.19565	1.1E07
6	0.137451	-463.24368	9.8E07
7	0.023161	-464.08761	2.6E08
8	0.017945	-464.73441	1.1E09
9	0.000339	-464.74657	4.2E13
10	0.000124	-464.75104	7.2E13
11	0.000050	-464.75282	2.3E14

DAT 5

Phase II

loop	μ	cond #
1	0.0110000	2.2E06
2	0.0000069	2.5E06

Initial Objective Value = -289.92699

Phase I

loop	cx	CX	cond #
1	1.483638	-467.99658	7.8E03
2	1.885336	-696.83533	7.6E03
3	1.195118	-846.34235	9.4E03
4	0.240044	-876.00422	1.9E04
5	0.117917	-890.18021	3.0E04
6	0.004640	-890.72835	2.7E04
7	0.001868	-890.94376	2.7E04
8	0.000399	-890.98866	2.9E04
9	0.000073	-890.99664	3.3E04

DAT 7

Phase II

loop	μ	cond #
1	0.8894799	8.7E06
2	0.0037376	1.2E07
3	0.0000023	3.5E07

Initial Objective Value = -983.67871

Phase I

loop	cx	CX	cond #
1	15.047611	-2515.30399	4.1E04
2	6.524582	-3189.02191	4.3E04
3	3.930912	-3608.10769	4.7E04
4	3.265923	-3976.85917	5.4E04
5	1.196356	-4116.28354	6.3E04
6	0.978393	-4239.69436	4.9E04
7	0.342909	-4284.22237	2.6E04
8	0.148122	-4303.54889	2.3E04
9	0.095927	-4315.97001	2.7E04
10	0.009758	-4317.22490	2.9E04
11	0.005256	-4317.89427	2.9E04
12	0.000628	-4317.97359	2.9E04
13	0.000152	-4317.99230	3.0E04
14	0.000033	-4317.99641	3.0E04

DAT 8

Phase II

loop	μ	cond #
1	0.9570915	3.7E08
2	0.6648277	2.8E08
3	0.0008999	5.8E08
4	0.0000006	6.8E14

Initial Objective Value = 558080.74102

Phase I

loop	cx	CX	cond #
1	1066.617920	412254.88963	2.7E19
2	469.131012	348595.20346	2.8E19
3	438.476196	289870.38084	2.8E19
4	259.832062	255340.88190	3.4E19
5	89.343163	243498.10460	4.5E19
6	56.711308	236043.50115	5.7E19
7	50.566032	229459.00759	7.1E19
8	13.313293	227711.66164	1.0E20
9	6.353201	226873.12672	1.3E20
10	3.088514	226461.51388	1.6E20
11	4.397704	225862.31110	2.0E20
12	0.683417	225768.27016	3.2E20
13	0.904609	225643.01792	3.3E20
14	0.507882	225572.54604	4.8E20
15	0.298861	225531.18967	6.3E20
16	0.165659	225508.12011	1.1E21
17	0.069543	225498.41380	1.6E21
18	0.021103	225495.60711	3.5E21
19	0.000568	225495.61190	8.3E21
20	0.000298	225495.61426	8.3E21

VITA

Byeong-Soo Kim

Candidate for the Degree of

Master of Science

Thesis: SOME MODIFICATIONS AND EXTENSIONS OF KARMARKAR'S MAIN ALGORITHM WITH COMPUTATIONAL EXPERIENCES

Major Field: Computing and Information Science

Biographical:

Personal Data: Born in Seoul, Korea, February 27, 1957, The son of Yeon-Soon and Hak-Yung Kim.

Education: Graduated from Seoul High School, Seoul, Korea, in February, 1976; received the Bachelor of Science degree in Mathematics from Seoul National University, Seoul, Korea, in February, 1981; completed requirements for the Master of Science Degree at Oklahoma State University in May, 1987.