A Variable-metric Variant of the
Karmarkar Algorithm for Linear Programming

by

Kathryn Turner

Technical Report 87-13, May 1987

---

1A Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, Rice University.
A Variable-metric Variant of the Karmarkar Algorithm
for Linear Programming

Kathryn Turner

Abstract

The most time-consuming part of the Karmarkar algorithm for linear programming is computation of the step direction, which requires the projection of a vector onto the nullspace of a matrix that changes at each iteration. We present a variant of the Karmarkar algorithm that uses standard variable-metric techniques in an innovative way to approximate this projection. We prove that the modified algorithm that we construct using a step direction obtained from this approximation retains the polynomial-time complexity of the Karmarkar algorithm. We extend applicability of the modified algorithm to the solution of linear programming problems with unknown optimal value, using a construction of monotonic lower bounds on the optimal objective value that approximates the lower bound construction of Todd and Burrell. We show that our modified algorithm for solving problems with unknown optimal value also retains the polynomial-time complexity of the Karmarkar algorithm. Computational testing has verified that our modification substantially reduces the number of matrix factorizations needed for the solution of linear programming problems, compared to the number of matrix factorizations required by the Karmarkar algorithm.

ACKNOWLEDGMENTS

I am deeply grateful to the members of my thesis committee, Professors John Dennis, Richard Tapia, and Richard Young, and Mike Marsden, for sharing with me their mathematical knowledge and their friendship. The chairman, John Dennis, has given an inscrutable amount of time and effort to inspiring, encouraging, and helping me. Richard Tapia has helped immeasurably through many helpful discussions. Both have been sources of tremendous inspiration, not only because of their broad knowledge of mathematics and reverence for truth, but because of their overwhelming generosity with their time, which is always in short supply. I am also very appreciative of the creative ideas and tremendous enthusiasm of Mike Marsden, and the encouragement of Richard Young.

The generous financial support of the Army Research Office under grant ARO DAAG-29-86-K-0005 and the Air Force Office of Scientific Research under grant AFOSR 86-0343 is gratefully acknowledged. Although the defense of this thesis took place before the financial support of Shell Development Company began, my appreciation for details of implementation has been greatly enhanced during the past year. I am very grateful to Shell Development Company for funding this post-doctoral research.

Special thanks are due to Michael Pearson for computer assistance and to Vivian Coo for word-processing assistance. Finally, I am deeply indebted to my family and friends for extremely kind and generous emotional support.
1. Introduction

Since the introduction of the polynomial-time algorithm for linear programming by N. Karmarkar [1984], there has been considerable interest in its use, and in developing strategies for modifying the algorithm to achieve greater computational efficiency.

In Section 2, the algorithm as proposed by Karmarkar is given and discussed, and in Section 3, modifications to the algorithm, contributions, and computational experience proposed or reported by Karmarkar and other researchers are summarized. Section 4 contains the development of our modification of the algorithm, in which we use standard variable-metric techniques in an innovative way to approximate a matrix that is central to the computation of the Karmarkar step, thus admitting a rank-one update to the matrix factorization that will be needed in approximating the step direction, and reducing the amount of work required at each step of the algorithm. Further discussion of the variable-metric variant of the algorithm, including details regarding the required computations, is included in Section 5. Section 6 contains a summary of numerical results obtained using an experimental implementation of the modified algorithm on eight medium-sized test problems, and Section 7 gives directions for further research.
2. The Karmarkar algorithm

The algorithm for linear programming as proposed by Karmarkar [1984] solves problems that have been transformed so that the feasible region of the linear program is the intersection of a simplex with a vector space and the minimum value of the objective function is zero. Following the suggestion of Shanno and Marsten [1985], we shall use the simplex \( \{ x \in \mathbb{R}^n : x \geq 0, e^T x = n \} \), rather than \( \{ x \in \mathbb{R}^n : x \geq 0, e^T x = 1 \} \), as originally used by Karmarkar, so that the problem to be solved is

\[
\begin{align*}
\text{minimize} & \quad e^T x \\
\text{subject to} & \quad Ax = 0 \\
& \quad e^T x = n \\
& \quad x \geq 0,
\end{align*}
\]

(2.1)

where \( e, x, \epsilon \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, \) and \( \epsilon = (1, \ldots, 1)^T \). Karmarkar suggests transforming a standard linear programming problem to this form by converting inhomogeneous equality constraints to homogeneous constraints by writing the \( i \)th constraint, \( a_i^T x = b_i \), as \( a_i^T x = \frac{b_i}{n} \sum_j x_j \), whence

\[
\sum_{j=1}^n (a_{ij} - \frac{b_i}{n}) x_j = 0,
\]

and using a projective transformation to map the non-negative orthant of \( \mathbb{R}^{n-1} \) onto a simplex in \( \mathbb{R}^n \). We prefer to transform the problem to the form (2.1) by commonly used (see, for example, Tomlin [1985]) strategies that do not involve a projective transformation.

The standard linear programming problem

\[
\begin{align*}
\text{minimize} & \quad e^T x \\
\text{subject to} & \quad Ax = b \\
& \quad e^T x = n \\
& \quad x \geq 0,
\end{align*}
\]

(2.1)

where \( x, F \in \mathbb{R}^{n-2}, b \in \mathbb{R}^{m-1}, \) and \( A \in \mathbb{R}^{(m-1) \times (n-2)} \) can be transformed into the linear programming problem (2.1) as follows. First, add a slack variable \( F_{n-1} \), express the requirement \( Ax = b \) as \( \bar{A} x = F_{n-1} + b \), and add a constraint requiring \( F_{n-1} = 1 \). The equality constraints can now be stated as

\[
\begin{bmatrix}
\bar{A} \\
F_{n-1} \\
0 \ldots 0 1
\end{bmatrix}
\begin{bmatrix}
F_1 \\
\vdots \\
F_{n-1}
\end{bmatrix}
= 
\begin{bmatrix}
b \\
0
\end{bmatrix}
\]

Now assume we have a bound \( B \) on the sum of the variables:

\[
\sum_{i=1}^n x_i \leq B
\]

and introduce a second slack variable so that

\[
\sum_{i=1}^n z_i = B.
\]

Using this condition, the constraint equation we have added can be written as

\[
\begin{bmatrix}
\bar{A} \\
F_{n-1} \\
0 \ldots 0 1
\end{bmatrix}
\begin{bmatrix}
F_1 \\
\vdots \\
F_{n-1}
\end{bmatrix}
= 
\begin{bmatrix}
b \\
0
\end{bmatrix}
\]
Application of the Karmarkar algorithm is dependent on the assumption that the optimal value of the objective function is zero. If the optimal value of the objective function is not zero, but is known to be $f^*$, the objective function can easily be transformed to an objective function whose minimum value is zero. Minimizing $\bar{T}^T x$ is equivalent to minimizing $\bar{T}^T x - f^*$, and this second objective function is equivalent to

$$\bar{T}^T x - f^* = \frac{\bar{T}^T x}{n} \sum_{i=1}^{\bar{n}} z_i,$$

so that setting

$$c = \bar{T} - \frac{f^*}{n},$$

(2.2)
gives an objective function $c^T x$ whose minimum value is zero.

For the test problems reported in Section 5, the optimal value of the objective function was known, so that only the shift in $c$ as above was necessary. If the minimum value of the objective function is not known, Karmarkar [1984] suggests the use of what he calls a sliding objective function. This involves attempting to solve the problem with objective function $c^T x$ with $c = \bar{T} - \frac{1}{n} \epsilon$, where $\epsilon$ is an estimate of $f^*$ that is refined as the solution progresses. Todd and Burrell [1985] have suggested using the dual of problem (2.3) to obtain an estimate $\bar{f}$ that is a lower bound on $f^*$ and that is refined at
The Karmarkar algorithm is:

given an initial feasible point $x_0 > 0$

begin
  $x = x_0$
  while $c^T x$ is too large
    do
      $D = \text{diag}(x_1, \ldots, x_n)$, $B = \begin{bmatrix} AD \\ c^T \end{bmatrix}$
      project $Dc$ onto the nullspace of $B$:
      $c_e = (I - B^T(BB^T)^{-1}B)Dc$
      $\hat{c} = c - \alpha c_e$
      $x = \frac{n}{c^T D \hat{c}} D \hat{c}$
    end do
end

The Karmarkar algorithm is an interior point method: the initial point and all subsequent iterates are strictly positive feasible points. At each step of the Karmarkar algorithm, the current iterate $(x_1, \ldots, x_n)^T$ is mapped to the center $(1, \ldots, 1)^T$ of the simplex $\{ x \in \mathbb{R}^n; x \geq 0, c^T x = n \}$, by the projective transformation.
\[ \tilde{z} = \frac{1}{\epsilon^T D^{-1} \epsilon} D^{-1} z, \]  

(2.4)

where \( D = \text{diag}(x_1, \ldots, x_n) \), and we use \( \tilde{z} \) to denote the transformed variable. This projective transformation is a one-to-one mapping of the simplex onto itself, and, in taking the current iterate to the center of the simplex, has the effect of stretching the part of the space near the current iterate. The application of the projective transformation to map the current iterate to the center of the simplex need not be done explicitly, but appears in the algorithm implicitly in the specification of the diagonal matrix \( D \). A step is taken in the transformed space from the center of the simplex, in the direction of the negative projected gradient of a linear function, and the new point in the untransformed space is found by applying the inverse of the projective transformation (2.4) (restricted to \( \{ x \in \mathbb{R}^n : x \geq 0, \epsilon^T x = n \} \)).

\[ z = \frac{1}{\epsilon^T D \epsilon} D \tilde{z}. \]

A fundamental clever idea in the Karmarkar algorithm is to transform the subproblem to be solved at each step to the minimization of a linear function over a sphere, starting from the center of the sphere. The feasible region for the linear programming problem has already been transformed into the intersection of a simplex with a vector space. Now consider approximating minimization over the simplex by minimization over a sphere centered at the center of the simplex. Applying a projective transformation to map a feasible point (the current iterate) to the center of the simplex and sphere yields a feasible region to be considered at each step that is the intersection of a sphere with a vector space that passes through the center of the sphere, so that the region under consideration is a sphere (of lower dimension) with the current iterate at its center. Minimization of a linear function over a sphere at each step, starting from the center of the sphere, is an easy problem. This is accomplished by taking a step from the center of the sphere in the direction of steepest feasible descent for the linear function, with the steplength equal to the radius of the sphere.

While applying the projective transformation transforms the feasible region as we have discussed, minimizing the linear objective function in the untransformed space is not equivalent to minimizing a linear objective function in the transformed space. Expressing the objective function in terms of the transformed variable, the objective function in the transformed space becomes:

\[ \text{minimize } \frac{\epsilon^T D \tilde{z}}{\epsilon^T D \epsilon}. \]

The Karmarkar algorithm approximates the solution of the transformed problem at each step by minimizing the linear function that appears in the numerator above, so that the subproblem solved at each step is
minimize \[ e^T D \hat{x} \]
subject to \[ A \hat{x} = 0 \]
\[ e^T \hat{x} = n \]
\[ \hat{x} \geq 0 \]
\[ \| e - \hat{x} \| \leq \beta r, \] (2.5)

where \( r \) is the radius of the largest sphere that can be inscribed in the simplex, and \( \beta \in (0, 1) \). The step direction from the center of the simplex to the minimizer of (2.5) is independent of the choice of \( \beta \); it is the direction of steepest feasible descent for the linear function \( e^T D \hat{x} \), and is found by projecting \( D e \) onto the nullspace of \( B = \begin{bmatrix} AD \\ e^T \end{bmatrix} \), and taking the negative of this direction. One obtains a strictly positive new iterate \( \hat{x} \) if \( \beta \) is restricted to lie in \((0, 1)\). Approximation of \( \frac{e^T D \hat{x}}{e^T D \hat{x}} \) by \( e^T D \hat{x} \) is justified by the observation that \( e^T D \hat{x} \) is positive for all feasible points and, since it is assumed that \( e^T \hat{x} \) has a minimum value of zero, the minimum value of \( \frac{e^T D \hat{x}}{e^T D \hat{x}} \) and of \( e^T D \hat{x} \) is also zero.

Karmarkar [1984] proves convergence of the algorithm based on using a constant steplength, and his theoretical results seem to suggest a steplength about one-fourth of the radius of the largest sphere that can be inscribed in the simplex would be most advantageous. However, all documented implementations we are aware of have at least attempted to take longer steps. In most directions, it is possible to take a step longer than the radius of the largest sphere that can be inscribed in the simplex without losing feasibility, and it may well be advantageous to do so. If a reduction in the untransformed linear objective function \( e^T r \) is being realized by a step in a given direction, a longer step in that direction will yield more reduction in the untransformed linear objective function, as we discuss in Section 5.3, although the amount of reduction in this function will not be proportional to the length of the step. The Karmarkar algorithm does not guarantee reduction of the untransformed linear objective function at every step, but in practice, such reduction almost always occurs. A complete discussion of how we choose the steplength parameter \( \alpha \) at each step is included in Section 5.3.

A second fundamental clever idea in the Karmarkar [1984] derivation is the introduction of an auxiliary "potential" function,

\[ f(x) = \sum_{i=1}^{n} \log \frac{e^T x}{x_i}, \] (2.6)

that does not appear in the algorithm itself. Karmarkar's proof of convergence of his algorithm and his polynomial-time bound are based on achieving at least a constant amount of reduction in the potential function (2.6) at each step. This potential function is transformable by the projective transformation to a function having the same form. When \( f \) is expressed in terms of the transformed variable, one has
\[ f(x) = \sum_{i=1}^{n} \log \frac{c^T z_i}{z_i} + \text{constant}, \]

so that in the transformed space, one may consider

\[ \tilde{f}(\tilde{x}) = \sum_{i=1}^{n} \log \frac{c^T \tilde{z}_i}{\tilde{z}_i}. \tag{2.7} \]

Karmarkar proves that a step from the center of the simplex to the minimizer of the linear function \( c^T \tilde{z} \) on the sphere having radius one-fourth that of the largest sphere that can be inscribed in the simplex produces at least a constant amount of reduction in this potential function. We shall have occasion to refer to Karmarkar's main convergence result, so we give a modified version of it here.

Theorem 2.8 (Karmarkar [1984], Theorem 1)

An algorithm for solving problem (2.1) that produces a sequence of feasible iterates \( \{x_i\} \) satisfying for all \( i \):

\[ f(x_{i-1}) \leq f(x_i) - \delta. \]

where \( \delta \) is any positive constant, finds a feasible point \( x \) such that

\[ \frac{c^T x_k}{c^T x_0} \leq 2^{-q} \]

in \( O(n (q + \log n)) \) steps.

By hypothesis, for all \( k \)

\[ f(x_k) \leq f(x_0) - k \delta, \]

which is to say that

\[ n \log \frac{c^T x_k}{c^T x_0} \leq \sum_{i=1}^{n} \log (x_k)_i - \gamma - k \delta, \]

where \( \gamma = \sum_{i=1}^{n} \log (x_0)_i \). Now \( (x_k)_i \leq n \) for all \( i, 1 \leq i \leq n \), so that

\[ n \log \frac{c^T x_k}{c^T x_0} \leq n \log n - \gamma - k \delta. \]

Hence, for \( k \geq \frac{n}{\delta} (\log n + q \log 2) - \frac{\gamma}{\delta} \), we have

\[ \frac{c^T x_k}{c^T x_0} \leq 2^{-q}. \]
3. Modifications and contributions of Karmarkar and other researchers

The polynomial-time complexity that has been established for the Karmarkar algorithm does not address the issue of a rate of convergence. Charnes, Song, and Wolfe [1984] show that the sequence of objective function values obtained in the Karmarkar algorithm converges r-linearly. Since the linear objective function values are not necessarily decreased at every step of the Karmarkar algorithm, this is the best convergence rate that can be established.

Several researchers, including Karmarkar, have proposed modifications to the basic Karmarkar algorithm presented in Section 2. Karmarkar [1984], Shanno [1985], Shanno and Marsten [1985], Goldfarb and Mehrotra [1985] and Pickel [1985] have proposed algorithms that use directions other than the negative of the projected gradient \( -c^T \). The Karmarkar [1984] modification approximates \( D \) by \( \tilde{D} \), a diagonal matrix obtained from the approximation used at the previous step by first applying a scaling factor, and then changing those elements corresponding to components of the current iterate that have changed significantly. The factorization is adjusted for each changed element by rank-one modifications. This modification reduces the time complexity of the Karmarkar algorithm by a factor of \( n^{6.5} \), since Karmarkar shows that, on average, \( \sqrt{n} \) updates per step are sufficient. Shanno [1985] investigated relaxing the criterion for changing components of the approximation. A relaxation of this criterion reduces the average number of updates required at each step, but also reduces the expected decrease of the potential function (2.6). Test results were reported by Shanno on randomly generated problems ranging in size from 10 to 50 rows and from 30 to 150 columns. Execution times were not given, but the results were interpreted as indicating improved performance for relaxation of the criterion up to a factor of 2.5 times that given by Karmarkar.

Shanno and Marsten [1985] proposed using a reduced gradient, rather than a projected gradient direction. They reported difficulties with the resulting algorithm, such as its being susceptible to converging to non-optimal points. A modification of the reduced gradient direction, based on the transformed objective function \( \frac{c^T D_{\tilde{z}}}{c^T D_{\tilde{x}}} \), overcame this difficulty, but still they were rather disappointed in the performance of the algorithm. They also implemented an inexact projection algorithm, using conjugate gradients without a preconditioner, also with limited success.

Pickel [1985, 1986] has also investigated using approximate projections, based on a partitioning of the constraint matrix as in Shanno and Marsten. Pickel's implementation was not very successful. In addition to the difficulties reported by Shanno and Marsten [1985] in identifying the correct basis,
difficulties were reported due to implicit treatment of upper bounds in this implementation.

An inexact projection algorithm, using preconditioned conjugate gradients, was proposed by Goldfarb and Mehrotra [1985]. They extended applicability of their algorithm [1986] to the solution of problems having unknown optimal objective value through the use of an improving sequence of lower bounds as in Todd and Burrell [1985], and showed that their relaxed version of the Karmarkar algorithm has the same polynomial-time complexity as the Karmarkar algorithm. They gave computational results on three test problems: AFIRO, ADLITTLE, and SHARE2B, solving the problems to greater accuracy than we did. Although they did not compare the time required for computing solutions using their relaxed version of the algorithm to the time required using the unmodified Karmarkar algorithm, they found their preliminary computational results encouraging.

Vanderbei, Meketon, and Freedman [1985], Cavalier and Soyster [1985], Cavalier and Schall [1985], Kortanek and Shi [1985], and Sherali [1985] have worked with an affine variant of the Karmarkar algorithm. This variant poses the problem in the positive orthant rather than in a simplex, and uses the transformation \( \bar{x} = D^{-1}x \), to simply translate the current iterate to \((1, \ldots, 1)^T\) rather than using the projective transformation (2.4). This simplifies the algorithm, conceptually and computationally, gives monotonicity of linear objective values, and does not require that the minimum value of the objective function be zero. However, the simplification is at the expense of losing the worst-case polynomial-time complexity of the Karmarkar algorithm. Vanderbei, Meketon, and Freedman [1985] prove convergence of the affine variant of the algorithm, assuming primal and dual non-degeneracy, and report that for small, dense problems, the affine variant of the algorithm is competitive with the revised simplex method. Kortanek and Shi [1985] and Sherali [1985] also give proofs of convergence for this variant. Cavalier and Schall give numerical results showing that their implementation of the affine variant of the Karmarkar algorithm for inequality constrained problems in which they also include constraint partitioning is about twice as fast as IMSL's ZX4LP on moderate-sized test problems. In the Cavalier and Schall implementation, slack variables are distinguished from variables appearing in the original inequality constraints. Their constraint partitioning scheme uses the active set strategy of identifying the constraints that become more binding from one iteration to the next, and computing a step direction using the submatrix of the constraint matrix consisting of the identified rows. Their implementation of the affine algorithm without constraint partitioning required about twice as much time as ZX4LP to solve their test problems.
Karmarkar [1984], Todd and Burrell [1985], Gay [1985a], and Anstreicher [1985] have given approaches for dealing with problems having unknown optimal values, as noted in Section 2. All of these approaches involve obtaining estimates of the optimal objective value, refined as the iteration progresses, and all except the approach given by Karmarkar compute lower bounds on the optimal objective value.

Gill, Murray, Saunders, Tomlin, and Wright [1985], and Anstreicher [1985] have given alternative ways of viewing the algorithm. Anstreicher interprets the Karmarkar algorithm as an algorithm for fractional linear programming on the simplex. He also gives a modification of the (fractional linear programming) algorithm for which monotonicity of objective values is assured. Gill, Murray, Saunders, Tomlin, and Wright [1985] show the equivalence of the Karmarkar algorithm to a projected Newton barrier method with a particular choice of the sequence of barrier parameters. The barrier parameters corresponding to the Karmarkar algorithm, however, are not always positive, so are not always admissible in the barrier method. They give numerical results comparing their barrier implementation with the primal simplex code MINOS 5.0 on nine test problems, including seven of the problems for which we report numerical results in Section 6: AFIRO, ADLITTLE, SHARE2B, SHARE1B, BEACONFD, ISRAEL, and BRANDY. Times for solving six of the nine problems by the simplex and barrier methods were comparable; for three of the test problems, the barrier method was slower than the simplex method by factors ranging from two to five. Also included in the numerical results were three highly degenerate problems. For these problems, their FORTRAN barrier method code was compared to the WHIZARD assembly language simplex code. Times required to solve these problems using the barrier code were from 1.28 to 2.34 times those required to solve the problems using WHIZARD.

Computational results for the Karmarkar algorithm, or its variants, have also been reported by Aronson, et al. [1985], Tomlin [1985], Lustig [1985], and Coimenes [1985]. None of the researchers reporting computational results on general linear programming problems have found the Karmarkar algorithm to be significantly faster than the simplex algorithm on all problems.

Aronson, et al tested their version of the affine variant of the Karmarkar algorithm, with bounds handled implicitly, against a pure network code NETFLO and IBM's MPSX/370 on five randomly generated dense assignment problems with the number of nodes ranging from 50 to 160. For these problems, the pure network code NETFLO was fastest; solution times using MPSX/370 were about 18 times those of NETFLO; and solution times using their variant of the Karmarkar algorithm were about 14 times those of MPSX/370. Tomlin [1985] tested his implementation of the Karmarkar algorithm against WHIZARD on nine test problems ranging in size from 20 to 365 constraints and from 23 to
472 variables excluding slacks. These test problems included AFIRO, ADLITTLE, SHARE2B, ISRAEL, and BRANDY, for which we give numerical results in Section 6. Tomlin used constant steplengths in his Karmarkar code, and reported that the number of iterations required for the solution of the test problems varied inversely with the steplength. His most efficient Karmarkar code used Givens rotations in computing step directions and a steplength .60 of the radius of the largest sphere that can be inscribed in the simplex. This version was faster than WHIZARD on the three smallest problems, but was less than twice as fast. On five of the remaining problems, the Karmarkar code was two to ten times slower than WHIZARD, and on the problem ISRAEL the Karmarkar code was more than 20 times slower than WHIZARD.

Lustig [1985] tested his variant of the Karmarkar algorithm against the simplex code MINOS on eight test problems, including AFIRO, ADLITTLE, SHARE2B, ISRAEL, and BRANDY. His projective algorithm was competitive with the simplex algorithm only on his two smallest problems. For the remaining six problems, the projective algorithm was 12 to 120 times slower than the simplex algorithm. Colmenares [1985] tested his straightforward implementation of the Karmarkar algorithm as given in Karmarkar [1984] against the primal-dual simplex code TOYLP on ten small problems (2 to 45 variables, 2 to 38 constraints), one larger problem (2086 variables, 101 constraints), and Klee-Minty problems of sizes 2, 3, and 4. Colmenares included in his small test problems: a 2-variable infeasible problem, a condition his Karmarkar code failed to detect; a 3-variable problem with nearly singular coefficient matrix, for which his Karmarkar code failed to converge; and a 2-variable unbounded problem, for which his Karmarkar code found the unbounded feasible direction at iteration 11. For the problems successfully solved by his Karmarkar code, solution times were two to five times those of TOYLP.

Hooker [1986] gives a good discussion of the Karmarkar algorithm, and a summary of contributions of other researchers.
4. A variable-metric variant of the Karmarkar algorithm

It is clear that the most time-consuming part of the Karmarkar algorithm is
computation of the step direction, which requires computing the projection of
the vector $Dc$ onto the nullspace of a matrix $B$ that changes at each iteration.
We propose to approximate the direction so obtained by employing an
approximation to $D$ in such a way that rank-one updates to the factorization of
the initial matrix $(AD_0)(AD_0)^T$ are possible. The updating strategies we have
adapted for use in this context have been used very successfully in the solution
of nonlinear equations and in the minimization of nonlinear functions. By
employing updates we reduce the amount of work required at each step, since
we bypass the need to obtain a matrix factorization at every step. We shall
approximate the direction $\hat{e}_p = [J - B^T (BB^T)^{-1} B] Dc$ by

$$\hat{e}_p = D^{-1} \hat{D} [J - \hat{B}^T (\hat{B} \hat{B}^T)^{-1} \hat{B}] \hat{D}^T e.$$  \hspace{1cm} (4.1)

where

$$\hat{D} = \begin{bmatrix} A \hat{D} \\ e^T D^{-1} \hat{D} \end{bmatrix} = BD^{-1} \hat{D},$$

and $\hat{D}$ is a nonsingular approximation to $D$. The new iterate in the transformed
space will be computed as

$$\tilde{x} = x - \alpha \hat{e}_p.$$ 

Before addressing the way we obtain the matrix $\hat{D}$, we note that if $\hat{D} = D$,
then $\hat{e}_p = e_p$. Moreover, we note that any direction $\hat{e}_p$ computed in this
manner has two desirable attributes: from $e = (1, \cdots, 1)^T$, the direction $-\hat{e}_p$ is
a feasible direction for the linear program and it is a descent direction for
Karmarkar's potential function (2.7).

**Theorem 4.2**

Let

$$\hat{e}_p = D^{-1} \hat{D} [J - \hat{B}^T (\hat{B} \hat{B}^T)^{-1} \hat{B}] \hat{D}^T e,$$

where $\hat{D}$ is any nonsingular approximation to $D$ and

$$\hat{B} = \begin{bmatrix} A \hat{D} \\ e^T D^{-1} \hat{D} \end{bmatrix} = BD^{-1} \hat{D}.$$

Then from $e = (1, \cdots, 1)^T$

(i) $-\hat{e}_p$ is a feasible direction for the linear programming problem (2.1).

and

(ii) $-\hat{e}_p$ is a descent direction for the potential function (2.7).

**Proof.**

Feasibility of the direction $-\hat{e}_p$ follows from the observation that $\hat{e}_p$ is in
the nullspace of $\hat{B}$. Recall that the potential function (2.7) is
\[ \tilde{j}(\tilde{x}) = \sum_{i=1}^{n} \log \frac{c^T D \tilde{x}}{\tilde{x}_i}. \]

Now
\[ \nabla_{\tilde{x}} \tilde{j}(\tilde{x}) = -\frac{n}{c^T D \tilde{x}} \tilde{x} \tilde{D}^{-1} \tilde{e}, \quad \text{where } \tilde{D} = \text{diag}(\tilde{x}_1, \ldots, \tilde{x}_n), \]

and
\[ \nabla_{\tilde{x}} \tilde{j}(\epsilon) = -\frac{n}{c^T x_c} \tilde{D} c - \epsilon, \]

where \( x_c \) denotes the current iterate in the untransformed space. The direction \( -\tilde{\epsilon}_p \) is a non-scent direction for the potential function in the transformed space since
\[ \nabla_{\tilde{x}} \tilde{j}(\epsilon)^T (-\tilde{\epsilon}_p) = \left[ -\frac{n}{c^T x_c} \tilde{D} c + \epsilon \right]^T \tilde{\epsilon}_p \]
\[ = -\frac{n}{c^T x_c} c^T \tilde{D} [I - \tilde{D} (\tilde{D}^T \tilde{D})^{-1} \tilde{D}] \tilde{D} \tilde{e} \]
\[ \leq 0. \]

In order to show that the direction \( -\tilde{\epsilon}_p \) is a descent direction for the potential function, we note that \( \epsilon_p \neq 0 \), since there exists a step in the direction \( -\tilde{\epsilon}_p \) that yields a reduction in the potential function, and that \( -\tilde{\epsilon}_p \) can fail to be a descent direction only if \( \tilde{D}^T \epsilon \) is orthogonal to the nullspace of \( \tilde{D} \).

But if \( \tilde{D}^T \epsilon \) is orthogonal to the nullspace of \( \tilde{D} \), then there exists \( y \in \mathbb{R}^m \) such that
\[ \tilde{B}^T y = \tilde{D}^T \epsilon \]
\[ \tilde{D}^T \tilde{D}^{-1} \tilde{B}^T y = \tilde{D}^T \epsilon, \text{ and since } \tilde{D} \text{ is nonsingular,} \]
\[ \tilde{B}^T y = D \epsilon, \]

which contradicts the fact that \( \epsilon_p \neq 0 \).

We now address the issue of how approximations \( \tilde{D} \) may be obtained. We shall first outline our modification of the Karmarkar algorithm as if only a factorization of the initial matrix \( (AD_0)(AD_0)^T \) were required. For a practical implementation, there will be a limit on the number of steps taken before computing another factorization. This will be discussed further in section 5.2. After computing the initial factorization, we may store update vectors. If update vectors are being used, the matrix \( \tilde{D} \) will not be explicitly formed, and the factorization of \( (AD_0)(AD_0)^T \) will not be disturbed. The projection computation will be done cheaply, requiring only solutions of linear systems from the initially factored matrix, scalar products, and sums of vectors.

We begin with \( D_0 = D_0^{(0)} \), a diagonal matrix whose diagonal elements are the components of the initial feasible point \( x_0 \). We may obtain rank-one secant updates to this matrix through the use of one of two nonlinear functions whose gradients and Hessians involve the matrix \( D \). The first and simplest of these is a logarithmic barrier function, and the second is the potential function we have already defined. Let us first examine the logarithmic barrier function.
Applying the barrier transformation to
\[
\begin{align*}
\min_x & \quad e^T x \\
\text{subject to} & \quad Ax = 0 \\
& \quad e^T x = n \\
& \quad x \geq 0
\end{align*}
\] (4.4)
gives
\[
\begin{align*}
\min_x & \quad F(x) = e^T x - \mu \sum_{i=1}^n \log x_i \\
\text{subject to} & \quad Ax = 0 \\
& \quad e^T x = n
\end{align*}
\] (4.5)
the inequality constraints having been replaced by a term in the objective function. It is well known (see, for example, Gill, Murray, Saunders, Tomlin, and Wright, [1985]) that under mild hypotheses the solution to (4.5) converges to the solution of (4.4) as $\mu \to 0$. It is not our intention to solve problem (4.5) for any value of $\mu$; we intend merely to use the logarithmic barrier function, whose gradient and Hessian involve $D$, in order to obtain an approximation to this matrix. We have
\[
\nabla F(x) = e - \mu D^{-1} e,
\]
where $D = \text{diag}(x_1, \ldots, x_n)$ and $e = (1, \ldots, 1)^T$; and
\[
\nabla^2 F(x) = \mu D^{-2}
\]
Analogous to the idea of a secant approximation to the derivative of a function

of one variable as the ratio of the change in function values to the change in the independent variable, a secant approximation to a Hessian is obtained by requiring the new approximation to the Hessian acting on the step in the independent variable to produce the difference in gradient vectors at the new and previous points. Using the subscript $+\dd$ to refer to the new iterate and the subscript $-\dd$ to refer to the previous (current) iterate, we may write the secant equation based on the logarithmic barrier function as
\[
\mu D^{-2}_{+}(x_{+} - x_{-}) = (e - \mu D^{-1}_{+} e) - (e - \mu D^{-1}_{-} e), \quad \text{or}
\]
\[
D^{-2}_{+}(x_{+} - x_{-}) = (D^{-1}_{+} - D^{-1}_{-}) e.
\] (4.5)
The righthand side will be computed exactly. Letting $y_{i}$ denote $(D_{+}^{-1} - D_{-}^{-1}) e$, and, since we do not wish to require the approximation $\tilde{D}$ to be symmetric, replacing $D_{+}^2$ with $\tilde{D}$, $\tilde{D}^T$, we require the approximation to satisfy
\[
(x_{+} - x_{-}) = \tilde{D} \tilde{D}^T y_{i}
\]
An approximation based on the potential function (2.6) requires a very similar secant equation to be satisfied. Recall that the potential function is
\[
f(x) = \sum_{i=1}^n \log \frac{e^T x}{x_i}
\]
so that
\[
\n\nabla^2 f(x) = \frac{n}{e^T x} - e - D^{-1}\varepsilon,
\]

\[
\nabla^2 f(x) = D^2 - \frac{n}{(e^T x)^2} ee^T,
\]

and the secant equation is

\[
\left[D^2 - \frac{n}{(e^T x)^2} ee^T\right] (x_i - x) = \left[\frac{n}{e^T x_i} - e - D^{-1}\varepsilon\right] - \frac{n}{e^T x_i} e - D_i^{-1}\varepsilon.
\]

We do not approximate the second term of the Hessian, nor do we approximate any terms appearing on the right-hand side of the above equation. Moving the second term on the left-hand side of the equation to the right-hand side and collecting terms gives

\[
D^2 (x_i - x) = (D_i^{-1} - D^2) e - \frac{n(e^T x_i - e^T x)^2}{(e^T x_i)^2} e.
\]

Now letting \( y \) denote the right-hand side of equation (4.7), and again replacing \( D^2 \) with \( D_i D_i^T \), we require the approximation to satisfy

\[
(x_i - x) = D_i D_i^T y. \tag{4.7}
\]

Recall that the matrix we are trying to approximate is a known diagonal matrix. A third and even simpler alternative requirement that can be imposed on the approximation is that it satisfy

\[
D^2 (x_i - x) = D_i^2 (x_i - x). \tag{4.8}
\]

We shall use \( y \) to denote \( D_i^2 (x_i - x) \) and express this requirement as

\[
(x_i - x) = D_i D_i^T y.
\]

To obtain an updating formula based on the barrier function or on the potential function or on the true matrix \( D_i \) (where \( y \) denotes either \( y_i \) or \( y_j \) or \( y_k \)), the righthand side of equation (4.8) or (4.7) or (4.8), respectively), we follow the development of the BFGS updating formula in Dennis and Schnabel [1983]. For completeness, we include the development of the updating formula here. We express the requirement \( (x_i - x) = D_i D_i^T y \) as

\[
(x_i - x) = D_i v
\]

\[
v = D_i^T y
\]

for some vector \( v \in \mathbb{R}^n \). Now we use the Broyden updating formula to obtain \( D_i \) as the closest matrix to \( D_i \) consistent with satisfying the first part of (4.9):

\[
D_i = D_i + \frac{(x_i - x) - D_i v) v^T}{v^Tv} y. \tag{4.10}
\]

The second part of (4.9) requires that

\[
v = D_i^T y = D_i^T y + \frac{(x_i - x - D_i v) v^T}{v^Tv} v. \tag{4.11}
\]

and this can be satisfied only if

\[
v = \kappa D_i^T y \tag{4.12}
\]
for some \( \kappa \in \mathbb{R} \). Now substituting (4.12) into (4.11) and simplifying gives

\[
\kappa^2 = \frac{(x_+ - x_0)^T y}{y^T \hat{D}_t \hat{D}_t^T y}
\]

Assuming \((x_+ - x_0)^T y > 0\), we choose the positive root, so that

\[
v = \left( \frac{(x_+ - x_0)^T y}{(\hat{D}_t^T y)^2 (\hat{D}_t^T y)} \right)^{1/2} \hat{D}_t^T y.
\]  

(4.13)

For the update based on the true \( D_t \), \( y_t = D_t^{-1} (x_+ - x_0) \), so that

\[
(x_+ - x_0)^T y_t = (x_+ - x_0)^T D_t^{-1} D_t^{-1} (x_+ - x_0) > 0.
\]

For the update based on the barrier function, \( y_t = (D_t^{-1} - D_t^{-1}) c + c \), and we have

\[
(x_+ - x_0)^T y_t = (x_+ - x_0)^T D_t^{-1} D_t^{-1} (x_+ - x_0) > 0.
\]

For the update based on the potential function,

\[
y_t = (D_t^{-1} - D_t^{-1}) c - \frac{n (c^T x_0 - c^T x_0) c}{(c^T x_0 - c^T x_0)} c,
\]

so that

\[
(x_+ - x_0)^T y_t = (x_+ - x_0)^T D_t^{-1} D_t^{-1} (x_+ - x_0) - \frac{n [c^T (x_+ - x_0)]^2}{(c^T x_0 - c^T x_0)} c.
\]

which is not guaranteed to be positive, but is certainly positive if \( c^T x_+ < c^T x_0 \). If this update is being used and the above is not positive at some step, we choose to use the update based on the true \( D_t \), rather than reject the update altogether. Thus far in our computational testing, the situation in which \((x_+ - x_0)^T y_t \leq 0\) has not arisen.

The updating formula given by equation (4.10) was obtained as a least change secant update for approximating \( D_t \), the matrix we are interested in, making use of nonlinear functions whose gradients and Hessians involve the matrix \( D_t^{-1} \). We have also explored the use of the updating strategy that changes the approximation to the Hessian of the nonlinear function as little as possible (rather than changing the approximation to the inverse of the Hessian as little as possible) consistent with having the approximation satisfy the secant equation (4.6) or (4.7) or the condition (4.8). Using this approach, the approximation is required to satisfy

\[
\hat{D}_t^{-1} \hat{D}_t^{-T} (x_+ - x_0) = y,
\]

and analogous to (4.9), we express this requirement as

\[
v = \hat{D}_t^{-1} v + \hat{D}_t^{-T} (x_+ - x_0)
\]

(4.14)

for some vector \( v \in \mathbb{R}^n \). Now the closest matrix \( \hat{D}_t^{-1} \) to \( D_t^{-1} \) satisfying the first part of (4.14) is

\[
\hat{D}_t^{-1} = D_t^{-1} + \frac{(y - \hat{D}_t^{-1} v)^T}{v^T v}.
\]  

(4.15)
and from the second part of (4.14) we obtain

\[
v = \left( \frac{y^T (x_n - z_1)}{\langle \hat{D}_I^{-1}(x_n - z_1) \rangle \hat{D}_I^{-1}(x_n - z_1)} \right)^{-1} \hat{D}_I^{-1}(x_n - z_1).
\]

(4.16)

Using the Sherman-Morrison-Woodbury formula, (4.15) is equivalent to

\[
\hat{D}_n = \hat{D}_I + \frac{(v - \hat{D}_I y) (x_n - z_1)^T}{y^T (x_n - z_1)}.
\]

(4.17)

Use of the update given by (4.17) requires more computation and more storage than is required if the update given by (4.10) is used, as we shall discuss in section 5. Furthermore, numerical experience thus far has indicated that better performance is achieved using the update given by (4.10).

We explored the use of sizing to attempt to improve the approximation (4.10). We adapted the idea of sizing from Oren [1973] and Dennis, Gay, and Welsch [1981] to attempt to improve the performance of the modified algorithm by making the approximation \( \hat{D}_I \) more closely resemble \( D \). Given any two nonsingular matrices \( M \) and \( N \), one may apply a sizing factor \( \sigma \) to force the spectra of \( M \) and \( \sigma N \) to overlap, by using Rayleigh quotients as follows. Using any nonzero vector \( w \), one may choose \( \sigma = \frac{w^T M w}{w^T N w} \), giving

\[
\frac{w^T M w}{w^T w} = \frac{\sigma w^T N w}{w^T w}.
\]

In the context of our approximation, the idea was to choose \( \sigma \) to force the spectra of \( \sigma^2 \hat{D}_I \hat{D}_I^T \) and \( D_n \) to overlap before applying the least change secant update to obtain \( \hat{D}_n \) from \( \sigma \hat{D}_I \). The vector used in the Rayleigh quotient was \( y \), where \( y \) again denotes \( y_c \) or \( y_r \) or \( y_l \), the right-hand side of equation (4.6) or (4.7) or (4.8), respectively. That is, we considered choosing

\[
\sigma = \left| \frac{y^T \hat{D}_I \hat{D}_I^T y}{y^T y} \right|.
\]

However, since \( D_n \) is a diagonal matrix, we had easy access to the spectrum of \( D_n^2 \). It seemed most sensible to us not to disturb the approximation if \( \frac{y^T \hat{D}_I \hat{D}_I^T y}{y^T y} \) was already in the spectrum of \( D_n^2 \). Hence, in this case, we chose \( \sigma = 1 \). In our computational testing, the spectra of these matrices always overlapped, so that application of a sizing factor different from 1 was never needed.

A natural question to ask is whether it is advantageous to use update vectors at all in the modified algorithm. A reasonable alternative strategy within the framework we have established in (4.1) is, as mentioned by Gill, Murray, Saunders, and Wright [1986], to retain the same approximation to \( D \) for several iterations before recomputing a factorization. In an algorithm with periodic refactorizations, one would have \( \hat{D}_I = D_0 \) for \( 0 \leq i \leq k - 1 \) for some \( k \).
and then treat the $k$th iterate as the initial point. We have also explored the
use of this strategy, and include it in the numerical results given in section 6.

5. Computational issues in the modified algorithm

5.1 Update vectors

The computation of $\hat{v}_k$ requires the solution of a linear system with
coefficient matrix $\hat{D}^T$, which requires the solution of a linear system with
coefficient matrix $(\hat{A}^T)(\hat{A}^T)^T$, since

$$
\hat{D}^T = \begin{bmatrix}
A \hat{D} \\
\epsilon^T \hat{D}^{-1} \epsilon
\end{bmatrix}
\begin{bmatrix}
(A \hat{D})^T \\
(\hat{A}^T)^T
\end{bmatrix}
= \begin{bmatrix}
(A \hat{D})(A \hat{D})^T & A \hat{D} \hat{D}^T \hat{D}^{-1} \epsilon \\
(\hat{A}^T)^T & \epsilon^T \hat{D}^{-1} \epsilon
\end{bmatrix}.
$$

Letting $M_a$, $a$, and $\sigma$ denote the matrix $(A \hat{D})(A \hat{D})^T$, the vector $A \hat{D} \hat{D}^T \hat{D}^{-1} \epsilon$, and
the scalar $\epsilon^T \hat{D}^{-1} \epsilon$, respectively, an expression for $(\hat{D}^T)^{-1}$ is given by

$$(\hat{D}^T)^{-1} = \begin{bmatrix}
M_a \\
a \epsilon
\end{bmatrix}^{-1} = \frac{1}{\sigma - a \epsilon^T M_a^{-1} a}
\begin{bmatrix}
(M_a^{-1} \epsilon)(M_a^{-1} \epsilon)^T & -M_a^{-1} a \\
-M_a^{-1} a & -M_a^{-1}
\end{bmatrix}.$$

If update vectors are not being used, so that $\hat{D}_i = \hat{D}_0$ for $0 \leq i \leq k$, the
matrix $[(A \hat{D})(A \hat{D})^T]$ remains unchanged for $k$ steps. At these steps one has
only to compute the vector $A \hat{D} \hat{D}^T \hat{D}^{-1} \epsilon$ ($= A \hat{D} \hat{D}^{-1} \epsilon$), and the scalar
$(D_0 \hat{D}^T \hat{D}^{-1} \epsilon)^T (D_0 \hat{D}^{-1} \epsilon)$. Computations required to compute the step direction can
then be carried out using the factorization of $(A \hat{D}_0)(A \hat{D}_0)^T$ and employing the
above expression for the partitioned inverse of $\hat{D}^T$. 
If update vectors are being used, we need only to be concerned with obtaining an expression for \( (A\hat{D})(A\hat{D})^T)^{-1} \). We first discuss the computation of \( \hat{e}_y \) using the update to \( \hat{D} \) given by (4.10) and (4.13). Referring to the updating formula (4.10),

\[
A\hat{D}_r = A\hat{D}_r - \frac{(A\hat{D}_r)v}{v^Tv} = A\hat{D}_r \left( I - \frac{vv^T}{v^Tv} \right).
\]

Hence,

\[
(A\hat{D}_r)(A\hat{D}_r)^T = A\hat{D}_r \left( I - \frac{vv^T}{v^Tv} \right)(A\hat{D}_r)^T
\]

\[
= (A\hat{D}_r)(A\hat{D}_r)^T - \frac{(A\hat{D}_r)v(A\hat{D}_r)v^T}{v^Tv}.
\]

Now letting \( M_r \) and \( M_v \) denote \( (A\hat{D}_r)(A\hat{D}_r)^T \) and \( (A\hat{D}_v)(A\hat{D}_v)^T \), respectively, and setting \( w = A\hat{D}_r v \) and \( \beta = v^Tv \), the Sherman-Morrison-Woodbury formula gives

\[
M_r^{-1} = \left[ I - \frac{vw^T}{\beta} \right]^{-1} = M_v^{-1} + \frac{M_v^{-1}ww^TM_v^{-1}}{\beta - w^TM_v^{-1}w}.
\]

Setting \( t = M_v^{-1}w \) and \( \gamma = \beta - w^TM_v^{-1}w \), this is

\[
= \left[ I + \frac{tw^T}{\gamma} \right] M_v^{-1}.
\]

In summary, at each step of the algorithm using the update to \( \hat{D} \) given by (4.10) and (4.13), we save four vectors and two scalars : \( v, w \in \mathbb{R}^n \), \( u, t \in \mathbb{R}^m \).

and \( \beta, \gamma \in \mathbb{R} \) as follows:

\[
v = \frac{(z_r-x_r)^T y}{(\hat{D}_r y)^T (\hat{D}_r y)}, \quad w = A\hat{D}_r v,
\]

\[
u = x_r - z_r - \hat{D}_r v, \quad t = [(A\hat{D}_r)(A\hat{D}_r)^T]^{-1}w,
\]

\[
\beta = v^Tv, \quad \gamma = \beta - w^T t.
\]

The computation of \( \hat{e}_y \) is carried out using

\[
\hat{D}_r = D_0 + \frac{u_1v_1^T}{\beta_1} + \cdots + \frac{u_kv_k^T}{\beta_k}
\]

and

\[
[(A\hat{D}_r)(A\hat{D}_r)^T]^{-1} = \left[ I + \frac{u_1v_1^T}{\gamma_1} \right] \cdots \left[ I + \frac{u_kv_k^T}{\gamma_k} \right]^{-1}[(AD_0)(AD_0)]^{-1}
\]

so that the only factorization needed for \( k \) steps of the algorithm is of the initial matrix \( (AD_0)(AD_0)^T \). Notice also that vector or pipeline architectures are immediately applicable to the required computations.

We are indebted to M.J. Todd (private communication) for pointing out to us that computational savings may be realized if the updates to \( M^{-1} \) are saved in factored form. To this end, suppose that \( (A\hat{D}_r)(A\hat{D}_r)^T = L_rL_r^T \). Then

\[
[(A\hat{D}_r)(A\hat{D}_r)^T]^{-1} = L_r^{-1}L_r^{-1} = N_rN_r^T.
\]
\[
M_{z,1}^{-1} = N_z N_z^T + \frac{N_x N_x^T w w^T N_x N_x^T}{\beta - \gamma w^T N_x N_x^T w}.
\]

Setting \( q = N_z^T w \) and \( \gamma = \beta - \gamma w^T q + \left[ \gamma \beta - \gamma q^T q \right]^T \), we obtain

\[
M_{z,1}^{-1} = N_z N_z^T + \frac{N_x q q^T N_x^T}{\beta - \gamma q^T q} = N_z \left[ I + \frac{q q^T}{\gamma} \right] N_z^T = N_z \left[ I + \frac{q q^T}{\gamma} \right] \left[ I + \frac{q q^T}{\gamma} \right]^T N_z^T,
\]

so that the update to the Cholesky factor is given by

\[
N_{z,1} = N_z \left[ I + \frac{q q^T}{\gamma} \right].
\]

Using this approach, at each step of the algorithm using the update to \( \hat{D} \) given by (4.10) and (4.13), one would save three vectors and two scalars:

\( v, u \in \mathbb{R}^n, q \in \mathbb{R}^m, \) and \( \beta, \gamma \in \mathbb{R} \) as follows:

\[
v = \left( \frac{(x_i - x_i) y}{(x_i y - x_i y)} \right)^T \hat{D}_i^T y, \quad q = N_i^T \hat{D}_i v, \quad u = x_i - x_i - \hat{D}_i v, \quad \beta = v^T v, \quad \gamma = \beta - \gamma q^T q + \left[ \gamma \beta - \gamma q^T q \right]^T.
\]

The computation of \( \hat{D}_1 \) would be carried out using (5.1.1) and

\[
(AD_x)^T = N_x N_x^T
\]

where

\[
N_x = N_0 \left[ I + \frac{q_1 q_1^T}{\gamma_1} \right] \left[ I + \frac{q_2 q_2^T}{\gamma_2} \right] \ldots \left[ I + \frac{q_k q_k^T}{\gamma_k} \right].
\]

Saving the updates in factored form would save storage of \( w \), and the computation of \( q \) rather than \( t \) would save \( N \) floating point operations, where \( N \) is the number of nonzeros in the Cholesky factor of \( (AD_b)(AD_b)^T \). The computation (5.1.4), however, would require \( m \) more floating point operations than (5.1.3) at the \( k \)th step, so that the amount of computation saved would vary, depending on sparsity. The implementation for which numerical results are given in section 6 uses updates (5.1.1).

If the updating formula given by (4.17) supported by (4.16) is used, the computation of \( \hat{D}_1 \) requires more than twice as much storage and nearly twice as much work as this computation using updating formula (4.10) supported by (4.13). This is because the use of (4.17) leads to a rank-two update to obtain \( (AD_x)(AD_x)^T \) at each step. At each step of the modified algorithm using the update (4.17), the eight vectors \( s, r, v, u \in \mathbb{R}^n \) and \( p, q, d, \sigma \in \mathbb{R}^m \) and the four scalars \( \beta, \gamma, \eta, \) and \( \rho \) are saved:
\[ s = z_+ - z_- \]
\[ u = v - \hat{D}_s y \]
\[ t = D^{-1}_s u \]
\[ \beta = y^T u \]
\[ \gamma = v^T v \]

where \( M_s = (\hat{A}\hat{D}_s)^T(\hat{A}\hat{D}_s) \),

\[ M_s^{-1} = \left[ I - \frac{d_1 s_1^T}{\eta_1} \right] \left[ I - \frac{d_2 s_1^T}{\eta_2} \right] \quad \ldots \]

\[ \left( 5.1.5 \right) \]

and

\[ M_s^{-1} = \left[ I - \frac{d_1 s_1^T}{\eta_1} \right] \left[ I - \frac{d_2 s_2^T}{\eta_2} \right] \left[ I - \frac{d_3 s_3^T}{\eta_3} \right] \left[ I - \frac{d_4 s_4^T}{\eta_4} \right] \quad \ldots \]

The approximation \( \hat{D}_s \) is given by

\[ \hat{D}_s = D_0 + \frac{u_1 s_1^T}{\beta_1} + \ldots + \frac{u_n s_n^T}{\beta_n} \]

The computation of \( v \) requires \( \hat{D}_s \hat{D}_s^T u \). The matrix \( \hat{D}_s \) is

\[ \hat{D}_s^{-1} = D_0^{-1} - \frac{d_1 v^T}{\eta_1} \ldots - \frac{d_n v^T}{\eta_n} \]

Using updating formula (4.17), it is still the case that the only matrix factorization needed for \( k \) steps of the modified algorithm is of the initial matrix \( (A\hat{D}_0)(A\hat{D}_0)^T \).

5.2 Restarting strategy for the modified algorithm

In practice, there is a limit on the number of update vectors one is willing to store. We have implemented two ways of proceeding when the allotted storage has been used. The option we recommend is to restart, by which we mean treating the current point as the initial point. This, of course, requires another matrix factorization. In our implementation, the restart strategy includes restarting when the approximation appears not to give a direction of sufficient decrease as well as when the maximum number of update vectors has been saved. We also plan to try a restart strategy based on the partial refactorization suggested by Karmarkar [1984a] and studied by Shanno [1985b].

A second option that we have considered but do not recommend is to discard all previous updates, and treat the current point as the first point; that is, to replace the collection of updates with a single update as if the current point had
been reached in one step from the initial point. One may also combine the two strategies, electing to perform an additional matrix factorization only when it is determined that progress is not being made. Using the pure strategy of discarding updates means that only the initial matrix is factored. This can yield the solution to the problem at extremely modest expense and using very limited storage, but convergence in this case is not guaranteed. We recommend the restarting strategy over the discarding strategy, both for its sound theoretical basis, as is elaborated in section 5.3, and for its superior performance in practice. Henceforth when we refer to “the modified algorithm,” the algorithm is assumed to employ periodic restarts. Use of the restarting strategy results in an algorithm that retains the polynomial worst-case time bound of the Karmarkar algorithm, since our step acceptance criterion, to be discussed below, requires reduction of the potential function by at least the constant amount at each step that is guaranteed for the Karmarkar algorithm, and clearly one can always get this reduction by restarting.

5.3 Linesearch and step acceptance criterion

Our implementation of the proposed algorithm includes a linesearch with a three-faceted step acceptance criterion as a way of selecting the steplength. We shall prove that the step acceptance criterion ensures that our modified algorithm with restarts retains the polynomial time bound of the unmodified Karmarkar algorithm. However, rather than restricting steps to have a predetermined fixed length, it allows taking longer steps when it appears advantageous to do so. We interpret failure to find an acceptable step after a specified number of trial steps as an indication that a restart is needed. Our first trial steplength is .99 of the distance to the edge of the simplex; our third trial steplength is .99 of the radius of the largest sphere that can be inscribed in the simplex, and our second trial steplength is midway between the first and third. Beginning with the fourth trial step, each is half as long as the previous trial step. Our step acceptance criterion is a combination of the Karmarkar criterion, the Goldstein-Armijo condition, and reduction of the linear objective function. The Karmarkar criterion, which is used to prove convergence of the algorithm in polynomial time, requires at least a constant amount of reduction in the potential function (2.7) at every step:

\[ f(e - a_\varepsilon p) \leq f(e) - \delta = n \log \varepsilon \tau z_\varepsilon - \delta, \]

(5.3.1)

where \( \delta \) is the minimum reduction in the potential function that is guaranteed for each step of the Karmarkar algorithm. The lower bound \( \delta \) on reduction in the potential function obtained by Karmarkar is dependent on the size of the problem as well as the steplength, and ensures reduction of the potential function at each step by at least .1 for a step in the direction \(-e_p\) of length equal to one-fourth the radius of the largest sphere that can be inscribed in the
simplex, provided the number of variables in the problem is at least 21. Since our test problems are larger than that, we use \( \delta = .1 \).

The Goldstein-Armijo condition (see, for example, Dennis and Schnabel [1983]), requires that the average rate of decrease in the potential function from \( \hat{x}_- \) to \( \hat{x}_+ \) is at least a prescribed fraction (\( \lambda \)) of the initial rate of decrease in that direction:

\[
\bar{f}(\hat{x}_+ \\leq \bar{f}(\hat{x}_-) + \lambda \nabla f(\hat{x}_-)^T(\hat{x}_- - \hat{x}_+) ;
\]

that is,

\[
f(x - \alpha \hat{x}_-) \leq n \log \frac{c^T x - \alpha \frac{\lambda n}{c^T x} (Dc)^T \hat{x}_- . (5.3.2)}{}
\]

In order to determine whether the trial step would result in a reduction of the linear function \( c^T x \), we consider the objective value of the image of the trial step in the untransformed space, and determine whether the trial step satisfies

\[
\frac{c^T x}{c^T D \hat{x}_-} < c^T x . (5.3.3)
\]

Polynomial-time complexity is ensured for the modified algorithm by requiring (5.3.1) to be satisfied at every step, as we formally state below. Our real objective, however, is minimization of the linear function \( c^T x \). While the linear function is not transformed to a linear function under the projective transformation, straight lines do map to straight lines under projective transformations. Hence the level sets of the linear function are mapped to flats in the transformed space, which do not intersect, since the projective transformation (2.4) is one-to-one. Therefore, if the direction we are considering gives descent on the untransformed linear function, a longer step in this direction will produce greater reduction in the linear objective function. Thus it seems sensible in this case to take the longest possible step in the computed direction that satisfies (5.3.1). In the event our direction is not a descent direction for the linear function, we guard against taking a step too long relative to the amount of reduction achieved in the potential function by requiring our step to satisfy (5.3.2) as well as (5.3.1). In summary, the step acceptance strategy that we use is to accept the step as soon as the trial step satisfies (5.3.1) and either (5.3.2) or (5.3.3). For an iteration at which a factorization has been done, if a longer step does not satisfy (5.3.1) and either (5.3.2) or (5.3.3), we accept a step in the direction \(-c_b\) of length equal to one-fourth the radius of the largest inscribed sphere. It will always satisfy (5.3.1). In our computational testing, the longest trial step was almost always taken, and it was never necessary to take a step as short as one-fourth the radius of the largest inscribed sphere.

We may now formally state that our modified algorithm with periodic restarts and the step acceptance strategy given above retains the polynomial
worst-case time bound of the unmodified Karmarkar algorithm.

Theorem 5.3.4

The modified Karmarkar algorithm that computes step directions as

\[ \hat{z}_p = D^{-1} \hat{\hat{D}} \left( \hat{\hat{D}}^T \hat{\hat{D}} \right)^{-1} \hat{\hat{D}}^T c, \]

where \( \hat{D} \) is any nonsingular approximation to \( D \) and \( \hat{\hat{D}} = BD^{-1} \hat{D} \), and that ensures at every step that

\[ \hat{j}(e - \alpha \hat{z}_p) \leq \hat{j}(e) - \delta, \quad \delta > 0 \]

finds a feasible point \( x_\delta \) such that

\[ \frac{\hat{e}^T x_\delta}{\hat{e}^T x_0} \leq 2^\delta \]

in \( O(n(q + \log n)) \) steps.

Proof.

This follows immediately from Theorem 2.8.

Todd and Burrell [1985] suggest choosing a steplength by approximately minimizing the potential function (2.7) in the computed direction. We have not taken this approach, placing emphasis instead on reduction of the untransformed linear objective function. However, examination of the behavior of the potential function in selected step directions computed in the solution of our test problems did not reveal a significant difference in the steplength chosen by our criteria and the steplength that would have been chosen by the strategy of approximately minimizing the potential function in this direction. An accurate computation of the minimizer of the potential function in the direction of the step would in some cases have resulted in taking a slightly shorter or a slightly longer step. Further discussion of this point is included in section 6.

5.4 Maintaining sparsity in the matrix to be factored

In transforming the constraints to homogeneous form, we appended one dense column and one dense row to the original (presumably sparse) constraint matrix \( \bar{A} \), obtaining

\[
A = \begin{bmatrix}
\bar{A} \\
-\bar{b} \\
0 \\
1 & \ldots & 1 & (1-B) & 1
\end{bmatrix}
\]

We may write

\[
A = \bar{A} + \hat{e} e_{k+1}^T + e_m \omega^T,
\]

where
\[ \begin{pmatrix} 0 & 0 \\ \bar{A} & 0 \\ \vdots & \vdots \\ 0 & 0 \end{pmatrix}, \quad \bar{b} = \begin{pmatrix} -b_1 \\ \vdots \\ -b_{m-2} \\ -b_{m-1} \end{pmatrix}, \quad \omega = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \]

and \( \epsilon_{n-1} \) and \( \epsilon_m \) are the \( n-1 \)st and \( m \)th unit vectors, respectively. The sparse matrix \( \bar{A} \) may then be used in the factorization, and rank-one corrections used to obtain solutions to linear systems with coefficient matrix \((\bar{A}D_0)(\bar{A}D_0)^T\) from solutions to linear systems with coefficient matrix \((\bar{A}D_0)(\bar{A}D_0)^T\), in the following way.

Let

\[ \begin{align*}
\nu_1 &= D_0 \bar{\epsilon}_{n-1}, \\
\nu_2 &= D_0 \bar{\omega}, \\
\nu_3 &= \bar{A}D_0 \nu_1, \\
\nu_4 &= \bar{A}D_0 \nu_2,
\end{align*} \]

and let \( \sigma_1 = \nu_1^T \nu_1 \), \( \sigma_2 = \nu_2^T \nu_2 \), and \( \sigma_3 = \nu_3^T \nu_3 \). Let

\[ \begin{align*}
\nu_5 &= \nu_3 + \sigma_1 \bar{\epsilon}_1 + \sigma_3 \bar{\epsilon}_m, \\
\nu_6 &= \nu_4 + \sigma_2 \bar{\epsilon}_1 + \sigma_3 \bar{\epsilon}_m, \\
\nu_7 &= [(\bar{A}D_0)(\bar{A}D_0)^T]^{-1} \nu_3,
\end{align*} \]

and let \( \sigma_4 = 1 + \bar{\epsilon}_1^T \nu_7 \). Now let

\[ \nu_8 = \left[ I - \frac{\nu_7 \bar{\epsilon}_1^T}{\sigma_4} \right] \left[ (\bar{A}D_0)(\bar{A}D_0)^T \right]^{-1} \nu_3 \]

and \( \sigma_5 = 1 + \bar{\epsilon}_1^T \nu_8 \). Let

\[ \nu_9 = \left[ I - \frac{\nu_8 \bar{\epsilon}_1^T}{\sigma_5} \right] \left[ I - \frac{\nu_7 \bar{\epsilon}_1^T}{\sigma_4} \right] \left[ (\bar{A}D_0)(\bar{A}D_0)^T \right]^{-1} \nu_3, \]

and let \( \sigma_6 = 1 + \bar{\epsilon}_1^T \nu_9 \). Finally, let

\[ \nu_{10} = \left[ I - \frac{\nu_9 \bar{\epsilon}_1^T}{\sigma_6} \right] \left[ I - \frac{\nu_8 \bar{\epsilon}_1^T}{\sigma_5} \right] \left[ I - \frac{\nu_7 \bar{\epsilon}_1^T}{\sigma_4} \right] \left[ (\bar{A}D_0)(\bar{A}D_0)^T \right]^{-1} \epsilon_m, \]

and \( \sigma_7 = 1 + \bar{\epsilon}_1^T \nu_{10} \). One may now solve linear systems with coefficient matrix \((\bar{A}D_0)(\bar{A}D_0)^T\) by using

\[ \left( (\bar{A}D_0)(\bar{A}D_0)^T \right)^{-1} = \left[ I - \frac{\nu_9 \bar{\epsilon}_1^T}{\sigma_7} \right] \left[ I - \frac{\nu_8 \bar{\epsilon}_1^T}{\sigma_6} \right] \left[ I - \frac{\nu_7 \bar{\epsilon}_1^T}{\sigma_5} \right] \left[ I - \frac{\nu_6 \bar{\epsilon}_1^T}{\sigma_4} \right] \left[ (\bar{A}D_0)(\bar{A}D_0)^T \right]^{-1}. \]

For ease of exposition, the above calculations were shown using ten additional vectors \( \nu_i \), \( 1 \leq i \leq 10 \), and seven scalars; one needs to retain in storage only the five vectors and four scalars appearing in (5.4.1). In the above, \( D_0 \) refers to any diagonal matrix encountered in the algorithm for which a factorization of \((\bar{A}D_0)(\bar{A}D_0)^T\) will be computed. For the modified algorithm using direction \( \bar{\epsilon}_1 \), given by (4.1), the vectors \( \nu_1, \ldots, \nu_{10} \) and the scalars \( \sigma_1, \ldots, \sigma_7 \) need be
computed only once each time a factorization is computed. If the approximation $\tilde{D}$ is being maintained using update vectors, one needs solutions of linear systems with coefficient matrix $(A\tilde{D}_1)(A\tilde{D}_1)^T$. These may be obtained by applying (5.1.3) or (5.1.4) or (5.1.5) to (5.4.1). The Karmarkar algorithm can be viewed as the modified algorithm with a new factorization computed at every step. Thus, using the unmodified Karmarkar algorithm with this strategy for maintaining sparsity, one computes the vectors $\nu_1, \cdots, \nu_{10}$ and the scalars $\sigma_1, \cdots, \sigma_7$ at each step.

The additional dense column appended to the coefficient matrix in the phase 1 problem is separated from the coefficient matrix in an analogous fashion in our implementation. This treatment could also be applied, although we have not done so, to separate dense columns appearing in the original constraint matrix.

5.5 Problems with special attributes

5.5.1 Bounds on the variables

A linear programming problem with bounded variables $l_i \leq x_i \leq u_i$, for some indices $i$ can, of course, be transformed into the canonical form given by (2.1) by treating these simple bound constraints the same as other inequality constraints, introducing slack variables to obtain equality constraints, and proceeding as in Section 2.

An alternative strategy is to deal with bounds implicitly in the algorithm in the choice of steplength. That is, to limit steps so that rather than restricting all components of each iterate to be positive, one is restricting iterates (in the untransformed space) to lie within the bounds. Although this approach has the benefit of not increasing the size of the problem, it has shortcomings that outweigh this benefit. The primary reason for not adopting this strategy is that the step direction computed in the algorithm would be an orthogonal projection onto a space that represents only a subset of the constraints, so that it is not clear that a feasible step could be taken in the computed direction. A second difficulty in taking this approach is that nonzero bounds in the untransformed space do not correspond to simple bounds in terms of transformed variables.

In our implementation, we have added slack variables to transform the upper bound constraints to equality constraints, and have translated the lower bounds to a requirement of nonnegativity by a shift of variables.

5.5.2 Fixed variables
Recall that the Karmarkar algorithm is an interior point method: it requires an initial point $x_0 > 0$, and maintains $x > 0$ for all subsequent iterates. A problem that includes variables fixed at 0 does not have an interior feasible point and cannot be solved using the Karmarkar algorithm. We have encountered problems that have this attribute, arising because the problems are cases generated using a general-purpose model-generating program. Our implementation includes pre-processing to remove all fixed variables from the problem. The values of the fixed variables are set aside, and are inserted into the solution.

5.5.3 Free variables

To transform a problem for which the original formulation includes variables that are unrestricted in sign to the canonical form (2.1), our implementation replaces free variables throughout the problem by the difference of two variables, both constrained to be nonnegative. This transformation is also carried out in the pre-processor.

5.5.4 Redundant constraints

The assumption of full rank of the constraint matrix is made in expressing the projection onto the nullspace of $B$ as $I - B^T(BB^T)^{-1}B$. If the constraint matrix is rank deficient, this condition must be detected, and redundant constraints set aside. Our implementation uses the Harwell factorization routine MA27 (see Duff and Reid [1982,1983]), which is a sparse frontal method variant of a Cholesky factorization with pivoting. Rank deficiency of $(AD)(AD)^T$, and hence of $AD$ is detected during factorization, with the offending rows conceptually having been pivoted to the end of the last block used. If $(AD)(AD)^T$ is rank deficient, we are able to recover from the indexing information of the subroutine the row numbers of redundant (or inconsistent) constraints. As discussed in section 5.4, the matrix used in the factorization does not include the right-hand side of the constraints in our sparse implementation, so that we are not certain at the time of detection that the offending rows are redundant. When rank deficiency is detected, we can determine only that a row or some rows are preventing the constraint matrix from having full rank. Thus, we set aside the offending rows, solve the problem without them, then test whether the solution satisfies the constraints that were set aside.

5.6 Solving problems with unknown optimal value

Todd and Burrell [1985] have contributed a modification of the Karmarkar algorithm that is applicable when the optimal value of the objective function is unknown, and that involves using estimates of the values of dual variables to
generate improving lower bounds on the optimal objective value. The lower bounds \( \bar{f}_k \) are then used to define an objective function as in (2.2) at each step. We shall show how a sequence of improving lower bounds on the optimal objective value can be obtained using the matrix factorization that is available in our modified Karmarkar algorithm. First, we discuss the construction of lower bounds as given by Todd and Burrell.

Recall that the primal problem under consideration is

\[
\begin{align*}
\text{minimize} & \quad \mathbf{c}^T \mathbf{x} \\
\text{subject to} & \quad \mathbf{A} \mathbf{x} = \mathbf{0} \\
& \quad \mathbf{e}^T \mathbf{x} = n \\
& \quad \mathbf{x} \geq 0,
\end{align*}
\]

where \( \mathbf{c}, \mathbf{x}, \mathbf{e} \in \mathbb{R}^n \), \( \mathbf{A} \in \mathbb{R}^{m \times n} \), and \( \mathbf{e} = (1, \cdots, 1 \cdots) \). The dual of this problem is

\[
\begin{align*}
\text{maximize} & \quad \mathbf{n}^T \mathbf{y} \\
\text{subject to} & \quad \mathbf{A}^T \mathbf{y} + \mathbf{e} \mathbf{z} \leq \mathbf{c},
\end{align*}
\]

Todd and Burrell note that for any \( \mathbf{y} \in \mathbb{R}^n \), \((y, z)\) with \( z = \min_j [\bar{f} - \mathbf{A}^T \mathbf{y}]_j \) is dual feasible with dual objective value \( nz \). They then show how to maintain dual estimates in such a way as to ensure that an algorithm using an objective function that is adjusted at each step.

\[
c_k = \mathbf{e} - \frac{\mathbf{l}}{n} \mathbf{e} = \mathbf{e} - \bar{z}_k \mathbf{e},
\]

and computing step directions as in the Karmarkar algorithm, with \( c_k \) replacing \( c \) \((= c_\star)\), will achieve at least a fixed reduction in the potential function

\[
f(x; c_k) = \sum_{i=1}^{n} \log \frac{\mathbf{c}_i^T \mathbf{x}}{\mathbf{x}}
\]

at each step. Their Lemma 5.1 states that reduction of the above potential function implies an equal reduction of the potential function

\[
f(x; c_\star) = \sum_{i=1}^{n} \log \frac{c_\star^T \mathbf{x}}{\mathbf{x}},
\]

where \( c_\star = \mathbf{e} - \frac{\mathbf{l}}{n} \mathbf{e} \), at each step, which establishes convergence of the modified Karmarkar algorithm. Todd and Burrell give a proof of convergence of the unmodified Karmarkar algorithm that uses the following pivotal lemma, to which we shall refer. To facilitate its subsequent application, we state the lemma using the objective function \( \mathbf{e}^T \mathbf{z} \).

Lemma 5.6.2 (Todd and Burrell [1985], Lemma 3.1)

Let \( d = -[I - B^T (BB^T)^{-1} B] D \bar{e} \) where \( B = \begin{bmatrix} \mathbf{A} \\ \mathbf{e}^T \end{bmatrix} \). Let

\[
y = ([A D \mathbf{(AD)}^T]^{-1} A) D \bar{e} \quad \text{and} \quad \mathbf{z} = \min_j (D \bar{e} - DA^T \mathbf{y})_j.
\]
Then
\[(D\tilde{e})^T \left( e + \alpha \frac{d}{\|d\|} \right) \leq \left| 1 - \frac{\alpha}{n} \right| (D\tilde{e})^T e + \frac{\alpha}{n} (n\tilde{F}).\]

Lemma (5.6.2) is used along with

Lemma (5.6.3) (Todd and Burrell [1985], lemma 3.4)

If \( \|x - e\| \leq \alpha \leq 1, e^T x = n, \) then
\[0 \leq \sum_{i=1}^{n} \log z_i \leq \frac{\alpha^2}{2(1-\alpha)}\]
to prove that a fixed reduction in the potential function (5.6.1) can be achieved at each step by a step in the direction \( d = -(I - B^T(BB^T)^{-1}B)d\tilde{e}. \)

Todd and Burrell [1985] show that convergence of their algorithm for solving problems with unknown optimal value is ensured if one obtains at the \( k \)th step

a feasible point \( z_k \)
\[y_k \in \mathbb{R}^m,\]
\[z_k = \min\{\gamma^T A^T y_k\}, \text{ such that } \gamma \leq \alpha \]
\[f(z_k,c_k) - f(z_k,c_k) \geq k\delta,\] (5.6.4)

and give the following construction. It will be useful to use the notation \( P_M \) to denote the projection onto the nullspace of \( M, P_M = I - MT(MMT)^{-1}M. \)

Construction of the Todd and Burrell estimates begins with
\[y_0 = [(A^T_0)(A^T_0)^T]^{-1}(A^T_0)D\tilde{e},\]
and
\[z_0 = \min\{\gamma^T A^T y_k\}, \] and for \( k = 0, 1, \ldots \) estimates of the dual variables \( y_{k+1} \) and \( z_{k+1} \) are obtained as follows. Let
\[u = P_{A^T_0}D\tilde{e},\]
and
\[v = P_{A^T_0}z_k.\]

If \( \min\{u - z_k v\} \leq 0, \) then set
\[y_{k+1} = y_k, \quad z_{k+1} = z_k, \quad \text{and } c_{k+1} = c_k.\]

Otherwise, find \( z_{k+1} \) with \( \min\{u - z_{k+1} v\} = 0, \) and set
\[c_{k+1} = z_{k+1} - z_{k+1}\gamma\]
and
\[y_{k+1} = [(A^T_0)(A^T_0)^T]^{-1}(A^T_0)D_{k+1}c_{k+1}.\]

In order to apply the Todd and Burrell strategy for dealing with problems with unknown optimal objective value to our modified algorithm, we give a construction of the estimates of the dual variables \( y_k \) and \( z_k \) appearing in (5.6.4) based on matrices that are available in our modified algorithm, and that are equal to the estimates given by Todd and Burrell [1985] for steps in which a
matrix factorization has been computed. In the following, we denote by \( \hat{D}_k \) the approximation at step \( k \) to the diagonal matrix with the components of the current iterate as its diagonal elements, so that at steps in which a matrix factorization is computed, \( \hat{D}_k = D_k \). We may succinctly give estimates of the dual variables obtained in our modified algorithm that are analogous to the Todd and Burrell estimates as follows. Let

\[
u = P_{AD_k} \hat{D}_k \hat{D}_k^T e,
\]

and

\[
v = P_{AD_k} \hat{D}_k^T e.
\]

If \( \min_j \{ D_k \hat{D}_k^{-T} (u - z_k v) \} \geq 0 \), then set

\[
y_{k+1} = y_k, \quad z_{k+1} = z_k, \quad \text{and} \quad c_{k+1} = c_k.
\]

Otherwise, find \( z_{k+1} \) with \( \min_j \{ D_k \hat{D}_k^{-T} (u - z_{k+1} v) \} = 0 \), and set

\[
c_{k+1} = \hat{F} - z_{k+1} e,
\]

and

\[
y_{k+1} = ([AD_k]([AD_k])^T)^{-1}(AD_k) \hat{D}_k^T c_{k+1}.
\]

We shall provide a constructive proof of the following theorem.

**Theorem 5.5.5**

The modified Karmarkar algorithm that computes step directions as

\[
e^{(k+1)} = D_k^{-1} \hat{D}_k \{ I - \hat{D}_k^T (\hat{B}_k \hat{B}_k^T)^{-1} \hat{B}_k \} \hat{D}_k^T e^{(k+1)},
\]

where \( \hat{D}_k \) is a nonsingular approximation to \( D_k \), \( \hat{B}_k = B_k D_k^{-1} \hat{D}_k \), and \( e^{(k+1)} = \hat{F} - z_{k+1} e \), where \( nz_{k+1} \) is a lower bound on the optimal value of \( \hat{F}^T x \) obtained as described above, and that ensures at every step that

\[
f((e - \alpha e^{(k+1)})c_{k+1}) \leq f(e; c_{k+1}) - \delta, \quad \delta > 0
\]

finds a feasible point \( x_j \) such that

\[
\frac{\varepsilon x_j}{\varepsilon x_0} \leq \varepsilon - \varepsilon
\]

in \( O(n(q + \log n)) \) steps.

Our constructive proof closely follows the construction of estimates of the dual variables given by Todd and Burrell [1985]. For \( k = 0 \), we will compute a matrix factorization, so that we begin with the same estimates used by Todd and Burrell:

\[
y_0 = ([AD_0]([AD_0])^T)^{-1}(AD_0)D_0 \hat{F}, \quad \text{and} \quad z_0 = \min_j (\hat{F} - A^T y_0).
\]

The fourth condition in (5.6.4) is trivially satisfied.
We show how to find \( x_{k+1} \), \( y_{k+1} \), and \( z_{k+1} \) satisfying (5.8.4). The sequence \( \{x_k\} \) computed using factorizations that are available in our modified algorithm will be monotonic by construction, corresponding to the sequence \( \{n_k\} \) of lower bounds on the optimal objective value.

Set 
\[
y = (A\tilde{D}_k)(A\tilde{D}_k)^T z A\tilde{D}_k \tilde{D}_k^T e_k,
\]
and \( \hat{z} = \min_j (\mathcal{F} - A^T y)_j \). There are two cases, depending on whether \( \hat{z} \leq z_k \).

**Case 1.**

Suppose first that \( \hat{z} \leq z_k \). Then
\[
\min_j (c_k - A^T y)_j = \min_j (\mathcal{F} - A^T y - z_i e) = \min_j (\mathcal{F} - A^T y) - z_k \leq 0,
\]
and
\[
\min_j (D_k c_k - D_k A^T y)_j \leq 0.
\]
In this case, we do not update the estimates. Set \( y_{k+1} = y_k \) and \( z_{k+1} = z_k \).

**Case 2.**

Now suppose that \( \hat{z} > z_k \), so that
\[
\min_j (c_k - A^T y)_j > 0.
\]
Then
\[
\min_j (D_k \hat{D}_k^T \hat{D}_k (c_k - A^T y))_j > 0.
\]
Using projection notation, the above can be written
\[
\min_j (D_k \hat{D}_k^T P_{A^T}(c_k - A^T y))_j > 0.
\]
Now let \( u = P_{A^T}\hat{D}_k^T \) and \( v = P_{A^T}\hat{D}_k^T e \) to write the above as
\[
\min_j (D_k \hat{D}_k^T (u - z_i v))_j > 0.
\]
Let \( z = \frac{t^T x_k}{n} \geq z_k \), and observe that
\[
\epsilon^T D_k \hat{D}_k^T (u - z e) = \epsilon^T D_k \hat{D}_k^T P_{A^T}(\hat{D}_k^T \mathcal{F} - z \hat{D}_k^T e) = (P_{A^T}\hat{D}_k^T D_k \epsilon)^T (\hat{D}_k^T \mathcal{F} - z \hat{D}_k^T e) = (\hat{D}_k^T D_k \epsilon)^T (\hat{D}_k^T \mathcal{F} - z \hat{D}_k^T e) = \epsilon^T D_k (\mathcal{F} - z e) = \mathcal{F}^T x_k - nz = 0.
\]
Thus, \( \min_j [D_k \hat{D}_k^T (u - zv)]_j \leq 0 \), since the sum of its components vanishes. It follows that there is some \( z_{k+1} \) with \( z_k < z_{k+1} \leq z \) such that
\[
\min_j [D_k \hat{D}_k^T (u - z_{k+1}v)]_j = 0.
\]
Choose this value as \( z_{k+1} \), let \( c_{k+1} = \tau - z_{k+1} \varepsilon \), and define
\[
y_{k+1} = [(A \hat{D}_k (A \hat{D}_k)^T)^{-1} A \hat{D}_k A^T] e_{k+1}.
\]
Now observe that
\[
\min_j [D_k (c_{k+1} - A^T y_{k+1})]_j
= \min_j \left\{ D_k \left[ \hat{D}_k^T - A^T \left[ (A \hat{D}_k (A \hat{D}_k)^T)^{-1} A \hat{D}_k A^T \right] \hat{D}_k^T \right] e_{k+1} \right\}_j
= \min_j [D_k \hat{D}_k^T P_{A \hat{D}_k} \hat{D}_k^T e_{k+1}]_j
= \min_j [D_k \hat{D}_k^T (u - z_{k+1}v)]_j
= 0.
\]
Since \( D_k \) is a diagonal matrix with all diagonal elements positive, it follows that
\[
\min_j [\hat{D}_k^T (u - z_{k+1}v)]_j = 0,
\]
which is to say that
\[
\min_j (c_{k+1} - A^T y_{k+1})_j = 0; \quad (5.6.6)
\]
i.e.,
\[
\min_j (\tau - A^T y_{k+1} - z_{k+1} \varepsilon) = 0.
\]
Hence,
\[
z_{k+1} = \min_j (\tau - A^T y_{k+1}).
\]
Now, having obtained an estimate \( z_{k+1} \) from either Case 1 or Case 2, we set \( c_{k+1} = \tau - z_{k+1} \varepsilon \) and use \( c_{k+1}^T x \) as the objective function, which is to say that we compute the step direction using
\[
\hat{e}_k = D_k^{-1} \hat{D}_k (I - \hat{B} \hat{B}^T)^{-1} \hat{B}^T e_{k+1}.
\]
We attempt to take a step in the computed direction, imposing the step acceptance criterion given in Section 5.3, as we did for problems with known objective values. Again, we interpret failure to find an acceptable step after a fixed number of trial steps as an indication that a restart is needed. Our modified algorithm for problems with unknown optimal value consists of using estimates \( \{z_k\} \) as discussed above, and using the restarting strategy and the step acceptance criterion of Section 5.3. Convergence in polynomial time of our modified algorithm for problems with unknown optimal value is ensured, since we can always achieve the desired reduction in the potential function by restarting if necessary. For steps in which a factorization has been computed.
the estimates $y_i$ and $z_i$ are identical to those constructed by Todd and Burrell, and the direction $\hat{z} = [I-B^T(BB^T)^{-1}B]D_{c_4+1}$. For these steps, the desired reduction in the potential function, and hence our acceptance of the step, is ensured by applying lemma 5.6.2 as they do. For an estimate arising from Case 1, lemma 5.6.2 can be applied with $\hat{z} = c_4$, giving $T \leq 0$. For an estimate arising from Case 2, we see from (5.6.6) that

$$\min_j (D_{c_4+1} - D_{c_4}A^T_{y_{k+1}})j = 0,$$

so that lemma 5.6.2 can be applied with $\hat{z} = c_{4+1}$, giving $T = 0$.

6. Numerical results

Preliminary testing using an experimental implementation of the modified algorithm has given very encouraging computational results. We have used eight test problems. The first is a problem with one hundred seven variables and sixty-seven constraints that we obtained from Shell Development Company; the second through eighth are test problems from the Systems Optimization Laboratory at Stanford University that we obtained through netlib (see Gay [1985b]). The number of variables shown for each problem includes slack variables introduced to transform inequality constraints to equalities, but does not include the two additional slack variables discussed in Section 2. Results obtained solving these problems using various options in the modified algorithm are summarized in Tables 6.1 through 6.8. For each problem, a feasible starting point $x_0$ was obtained using the phase I technique discussed in Section 2, from the initial point $(1, \cdots, 1)^T$. All options were then run from the same feasible starting point. The number of steps to convergence shown in the tables does not include steps required to obtain the starting point. The first option shown for each problem is restarting every step, which means that the Karmarkar algorithm, modified only by the incorporation of a linesearch, is used. In each instance, restart after $k$ updates means that a new factorization is computed after saving at most $k$ updates; that is, at least as often as every $k + 1$ steps. Although our restarting strategy calls for computing a new factorization earlier
whenever the linesearch is unsuccessful, this situation did not arise in our testing. Similarly, restart after k steps (using no updates, but retaining the same approximation to $D$ between factorizations) means restart after at most $k$ steps. To test the modified algorithm against the Karmarkar algorithm, we solved the test problems to only limited accuracy. We expect that relative times needed to compute more accurate solutions would be similar to those given in Tables 6.1 through 6.8. For all problems, the stopping criterion was $e^T x \leq 10^{-3} e^T x_0$. This gave three to four digits of accuracy in the optimal objective value for all of the test problems except ISRAEL, where only one digit of accuracy in the optimal objective value was obtained. Subsequently, we continued the solution of ISRAEL. We obtained four digits of accuracy in the optimal objective value at the cost of twelve additional steps and factorizations using the Karmarkar algorithm, seventeen steps with nine factorizations restarting after one update, twenty-eight steps with seven factorizations restarting after three updates, and thirty-four steps with six factorizations restarting after five updates.

In Tables 6.1 through 6.8 the update used is identified as follows:

<table>
<thead>
<tr>
<th>update number</th>
<th>updating formula</th>
<th>right-hand side of secant equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(4.10)</td>
<td>$y_1$ of (4.6)</td>
</tr>
<tr>
<td>2</td>
<td>(4.17)</td>
<td>$y_1$ of (4.5)</td>
</tr>
<tr>
<td>3</td>
<td>(4.17)</td>
<td>$y_1$ of (4.7)</td>
</tr>
<tr>
<td>4</td>
<td>(4.10)</td>
<td>$y_1$ of (4.7)</td>
</tr>
<tr>
<td>5</td>
<td>(4.10)</td>
<td>$y_1$ of (4.8)</td>
</tr>
<tr>
<td>6</td>
<td>(4.17)</td>
<td>$y_1$ of (4.8)</td>
</tr>
</tbody>
</table>

For options involving the use of updates, we show only the results obtained using the most successful updating formula. In general, better performance was achieved using updating formulas 1, 4 and 5, the least change updates to $D$ based on the barrier function, the potential function, and on the true $D$, respectively, than was achieved using updating formulas 2, 3 and 6, the updates that result from the strategy of least change to the Hessians (i.e. least change to $D^{-1}$) of the functions used to obtain the approximations. The most successful updating formula was the simplest, the least change update to $D$ based on the true $D$, identified as update 5. In cases where two or more updating formulas produced identical results in number of steps and factorizations needed to obtain the solutions, times given are for updating formula 5.

The times shown for solving the test problems were obtained on a Pyramid 90x, using the Unix operating system OSx version 2.5. Times are the sum of
CPU times attributed to the user and to the operating system in seconds, rounded to the nearest second. The last column of each table contains normalized times, the ratio of the time required to obtain the solution using the given option to the time required using the Karmarkar algorithm (with linesearch).

For each of the test problems, use of the modified algorithm with periodic restarts, and using rank-one updates to approximate $D$ at each step, results in obtaining the solution with (generally but not monotonically) progressively fewer matrix factorizations as the number of updates allowed between factorizations increases. However, the number of iterations required to obtain the solution increases as the number of factorizations decreases. Timings on the test problems indicate an overall reduction in the amount of work can be achieved using the modified algorithm, compared to the Karmarkar algorithm, for all problems except our smallest test problem, AFIRO (Table 6.2). Times are accurate only to within about one second, so that little significance should be given to relative times for solving this very small problem that takes only two to three seconds. The largest percentages of savings in computational effort appear, as one would expect, in the larger problems, ISRAEL (Table 6.7) and BRANDY (Table 6.8).

The strategy of periodic restarts with no updates used may be advantageous if only a few steps are taken between factorizations. The increase in number of steps required is generally greater, and the number of factorizations saved is generally smaller, than occurs when update vectors are used, but the computation of steps that require neither factorizations nor updates is very inexpensive.

<table>
<thead>
<tr>
<th>Test problem: SHELL: 107 variables, 87 constraints</th>
<th>Option</th>
<th>Update</th>
<th>Steps</th>
<th>Factorizations</th>
<th>Time</th>
<th>Time/Time(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>every step</td>
<td>8</td>
<td>8</td>
<td>17</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>after 1 update</td>
<td>5</td>
<td>12</td>
<td>6</td>
<td>16</td>
<td>.94</td>
<td></td>
</tr>
<tr>
<td>after 2 updates</td>
<td>5</td>
<td>14</td>
<td>5</td>
<td>15</td>
<td>.88</td>
<td></td>
</tr>
<tr>
<td>after 4 updates</td>
<td>5</td>
<td>18</td>
<td>4</td>
<td>18</td>
<td>.94</td>
<td></td>
</tr>
<tr>
<td>after 9 updates</td>
<td>5</td>
<td>28</td>
<td>3</td>
<td>21</td>
<td>1.24</td>
<td></td>
</tr>
<tr>
<td>after 1 step</td>
<td>12</td>
<td>7</td>
<td>12</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>after 2 steps</td>
<td>19</td>
<td>7</td>
<td>16</td>
<td>1.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>after 3 steps</td>
<td>21</td>
<td>7</td>
<td>20</td>
<td>1.18</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1
### Test problem: AFIR0: 51 variables, 27 constraints

<table>
<thead>
<tr>
<th>Option</th>
<th>Update</th>
<th>Steps</th>
<th>Factorizations</th>
<th>Time</th>
<th>Time/Time(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>every step</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>after 1 update</td>
<td>1, 4, 5</td>
<td>11</td>
<td>6</td>
<td>3</td>
<td>1.50</td>
</tr>
<tr>
<td>after 2 updates</td>
<td>5</td>
<td>12</td>
<td>5</td>
<td>3</td>
<td>1.50</td>
</tr>
<tr>
<td>after 3 updates</td>
<td>5</td>
<td>16</td>
<td>4</td>
<td>3</td>
<td>1.50</td>
</tr>
<tr>
<td>after 7 updates</td>
<td>5</td>
<td>24</td>
<td>3</td>
<td>5</td>
<td>2.50</td>
</tr>
<tr>
<td>after 1 step</td>
<td>11</td>
<td>6</td>
<td>3</td>
<td>1.50</td>
<td></td>
</tr>
<tr>
<td>after 2 steps</td>
<td>16</td>
<td>6</td>
<td>3</td>
<td>1.50</td>
<td></td>
</tr>
<tr>
<td>after 3 steps</td>
<td>21</td>
<td>6</td>
<td>4</td>
<td>2.00</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2

### Test problem: ADLITTLE: 138 variables, 56 constraints

<table>
<thead>
<tr>
<th>Option</th>
<th>Update</th>
<th>Steps</th>
<th>Factorizations</th>
<th>Time</th>
<th>Time/Time(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>every step</td>
<td>12</td>
<td>12</td>
<td>22</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>after 1 update</td>
<td>1</td>
<td>15</td>
<td>8</td>
<td>10</td>
<td>0.88</td>
</tr>
<tr>
<td>after 2 updates</td>
<td>5</td>
<td>18</td>
<td>5</td>
<td>17</td>
<td>0.77</td>
</tr>
<tr>
<td>after 4 updates</td>
<td>5</td>
<td>23</td>
<td>5</td>
<td>20</td>
<td>0.91</td>
</tr>
<tr>
<td>after 7 updates</td>
<td>5</td>
<td>30</td>
<td>4</td>
<td>22</td>
<td>1.00</td>
</tr>
<tr>
<td>after 1 step</td>
<td>15</td>
<td>8</td>
<td>17</td>
<td>0.77</td>
<td></td>
</tr>
<tr>
<td>after 2 steps</td>
<td>23</td>
<td>8</td>
<td>10</td>
<td>0.88</td>
<td></td>
</tr>
<tr>
<td>after 3 steps</td>
<td>30</td>
<td>8</td>
<td>21</td>
<td>0.95</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3

### Test problem: SHARE2B: 162 variables, 96 constraints

<table>
<thead>
<tr>
<th>Option</th>
<th>Update</th>
<th>Steps</th>
<th>Factorizations</th>
<th>Time</th>
<th>Time/Time(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>every step</td>
<td>9</td>
<td>9</td>
<td>26</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>after 1 update</td>
<td>5</td>
<td>14</td>
<td>7</td>
<td>24</td>
<td>0.92</td>
</tr>
<tr>
<td>after 3 updates</td>
<td>5</td>
<td>22</td>
<td>6</td>
<td>28</td>
<td>1.08</td>
</tr>
<tr>
<td>after 6 updates</td>
<td>5</td>
<td>33</td>
<td>5</td>
<td>35</td>
<td>1.38</td>
</tr>
<tr>
<td>after 1 step</td>
<td>18</td>
<td>8</td>
<td>27</td>
<td>1.04</td>
<td></td>
</tr>
<tr>
<td>after 2 steps</td>
<td>22</td>
<td>8</td>
<td>29</td>
<td>1.12</td>
<td></td>
</tr>
<tr>
<td>after 5 steps</td>
<td>41</td>
<td>7</td>
<td>35</td>
<td>1.35</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.4
### Test problem: SHARE1B: 253 variables, 117 constraints

<table>
<thead>
<tr>
<th>Option</th>
<th>Update</th>
<th>Steps</th>
<th>Factorizations</th>
<th>Time</th>
<th>Time/Time(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>restart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>every step</td>
<td>19</td>
<td>19</td>
<td>248</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>after 1 update</td>
<td>5</td>
<td>25</td>
<td>198</td>
<td>.80</td>
<td></td>
</tr>
<tr>
<td>after 2 updates</td>
<td>5</td>
<td>31</td>
<td>175</td>
<td>.71</td>
<td></td>
</tr>
<tr>
<td>after 3 updates</td>
<td>5</td>
<td>34</td>
<td>158</td>
<td>.64</td>
<td></td>
</tr>
<tr>
<td>after 4 updates</td>
<td>5</td>
<td>39</td>
<td>155</td>
<td>.63</td>
<td></td>
</tr>
<tr>
<td>after 5 updates</td>
<td>5</td>
<td>81</td>
<td>222</td>
<td>.90</td>
<td></td>
</tr>
<tr>
<td>after 6 updates</td>
<td>5</td>
<td>49</td>
<td>183</td>
<td>.86</td>
<td></td>
</tr>
<tr>
<td>after 7 updates</td>
<td>5</td>
<td>58</td>
<td>165</td>
<td>.87</td>
<td></td>
</tr>
<tr>
<td>after 8 updates</td>
<td>5</td>
<td>37</td>
<td>193</td>
<td>.78</td>
<td></td>
</tr>
<tr>
<td>after 2 steps</td>
<td>44</td>
<td>12</td>
<td>190</td>
<td>.77</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.5

### Test problem: BEACONFD: 293 variables, 173 constraints

<table>
<thead>
<tr>
<th>Option</th>
<th>Update</th>
<th>Steps</th>
<th>Factorizations</th>
<th>Time</th>
<th>Time/Time(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>restart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>every step</td>
<td>9</td>
<td>9</td>
<td>273</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>after 1 update</td>
<td>1,4,5</td>
<td>13</td>
<td>227</td>
<td>.83</td>
<td></td>
</tr>
<tr>
<td>after 2 updates</td>
<td>5</td>
<td>17</td>
<td>215</td>
<td>.79</td>
<td></td>
</tr>
<tr>
<td>after 3 updates</td>
<td>5</td>
<td>24</td>
<td>200</td>
<td>.73</td>
<td></td>
</tr>
<tr>
<td>after 4 updates</td>
<td>5</td>
<td>38</td>
<td>212</td>
<td>.78</td>
<td></td>
</tr>
<tr>
<td>after 5 updates</td>
<td>5</td>
<td>47</td>
<td>227</td>
<td>.83</td>
<td></td>
</tr>
<tr>
<td>after 6 updates</td>
<td>5</td>
<td>13</td>
<td>222</td>
<td>.81</td>
<td></td>
</tr>
<tr>
<td>after 7 updates</td>
<td>19</td>
<td>7</td>
<td>232</td>
<td>.85</td>
<td></td>
</tr>
<tr>
<td>after 2 steps</td>
<td>24</td>
<td>7</td>
<td>241</td>
<td>.88</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.6
### Table 6.7

<table>
<thead>
<tr>
<th>Option</th>
<th>Update</th>
<th>Steps</th>
<th>Factorizations</th>
<th>Time</th>
<th>Time/Time(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>every step</td>
<td>11</td>
<td>11</td>
<td>398</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>after 1 update</td>
<td>5</td>
<td>15</td>
<td>8</td>
<td>323</td>
<td>.81</td>
</tr>
<tr>
<td>after 2 updates</td>
<td>5</td>
<td>17</td>
<td>6</td>
<td>259</td>
<td>.65</td>
</tr>
<tr>
<td>after 3 updates</td>
<td>5</td>
<td>19</td>
<td>5</td>
<td>232</td>
<td>.58</td>
</tr>
<tr>
<td>after 7 updates</td>
<td>5</td>
<td>31</td>
<td>4</td>
<td>226</td>
<td>.57</td>
</tr>
<tr>
<td>after 8 updates</td>
<td>5</td>
<td>41</td>
<td>3</td>
<td>248</td>
<td>.62</td>
</tr>
<tr>
<td>after 1 step</td>
<td>16</td>
<td>8</td>
<td>325</td>
<td>.82</td>
<td></td>
</tr>
<tr>
<td>after 2 steps</td>
<td>22</td>
<td>8</td>
<td>332</td>
<td>.83</td>
<td></td>
</tr>
<tr>
<td>after 4 steps</td>
<td>30</td>
<td>7</td>
<td>312</td>
<td>.78</td>
<td></td>
</tr>
</tbody>
</table>

### Table 6.8

<table>
<thead>
<tr>
<th>Option</th>
<th>Update</th>
<th>Steps</th>
<th>Factorizations</th>
<th>Time</th>
<th>Time/Time(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>every step</td>
<td>12</td>
<td>12</td>
<td>276</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>after 1 update</td>
<td>1.5</td>
<td>17</td>
<td>9</td>
<td>223</td>
<td>.81</td>
</tr>
<tr>
<td>after 2 updates</td>
<td>5</td>
<td>20</td>
<td>7</td>
<td>189</td>
<td>.68</td>
</tr>
<tr>
<td>after 3 updates</td>
<td>5</td>
<td>24</td>
<td>6</td>
<td>175</td>
<td>.64</td>
</tr>
<tr>
<td>after 8 updates</td>
<td>5</td>
<td>32</td>
<td>5</td>
<td>175</td>
<td>.64</td>
</tr>
<tr>
<td>after 10 updates</td>
<td>5</td>
<td>41</td>
<td>4</td>
<td>179</td>
<td>.65</td>
</tr>
<tr>
<td>after 1 step</td>
<td>20</td>
<td>10</td>
<td>244</td>
<td>.68</td>
<td></td>
</tr>
<tr>
<td>after 2 steps</td>
<td>25</td>
<td>9</td>
<td>229</td>
<td>.83</td>
<td></td>
</tr>
<tr>
<td>after 3 steps</td>
<td>33</td>
<td>9</td>
<td>240</td>
<td>.67</td>
<td></td>
</tr>
</tbody>
</table>

Figures 6.9 through 6.12 show the behavior of the untransformed linear objective function and of the transformed potential function in the direction of selected steps in the transformed space taken in the solution of the test problems. In each figure, the variable on the horizontal axis is the steplength parameter \( \alpha \), ranging from zero on the left to a value of \( \alpha \) corresponding to a step 50% of the distance to the edge of the simplex for the last point plotted on the right. The scale shown on the left vertical axis is for values of the untransformed linear objective function, shifted to have a minimum value of zero, and the scale shown on the right vertical axis is for values of the potential...
function. In nearly all steps of all test cases, the longest trial step was accepted.

The behavior of the linear and potential functions in the step directions was similar to that shown in Figures 6.9 and 6.10, arising in the solution of AFIRO and of SHARE1B, respectively, using the Karmarkar algorithm modified only by the incorporation of a linesearch. Figures 6.11 and 6.12 show two of the more interesting directions where the longest step was not taken. Figure 6.11 shows selected steps in the solution of BEACONFD, using the modified algorithm with one step computed using updates after each step requiring a factorization. Selected steps in the solution of BRANDY using two update steps between factorizations are shown in Figure 6.12. At step 8 in this solution of BEACONFD and at step 15 in this solution of BRANDY, the second-longest step, midway between the edge of the simplex and the edge of the largest inscribed sphere, was taken. At all other steps in the solution of these cases, the behavior of the linear and potential functions in the step directions was similar to that shown for steps 2, 5, and 11 for BEACONFD and for steps 7, 11, and 19 in the solution of BRANDY, and the longest step was accepted. In all cases, the steplength selected was close to the steplength that would have been chosen by the strategy of approximately minimizing the potential function in the step direction. In each of Figures 6.9 through 6.12, the steplength that was selected is indicated by a vertical line in the graph.
Linear(*) and Potential (o) Function Values in Step Directions for SHAREI8

Step 3

Step 8

Step 9

Figure 6.10

Step 10

Step 11

Figure 6.11
7. Directions for further research

Some details of implementation of the Karmarkar algorithm and the variable-metric variant that has been proposed remain to be explored. In the current implementation, there remain opportunities to save computing time through more efficient arrangement of the existing calculations. In addition, alternative strategies for implementing parts of the algorithm are of interest. For example, in the current implementation, the matrix $(AD)(AD)^T$, or $(\tilde{A}D)(\tilde{A}D)^T$, with $\tilde{A}$ as defined in Section 5.4, is formed at each step requiring a matrix factorization, and is factored. We shall use $\tilde{A}$ to denote either $AD$ or $\tilde{A}D$. We would like to investigate computing instead a factorization of a matrix of the form

$$\begin{bmatrix} I & \tilde{A}^T \\ \tilde{A} & 0 \end{bmatrix}$$

(7.1)

This matrix differs from a matrix used by Duff and Reid [1975] in the least-squares solution of sparse overdetermined systems of linear equations only in that $\tilde{A}$ and $\tilde{A}^T$ are interchanged. If $LU$ factorization of (7.1) is performed without pivoting, one obtains

$$\begin{bmatrix} I & \tilde{A}^T \\ \tilde{A} & 0 \end{bmatrix} = \begin{bmatrix} I & 0 \\ \tilde{A} & L_{22} \end{bmatrix} \begin{bmatrix} I & \tilde{A}^T \\ 0 & U_{22} \end{bmatrix},$$

(7.2)

where $L_{22}U_{22} = -\tilde{A}\tilde{A}^T$. We were inclined to form the matrix $\tilde{A}\tilde{A}^T$ at each
step requiring a matrix factorization, since we interpreted (7.2) as indicating that
the work involved in forming and factoring this matrix would be done if we
chose to factor (7.1). However, the work of Duff and Reid [1975], which has only
recently come to our attention, indicates that an efficient exploitation of sparsity
in factoring (7.1) may result in a saving of computation, and we would like to
investigate this possibility.

Another area that remains for investigation is the establishment of
conditions on the approximation \( \hat{D} \) as used in the variable-metric variant of the
algorithm that would ensure polynomial-time complexity for an algorithm with
steps computed by (4.1), independent of a restart strategy. This would be of
theoretical interest, even though it appears that for the present form of our
modified algorithm it is most practical to ensure polynomial-time complexity
through the step acceptance criterion and restart strategy as we have done.
Certainly some restart strategy must be employed in any algorithm that saves
update vectors. We may wish to consider carrying out updates rather than
saving update vectors, however.

A special case of a very difficult problem arises when one considers a
modified Karmarkar algorithm that uses approximations in a manner similar to
(4.1), but rather than saving update vectors, carries out updating to maintain an
approximation. Such an updating practice would only be reasonable if sparsity
were preserved by the update. One could consider approximating, as Karmarkar
[1984] does, \( D \) by a diagonal matrix, or approximating \( D^2 \) by a diagonal matrix,
or directly maintaining an approximation \( (A\hat{D})(A\hat{D})^T \) with the same sparsity
pattern as \( AA^T \). In any of these cases, one would want an approximation that
was sparse, symmetric, and positive-definite. Many researchers have tried,
without success, to develop a reasonable way of updating sparse Hessian
approximations to preserve sparsity, symmetry, and positive-definiteness.
Dennis and Vu [1983] approached this problem by considering updating the
Cholesky factor of an approximation to a Hessian directly, using sparse Broyden
updates. They encountered difficulty in deciding on a secant or secant-like
condition to use in defining the update. It is possible that consideration of the
particular approximation that is needed in the context of the Karmarkar
algorithm will lead to a reasonable secant or secant-like condition for this case,
so that sparse updates can be used successfully in the approximation.
REFERENCES


COLMENARES, OCTAVIO [1985]. Karmarkar's linear programming algorithm, better or worse than the classical method? Instituto Tecnologico Autonomo de Mexico.


GAY, DAVID M. [1985b]. Electronic mail distribution of linear programming test problems Committee on Algorithms Newsletter, Mathematical Programming Society, No. 13, December.


PICKEL, P.F. [1985]. Approximate projections for the Karmarkar algorithm. I. Theory. Polytechnic Institute of New York, Farmingdale, N.Y.


