COMPUTING KARMAKAR'S PROJECTIONS QUICKLY USING MATRIX FACTORIZATION

John R. Birge
Hengyong Tang

Department of Industrial and Operations Engineering
The University of Michigan
Ann Arbor, MI 48109-2117

Technical Report 93-24
August 1993
Computing Karmarkar’s Projections Quickly Using Matrix Factorization

John R. Birge
Department of Industrial and Operations Engineering
The University of Michigan
Ann Arbor MI 48109 U.S.A.

Hengyong Tang
Mathematics Department Liaoning University
Shenyang Liaoning 110036 P.R.China

August 18, 1993

Abstract

In this paper we compute Karmarkar’s projections quickly using Moore-Penrose g-inverse and matrix factorization. So computation work of \((A^T D^2 A)^{-1}\) is decreased.

Keywords: Linear Programming, Karmarkar’s Algorithm, Karmarkar’s Projection, Moore-Penrose G-inverse, Matrix Factorization.

1 Introduction

In 1984, Karmarkar [1] proposed a new polynomial time algorithm for linear programming based on repeated projective transformation. This algorithm creates a series of interior points converging to the optimal solution. His work has sparked tremendous interest and inspired others to modify his algorithm or to investigate similar methods for linear programming. A lot of variations of Karmarkar’s algorithm have been made.

In all variations of Karmarkar’s algorithm, the major work is repeated computation of \((A^T D^2 A)^{-1}\) or solution of system of linear equations

\[ A^T D^2 A y = b. \]

Since \(D\) changes at each iteration. To decrease computation work of Karmarkar’s algorithm computing \((A^T D^2 A)^{-1}\) quickly is needed. Many papers do their best to compute \((A^T D^2 A)^{-1}\) quickly.
using rank-one update, approximate scaling matrix, updated Cholesky factorization and so on [1] [2] [3] [4], [5] [6] have given efficient solution for block-angular linear programming, especially for two-stage stochastic linear programming.

In this paper we decrease the work of computing \( (A^T D^2 A)^{-1} \) using Moore-Penrose g-inverse and matrix factorization.

In section 2, we review some characters of the Moore-Penrose g-inverse. Karmarkar’s algorithm is formulated in section 3. In section 4, we give method for computation of \( (A^T D^2 A)^{-1} \) using Moore-Penrose g-inverse and matrix factorization. The complexity of the algorithm is analysed in section 6.

2 Moore-Penrose g-inverse

In this section, we review some characters of the Moore-Penrose generalized inverse (g-inverse) [7] [8] which are useful in following sections.

We consider real \( m \times n \) matrix \( A \). Let \( A^+ \) be the Moore-Penrose g-inverse of \( A \).

Character 1  a) \( A^+ A A^T = A^T A A^+ = A^T \).

b) \( A A^T (A^T)^+ = (A^T)^+ A^T A = A \).

c) \( (A^T A)^+ = A^+ (A^T)^+ \).

d) \( (A A^T)^+ = (A^T)^+ A^+ \).

Character 2  a) If \( \text{rank}(A) = n \), then

\[
A^+ = (A^T A)^{-1} A^T.
\]

b) If \( \text{rank}(A) = m \), then

\[
A^+ = A^T (A A^T)^{-1}.
\]

Character 3 Let \( A = B C \), where \( A \), \( B \) and \( C \) be \( m \times n \), \( m \times r \) and \( r \times n \) matrices respectively, and

\[
\text{rank}(A) = \text{rank}(B) = \text{rank}(C) = r,
\]

then

\[
A^+ = C^+ B^+.
\]

3 Karmarkar’s algorithm

For the purpose of discussion, we consider linear programming problem without generality

\[
\begin{align*}
\text{max} & \quad c^T x \\
\text{s.t} & \quad Ax \leq b
\end{align*}
\]
where $c$ and $x$ are $n$-vectors, $b$ is an $m$-vector and $A$ is a full rank $m \times n$ matrix, where $m \geq n$ and $c \neq 0$. Problem (1) has an interior feasible solution $x^o$.

We focus on a variation of Karmarkar's algorithm which is generally called the dual affine scaling method [2].

**Algorithm 1** $(A,b,c,x^o,\text{stopping criterion},0 < \gamma < 1)$

1. $k=0$.
2. **stop if optimality criterion is satisfied.**
3. $\nu^k = b - Ax^k$.
4. $D^k = \text{diag}\{1/\nu^k_1, \ldots, 1/\nu^k_m\}$.
5. $h_x = (A^T(D^k)^2A)^{-1}c$.
6. $h_\nu = -Ah_x$.
7. $\alpha = \gamma \times \min\{-\nu^k_i/(h_\nu)_i|(h_\nu)_i < 0, i = 1, \ldots, m\}$.
8. $x^{k+1} = x^k + \alpha h_x$.
9. $k = k + 1$, goto 2.

The major computation work of the algorithm is computing $(A^T(D^k)^2A)^{-1}$.

Since $D^k$ is $m \times m$ full rank diagonal matrix, then rank$(D^kA) = n$, and

$$(D^kA)^+ = ((D^kA)^T D^kA)^{-1}(D^kA)^T,$$

$$(D^kA)^+((D^kA)^+)^T = (A^T(D^k)^2A)^{-1}(D^kA)^T D^kA (((D^kA)^T D^kA)^T)^{-1}$$

$$= (A^T(D^k)^2A)^{-1}.$$ 

Thus step 5 of algorithm 1 can be written as

$$h_x = (D^kA)^+((D^kA)^+)^Tc.$$ 

In following section we will discuss computing $(D^kA)^+$ quickly. For simplicity, we write $D^k$ as $D$. In general we let rank$(A) = r \leq n$.

4 Matrix factorization

**Proposition 1** Let $D$ be a full rank diagonal matrix, $A$ be a $m \times n$ matrix, and $A = BC$, where $B$ is a full rank $m \times r$ matrix, $C$ is a full rank $r \times n$ matrix, then

$$(DA)^+ = C^+(DB)^+$$

$$= C^T(B^T D^2 B)^{-1} B^T D.$$ 

3
proof. Since $D$ is a full rank square matrix, so $\text{rank}(DA) = r$ and $\text{rank}(B) = r$. According Character 3 and Character 2, We get needed result. This completes the proof of the proposition.

Using Gauss elimination method, we factorize $A$ into $A = LU$, where $L$ is a $m \times r$ unit lower trapezoidal matrix, $U$ is a $r \times n$ full rank upper trapezoidal matrix. Thus

$$(DA)^+ = U^T(UU^T)^{-1}(LTD^2L)^{-1}LTD.$$  

$(UU^T)^{-1}$ does not change at each iteration. If $r = n$, $U$ is full rank upper triangular matrix, we have

$$(DA)^+ = V^{-1}(LTD^2L)^{-1}LTD.$$  

So major work of computing $(DA)^+$ is the computation of $(LTD^2L)^{-1}$.

We partition $L$ into

$$L = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix}$$

where $L_1$ is a $r \times r$ unit lower triangular matrix, $L_2$ is a $(m-r) \times r$ matrix. Correspondingly we partition $D$ into

$$D = \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix}$$

where $D_1$ is $r \times r$ full rank diagonal matrix, $D_2$ is $(m-r) \times (m-r)$ diagonal matrix. Thus

$$LT^2DL = \begin{pmatrix} L_1^TD_1L_1^T & L_1^TD_2L_2 \\ 0 & L^TD_2^TL_2 \end{pmatrix} \begin{pmatrix} D_1L_1 \\ D_2L_2 \end{pmatrix}$$

$$= L_1^TD_1^2L_1 + L_1^TD_2^2L_2$$

$$= L_1^TD_1^2L_1 + \sum_{i=1}^{m-r} d_{r+i}^2 l_i^T, \quad (2)$$

where

$$D_2 = \text{diag} \{ d_{r+1}, \ldots, d_m \},$$

$$L_2^T = (l_1, \ldots, l_{m-r}).$$

$L_1^TD_1^2L_1$ is already Cholesky factorization form. A Cholesky factorization of $LT^2DL$ can be computed using a series of rank-one updates by (2).

To compute a Cholesky factorization of $LT^2DL$, $m - r$ rank-one updates are needed. But $m$ rank-one updates have to be done to compute a Cholesky factorization of $A^TD^2A$ in general Karmarkar's algorithm [3]. So computation work is decreased. As in [3], rank-one update can be done by Fletcher and Powell [9].
5 Complexity of algorithm

Proposition 2 If computation of \((A^T(Dk)^2A)^{-1}\) is done by LU factorization, then Algorithm 1 solves Problem (1) in no more than

\[ mn^2 + O(m^{2.5}(m - n)R) \]

arithmetic operations, where \(R\) is a measure of the problem’s size.

proof. The major operation work of the algorithm is:

\[ \text{LU factorization needs} \]

\[ mn^2 - (m + n)n^2/2 + n^3/3 = O(mn^2) \]

arithmetic operations [11].

Algorithm 1 finds an optimal solution to problem (1) in \(O(\sqrt{n}R)\) iterations [12] [13].

\((m - n)\) rank-one updates need \(O(m^2(m - n)R)\) arithmetic operations.

So algorithm 1 solves problem (1) at most \(mn^2 + O(n^{2.5}(m - n)R)\) arithmetic operations. This completes the proof of the proposition.

In general, \(R\) is very large. So

\[ O(m^3R) >> m^2n + O(m^2(m - n)R), \]

especially, when \(m - n\) is small.

Using approximation \(x^k, \nu^k\), the complexity of algorithm 1 can be reduced [12] [13].

References


