Revista de Matemática: Teoría y Aplicaciones 2(2): 35–48 (1995)

ALGORITMO DE KARMARKAR Y MATRICES RALAS

Juan Félix Avila Herrera[1]

Resumen

Este es el segundo de una serie de dos artículos en los que se estudia el método de Karmarkar. Se muestra cómo utilizar la teoría de matrices ralas para obtener una implementación eficiente del proceso de Karmarkar, presentado en el primer artículo. En la fase I del proceso de Karmarkar, se pone en evidencia la forma como se incrementa el tamaño de la matriz de restricciones tecnológicas. La nueva matriz sin embargo, posee una estructura peculiar bastante favorable, debido a la presencia de bloques de ceros que la hacen parte de una familia de matrices bastante conocida, a saber las matrices ralas. Se discute en este trabajo algunas técnicas para manejar este tipo de matrices, y finalmente el autor propone una variante del método de Karmarkar que aprovecha dicha situación.

Abstract

This is the second of a series of two articles in wich we study the Karmarkar's method. In this article we are going to show how can we use sparse matrix theory to get an effcient implementation of the Karmarkar's process presented in the first article. In phase I of the Karmarkar's process, it was evident how the size of the technological matrix increased. However, the new matrix has a special structure in wich we observed the presence of zero's blocks that make it a sparse matrix. We will discuss here some techniques to be used with this kind of matrix. Finaly we propose a Kamarkar's variant that takes advantage of this situation.

1. Resumen del método de Karmarkar

A continuación se mencionan algunos de los aspectos más importantes sobre el método de Karmarkar explicado en [2]. Este método permite resolver programas lineales en tiempo polinomial, mejorando en este sentido el consabido algoritmo del símplex. Es recomendable en todo caso consultar [2] para tener una mejor visión de lo que se expondrá seguidamente.

El método de Karmarkar consiste en tres fases que se resumen a continuación

1. (Fase I: Convirtiendo un programa lineal a la forma de Karmarkar) En esta fase se convierte un programa lineal dado en la forma estándar

$$\text{Maximizar} \{ z = cx : Ax \geq b,\ x \geq 0 \}, \tag{1}$$

[1]Escuela de Informática, Universidad Nacional

a la forma estándar de Karmarkar (FEK):

$$\text{Minimizar} : z = cx, \quad \text{sujeta a:} \quad Ax = 0,\ \mathcal{U}_n x = 1,\ x \geq 0, \tag{2}$$

donde A es $m \times n$ de rango m, $n \geq 2$, c y A están compuestos por números enteros, $\mathcal{U}_n$ es un vector con n unos, y cumple las siguientes condiciones **(S1)** el punto $x_0 = \left(\frac{1}{n}, \cdots, \frac{1}{n}\right)$, es factible, y **(S2)** el valor objetivo óptimo del problema es cero.

Al convertir (1) a la (FEK) obtenemos:

$$\begin{array}{l} \text{Minimizar} : z = x'_{2(m+n)+1} \\ \text{sujeta a:} \left\{ \begin{array}{l} H'x' = 0, \\ \displaystyle\sum_{i=1}^{2(m+n+1)} x'_i = 1, \\ x'_i \geq 0, \quad \text{para} \quad 1 \leq i \leq 2(m+n+1). \end{array} \right. \end{array} \tag{3}$$

en donde

$$H' := \left[\begin{array}{cc|cc|c|c} A_{m\times n} & -I_m & \mathcal{Z}_{m\times(m+n)} & & \alpha_{m\times 1} & -b_{m\times 1} \\ \mathcal{Z}_{n\times(m+n)} & & A^T_{n\times m} & I_n & \beta_{n\times 1} & -c_{n\times 1} \\ c_{1\times n} & \mathcal{Z}_{1\times m} & -b_{1\times m} & \mathcal{Z}_{1\times n} & \gamma & 0 \end{array}\right],$$

con

$$\begin{array}{l} \alpha = b + \mathcal{U}_m - A\,\mathcal{U}_n, \\ \beta = c - \mathcal{U}_n - A^T\,\mathcal{U}_m, \\ \gamma = \sum_{i=1}^{m} b_i - \sum_{i=1}^{n} c_i \end{array}$$

Como se observó en [2] H' es una matriz rala.

2. (Fase 2: Resumen del Algoritmo de Karmarkar)

El siguiente es el resumen del algoritmo de Karmarkar que aplica solo para programas lineales que están bajo la (FEK).

a) Iniciación: $(k = 0)$

1) Calcule el radio $r = 1/\sqrt{n(n-1)}$.

2) Calcule $L = \lceil 1 + \log(1 + |c_{j\,\text{máx}}|) + \log(|\det_{\text{máx}}|)\rceil$. Aquí $\lceil\cdot\rceil$, significa la parte entera superior y $\log(\cdot)$ denota logaritmo en base 2, además $|c_{j\,\text{máx}}| = \max\limits_{1\leq j\leq n} \{\,|c_j|\,\}$, y la cantidad $|\det_{\text{máx}}|$ es

$$|\det_{\text{máx}}| = \text{máx}\{\,|\det(B)| : B \text{ es una base para el problema (2)}\,\}.$$

La constante L se llama la *longitud de entrada* del problema (2). Una forma de estimar L es utilizar $\tilde{L}$ en donde:

$$\tilde{L} := \lceil 1 + \log(1 + |c_{j\,\text{máx}}|) + \log(1+m) + \sum_{i=1}^{m}\sum_{j=1}^{n} \log(1 + |a_{ij}|)\rceil, \tag{4}$$

y se satisface que $L \leq \tilde{L}$.

3) Calcule $\alpha = (n-1)/3n$.
4) Coloque $x_0 = (1/n, \cdots, 1/n)$.

b) ELEGIR SOLUCIÓN: Si $cx_k < 2^{-L}$, la solución actual x_k es factible y satisfactoria. Parar el algoritmo.

c) PASO PRINCIPAL: Se definen las siguientes cantidades matriciales:

1) La matriz $D_k = \text{diag}\{x_{k1}, \cdots, x_{kn}\}$, en donde $x_k = (x_{k1}, \cdots, x_{kn})$.
2) La matriz $P = \begin{bmatrix} AD_k \\ \mathcal{U}_n \end{bmatrix}$.
3) El vector $\bar{c} = cD_k$.

Se calcula entonces x^* como $x^* = x_0 - \alpha r \frac{c_p}{||c_p||}$, con $c_p = \left[I - P^T(PP^T)^{-1}P\right]\bar{c}^T$.

De esta forma se obtiene $x_{k+1} = (D_k x^*)/(\mathcal{U}_n D_k x^*)$. Incremente k en uno y regrese al paso ELEGIR SOLUCIÓN.

3. (Fase 3: Rutina de redondeo optimal) La solución x_k obtenida en la Fase 2, no necesariamente es un punto extremo. Es por esto que se debe "redondear" x_k a una solución óptima, de suerte que tenga tan buen valor objetivo como x_k. A continuación se resume la forma de lograr esto.

a) ELEGIR SOLUCIÓN: Si al sustituir x_k en el programa lineal (2), n restricciones linealmente independientes están siendo satisfechas en forma activa, se concluye entonces que x_k es una solución básica óptima. Parar.

b) CALCULAR DIRECCIÓN: Si x_k no es una solución de punto extremo, existen varias restricciones (digamos $l < n$) del problema (2) que x_k satisface en forma activa. Denotemos con $\tilde{A}$ la matriz formada a partir de esas restricciones, de la siguiente forma:

1) Si una restricción de $Ax = b$ se satisface en forma activa con x_k, se incluye la fila correspondiente de A en $\tilde{A}$.
2) Si $\mathcal{U}_n x_k = 1$, el vector $\mathcal{U}_n = (1, 1, \cdots, 1)$ se incluye en $\tilde{A}$.
3) Si la entrada i-ésima de x_k es nula, se agrega $e_i = (0, 0, \cdots, 1, \cdots, 0)$ (con un único 1 en la i-ésima posición) a $\tilde{A}$.

La matriz $\tilde{A}$ resultante es entonces de tamaño $l \times n$ con $l < n$. Considere ahora el sistema $\tilde{A}x = 0$. y sea d una solución no trivial de este sistema.

c) ACTUALIZAR SOLUCIÓN: Se obtiene una nueva solución x_k^*, moviéndose a lo largo de la dirección d si $cd < 0$ y a lo largo de $-d$ en otro caso[2], hasta que algunas restricciones bloqueén cualquier movimiento adicional por consideraciones de factibilidad. Así:

1) si $cd < 0$, y todas las entradas de d son no negativas, se concluye que la solución es no acotada. En el caso contrario $x_k^* = x_k + \lambda \cdot d$, en donde $\lambda = \min_{1 \le i \le n} \{-x_{ki}/d_i : d_i < 0\}$, en donde $x_k = (x_{k1}, x_{k2}, \cdots, x_{kn})$ y $d = (d_1, d_2, \cdots, d_n)$.

[2]En virtud de la fase I, basta con considerar el signo de $d[n-1]$.

2) si $cd \geq 0$, y todas las entradas de d son no positivas, se concluye que la solución es no acotada. En el caso contrario

$$x_k^* = x_k - \lambda \cdot d,$$

en donde

$$\lambda = \min_{1 \leq i \leq n} \{ x_{ki}/d_i : d_i > 0 \}.$$

Se regresa entonces al paso ELEGIR SOLUCIÓN.

2. Matrices ralas y eliminación Gaussiana

Cuando una matriz posee "muchas" entradas nulas, se advierte la necesidad de caracterizarlas y buscar estructuras de datos que permitan manejar esta clase especial de matrices de una forma apropiada.

Definición 1 La *densidad* $\delta(A)$ de una matriz A se define como el cociente entre el número de entradas no nulas y el tamaño total de la matriz.

Definición 2 Se dice que una matriz A es ϵ-*rala* para $0 \leq \epsilon \leq 1$, si su densidad $\delta(A)$ satisface $\delta(A) \leq \epsilon$.

De esta forma se dirá que una matriz es *rala* si es ϵ-rala para ϵ cercano a cero. Más concretamente, se aceptará como rala una matriz A (por lo menos en el presente trabajo), si $\delta(A) \leq 1/3$.

Se utilizarán dos estructuras de datos distintas para representar una matriz rala según convenga. Estas ideas fueron tomadas de [7].

1. **(EU)**: diseñada para matrices ralas de características generales, y que se empleará para almacenar la matriz U en su descomposición L–U. La estructura **(EU)** la conforman una lista de vectores { **A**, Indr, Lonr, Locr }, donde

   a) **A** contiene las entradas no nulas de U, listadas por filas no necesariamente ordenadas.

   b) Indr[i] indica la columna en la que se halla la i-ésima entrada no nula de U almacenada en **A**[i]. Por conveniencia se supondrá que Indr[i] = 0 si la entrada **A**[i] está "vacía" o disponible.

   c) Lonr[i] indica la longitud (en término de entradas no nulas) de la fila i.

   d) Locr[i] indica la entrada de **A** en donde empieza la fila i. Si la i-ésima fila de U es nula se coloca Locr[i] en cero.

   Es importante notar que los elementos no nulos de la i-ésima fila son Lonr[i] en total, y que se hallan ubicados en las posiciones

$$\mathbf{A}[\,\mathrm{Locr}[i]], \cdots A[\,\mathrm{Locr}[i] + \mathrm{Lonr}[i] - 1],$$

del vector **A**.

(Ejemplo) Una representación **(EU)** de la matriz

$$U = \begin{bmatrix} 0 & 2 & 3 \\ 1 & 9 & 4 \\ 8 & 4 & 7 \end{bmatrix}$$

está dada en la Fig. 1. La entradas 3 y 4 del vector **A** se consideran "vacías" o disponibles.

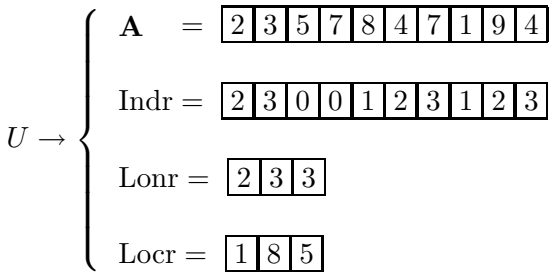

Figura 1: Ejemplo de una representación **(EU)**.

2. **(EL)**: diseñada para almacenar matrices ralas del tipo triangular inferior, complementando de esta forma, la tarea asignada a la **(EU)** en la descomposición L–U. La estructura **(EL)** está constituida por una lista de vectores de la forma { **A**, Indc, Indr }, que representan una matriz L como sigue:

   a) **A** contiene la entradas no nulas ubicadas debajo de la diagonal de L.

   b) La entrada $\mathbf{A}[i]$ se halla en la posición (Indc[i], Indr[i]) de L.

3. Factorización L–U de una matriz rala

Se tienen ya las estructuras necesarias para almacenar la descomposición L–U de una matriz rala A de tamaño $n \times n$. Este proceso consiste en factorizar la matriz A como $A = LU$, en donde U es una matriz triangular superior, mientras que L está expresada como un producto de transposiciones [3] y matrices de tipo especial. Formalizamos estas ideas en la siguiente definición.

Definición 3 Sean A, L y U matrices de tamaño $n \times n$. L y U son una *descomposición* o *factorización* L–U de A si $A = LU$, y satisfacen que:

   a) U es una matriz triangular superior y

[3] Una transposición es una permutación de dos filas de la matriz identidad.

b) $L = (P_1M_1)(P_2M_2)(P_3M_3)\cdots(P_rM_r)$, en donde $r \leq n(n-1)/2$, P_i es una transposición y cada factor M_k tiene la forma

$$M_k = I - \mu_k e_{i_k} e_{j_k}^T, \qquad k = 1, 2, \cdots, r, \tag{5}$$

donde e_{i_k} y e_{j_k} son vectores unitarios, $i_k \neq j_k$, y a los escalares μ_k se les llama *multiplicadores*.

Con el fin de mantener un balance entre estabilidad numérica y la preservación de la estructura rala se exigirá que los multiplicadores dados en (5) estén acotados de acuerdo con $|\mu_k| \leq \bar{\mu}$, para un $\bar{\mu} \geq 1$. Típicamente un valor de $\bar{\mu} = 10$ logra este propósito. Recuérdese del álgebra lineal estándar que una potencia de $\bar{\mu}$ aparece en la cota de crecimiento sobre los elementos de la factorización L–U.

El algoritmo que se construirá para obtener la descomposición L–U de una matriz A de tamaño [4] $n \times n$, está basado en la siguiente factorización:

$$\begin{bmatrix} \alpha & v^T \\ \beta & w^T \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\mu & 1 \end{bmatrix} \begin{bmatrix} \alpha & v^T \\ 0 & \bar{w}^T \end{bmatrix}, \tag{6}$$

en donde $\mu = -\beta/\alpha$ ($\alpha \neq 0$, $\beta \neq 0$) y $\bar{w} = w + \mu v$. La ecuación (6), permite hallar la descomposición L–U de la matriz A, factorizando progresivamente dos de sus filas. Los vectores (α, v^T) y (β, w^T) son del mismo tamaño con una longitud menor o igual a n. Normalmente (β, w^T), por ejemplo, está formado por entradas de una determinada fila de A, tomada a partir de su primer elemento no nulo β.

La idea para factorizar la matriz A es sencilla, y se expone someramente a continuación:

1. Inicie $L := I$, $U := A$ y coloque $P_1 = P_2 = \cdots = P_r = I$.

2. Al pretender formar un escalón [5] en la fila 2 se tienen varias posibilidades:

   a) Si $A[2,1] \neq 0$ y $A[1,1] \neq 0$, se emplea la factorización (6) para "hacer" un cero en $A[2,1]$. Defina entonces $M_1 = I - \mu e_2 e_1^T$ y reemplace la segunda fila de A, a saber (β, w^T) por $(0, \bar{w}^T)$.

   b) Si $A[2,1] = 0$, defínase entonces $M_1 = (I - 0e_2e_1^T) = I$.

   c) Si $A[2,1] \neq 0$ pero $A[1,1] = 0$, se efectúa un intercambio entre las filas 1 y 2, que se registra permutando las filas 1 y 2 de P_1. Se llega, por tanto, al caso (b) anterior.

   Colóquese entonces $L := L \cdot (P_1M_1)$.

[4] La ideas son también válidas para matrices rectangulares.

[5] Entiéndase por hacer un escalón en la fila i, al proceso de hacer ceros las entradas

$$A[i,1], \cdots, A[i,i-1],$$

utilizando para ello operaciones elementales sobre las filas precedentes.

3. Para formar un escalón en la fila 3 se utilizan las filas 1 y 2, siguiendo las mismas ideas presentadas en el punto interior. Este proceso arroja dos nuevos factores M_2 y M_3. Se define entonces $L := L \cdot (P_2 M_2) \cdot (P_3 M_3)$.

4. Se procede de igual forma con las filas restantes. Observe que el número de factores M_k que se obtiene es $1 + 2 + 3 + \cdots + (n-1) = n(n-1)/2$.

A continuación se describen las 10 subrutinas principales que intervienen en la solución del sistema $Ax = b$, mediante la factorización L–U. Los detalles de cada uno de estos procedimientos se hallan en [1].

1. **Comprimir**: se encarga de compactar el vector **A** de la estructura **(EU)** para procurar un uso apropiado del espacio. Esta rutina debe utilizarse con moderación. Si un porcentaje significativo [6] del espacio reservado en memoria principal ha sido usado, la utilización de **Comprimir** es justificable, de otro modo, sencillamente se estaría adrede retardando el algoritmo que lo invoque.

2. **CRF** (**c**orrer **r**englón al **f**inal): se encarga de trasladar "físicamente" el i-ésimo renglón de A (en la estructura **(EU)**) hasta el final del vector **A**. Esta rutina es necesaria puesto que al efectuar operaciones elementales sobre un renglón, éste puede incrementar su tamaño (en término de entradas no nulas), siendo necesario trasladarlo hasta el final del vector **A** donde puede crecer "ilimitadamente [7]".

3. **Apuntar**: Para la i-ésima fila de la matriz A (representada mediante la estructura **(EU)**), la subrutina **Apuntar** se encarga de anotar en un vector n-dimensional v, las posiciones de **A** en las que se encuentran sus entradas no nulas, i.e. se carga v de la siguiente forma:

$$v[j] = \begin{cases} l & \text{si} \quad A[i,j] = \mathbf{A}[l],\ (\mathbf{A}[l] \neq 0,\ l > 0); \\ 0 & \text{en otro caso.} \end{cases}$$

   Al tratar de formar un escalón en la i-ésima fila de A, en la eliminación Gaussiana, el vector v permite escudriñar dicha fila en busca de la próxima entrada no nula.

4. **Valor_U**: Este procedimiento se encarga de retornar la posición $U[i,j]$ de una matriz U, dada en la forma **(EU)**.

5. **Valor_L**: Este procedimiento se encarga de retornar la posición $L[i,j]$ de una matriz L, dada en la forma **(EL)**.

6. **Agregue**: Se encarga de agregar un nuevo factor M_k a L. Las transposiciones P_k se manejan mediante un vector R que se explicará después.

[6] Un 90 % puede considerarse razonable.

[7] En realidad el crecimiento está limitado por la cantidad de memoria reservada o disponible para **A**, ya sea que se use programación *estática* o *dinámica*.

7. **Escalón**: Se encarga de formar un escalón en una fila cualquiera de la matriz U. Tal fila se llama la *espiga*. **Escalón** es la rutina clave de la descomposición L–U.

   Es pertinente decir algunas palabras sobre las permutaciones. Cuando se trata de formar un escalón en una fila de una matriz, algunas veces se tropieza con 2 tipos de dificultades:

   a) La cantidad $\mu = -\beta/\alpha$, no puede ser calculado pues $\alpha = 0$.

   b) El *test* de estabilidad falla, i.e. $|\mu| > 10$.

   En tales casos es necesario intercambiar la fila espiga y la que se estaba usando para formar un escalón en una entrada de ésta. Tal operación es registrada por dos vectores que llamaremos P y R. El n-vector P es una permutación, ésto quiere decir que si $P[i] = j$, entonces se intercambiaron las filas i y j. R, por otro lado es un r-vector (con $r = n(n-1)/2$) que se encarga de registrar las transposiciones. Cada entrada $R[i]$ tiene dos componentes que se llamaran respectivamente $R[i].f$ y $R[i].g$. Antes de invocar el procedimiento **Escalón** se inicia el vector R como

$$R = \left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \cdots, \begin{bmatrix} r \\ r \end{bmatrix} \right). \tag{7}$$

   Si en el cálculo de el i-ésimo factor de L, se deben intercambiar las filas j y k, se modifica R como sigue $R[i].f := j$, y $R[i].g := k$. Se puede utilizar un tercer componente $R[i].p$, para almacenar solamente la entradas significativas del R propuesto en (7).

8. **Factorizar**: Este procedimiento obtiene la factorización L–U de una matriz, al aplicar $(n-1)$ veces la subrutina **Escalón**. Si se producen permutaciones de filas, éstas se anotan (usando P y R) con el fin de resolver cualquier sistema que utilice esta descomposición. El lector debe notar que en el caso particular de la matriz $\tilde{P}$, dada en (14), se puede empezar la eliminación de Gauss a partir de la fila $1 + 2(m + n + 1)$.

9. **Evaluar**: Si se define $\tilde{L}$ como la matriz triangular inferior dada por [8]

$$\tilde{L} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \mu_1 & 1 & 0 & \cdots & 0 \\ \mu_2 & \mu_3 & 1 & \cdots & 0 \\ \vdots & & & & \vdots \\ \mu_s & \mu_{s+1} & \mu_{s+2} & \cdots & 1 \end{bmatrix},$$

   Evaluar se encarga de regresar $U[i,j]$ si $i \leq j$, y $\tilde{L}[i,j]$ en caso contrario.

10. **Resolver**: Esta rutina se encarga de resolver un sistema de la forma $Ax = b$, utilizando su descomposición L–U. Si la matriz A en cuestión es singular, se produce

[8] Es fácil ver que $s = 1 + (n-1)(n-2)/2$.

un mensaje de error y se detiene el proceso. De acuerdo con la Def. 3

$$A = LU = (P_1M_1)(P_2M_2)(P_3M_3)\cdots(P_rM_r)\cdot U,$$

en donde $r = n(n-1)/2$. De esta forma para resolver el sistema original $Ax = b$, será suficiente resolver primero $Lz = b$ y después $Ux = z$. El sistema $Lz = b$, se puede escribir explícitamente como:

$$(P_1M_1)(P_2M_2)(P_3M_3)\cdots(P_rM_r)z = b. \tag{8}$$

El sistema (8) se puede resolver en $r \leq n(n-1)/2$ pasos. Veamos el primero para comprobar esta aseveración. Nótese que (8) se puede poner como

$$(P_2M_2)(P_3M_3)\cdots(P_rM_r)z = M_1^{-1}P_1^{-1}b.$$

Para calcular $M_1^{-1}P_1^{-1}b$ se debe efectuar primero la transposición $P_1^{-1} = P_1$, sobre el vector b. Esto transforma b en un nuevo vector $b^1 = P_1b$. Se nota ahora que

$$M_1^{-1}P_1^{-1}b = (I + \mu_1 e_2 e_1^T)b^1 = b^1 + \mu_1 e_2 e_1^T b^1,$$

o bien

$$M_1^{-1}P_1^{-1}b = b^1 + (\mu_1 b_1^1)e_2 = \begin{bmatrix} b_1^1 \\ b_2^1 + \mu_1 b_1^1 \\ \vdots \\ b_n^1 \end{bmatrix}.$$

Es fácil generalizar este paso. En el paso k-ésimo ($M_k = (I + \mu_k e_s e_t^T)$) se debe entonces efectuar primero la transposición P_k, obteniendo así un nuevo vector b^k. Se actualiza posteriormente la s-ésima entrada de b^k mediante $b_s^k := b_s^k + \mu_k b_t^k$. La solución de $Ux = z$ se obtiene mediante la conocida técnica de *sustitución hacia atrás* o también llamada *sustitución regresiva*, consulte por ejemplo [5].

4. Una implementación eficiente del algoritmo de Karmarkar

En la Fase I del proceso de Karmarkar (consulte [2]) se puso en evidencia cómo la matriz tecnológica H' del programa lineal transformado, tenía una estructura bastante rala. De hecho si $\delta(A)$ es la densidad de la matriz $A_{m\times n}$ del problema original

$$\text{Maximizar}\,\{\, cx : Ax \geq b,\ x \geq 0\,\},$$

es posible estimar la densidad de la matriz H'. El resultado se presenta en la siguiente proposición.

Proposición 1 La densidad de la matriz tecnológica H', del programa lineal transformado (presentado en el primer artículo de esta serie de tres), satisface

$$\delta(H') \leq \frac{2mn\delta(A) + 4m + 4n + 1}{2(m+n+1)^2}. \tag{9}$$

DEMOSTRACIÓN: El resultado se obtiene simplemente contando el número de entradas no nulas en la matriz H' dada en (3).

En en efecto, nótese que si $\delta(A) = k/mn$, en donde k es el número de entradas no cero de A, entonces $k = mn\delta(A)$, de esta forma k' (el total de entradas no nulas de H') satisface

$$k' \leq mn\delta(A) + m + m + m + mn\delta(A) + n + n + n + n + m + 1,$$

puesto que en el peor caso b, c, α, β son vectores sin entradas nulas y $\gamma \neq 0$. Dado que H' es de tamaño $(m + n + 1) \times 2(m + n + 1)$, se concluye fácilmente que la desigualdad (9) es correcta. □

Es fácil verificar numéricamente como disminuye la densidad de H' conforme aumenta el tamaño de la matriz A. Esto hace muy conveniente el uso de estructuras de datos (como **(EU)** y **(EL)**) que sean sensibles a tal fenómeno. Se debe notar también que si $m = n$ entonces

$$\delta(H') \leq \frac{2n^2\delta(A) + 8n + 1}{2(2n + 1)^2},$$

por tanto

$$\lim_{n\to+\infty} \delta(H') \leq \lim_{n\to+\infty} \frac{2n^2\delta(A)}{8n^2},$$

de esta forma, en el peor caso ($\delta(A) = 1$) se tiene:

$$\lim_{n\to+\infty} \delta(H') \leq 0{,}25.$$

De hecho, aún en el caso $m < n$, la desigualdad de Cauchy $2mn \leq m^2 + n^2$ da $4mn \leq (m + n)^2 < (m + n + 1)^2$, así que

$$\delta(H') \leq \frac{2mn\delta(A) + 4m + 4n + 1}{2(m + n + 1)^2} < \tfrac{1}{4}\delta(A) + \frac{2}{m + n + 1}.$$

De esta forma para, m y n "grandes", $\delta(H')$ es aproximadamente $1/4$.

El análisis anterior nos deja concluir que utilizando estructuras de datos como **(EU)** y **(EL)** se ahorra un porcentaje significativo de memoria principal, permitiendo de esta forma, manejar problemas de mayor tamaño. Hay sin embargo un paralogismo sobre el que se debe advertir. El hecho de que la densidad disminuya conforme el tamaño de problema lineal aumente, no garantiza de ninguna manera que cualquier programa lineal sea manejable, pues la matriz H' puede ser virtualmente "gigantesca", agotando inexorablemente la memoria principal. La ventaja de utilizar estructuras como **(EU)** y **(EL)** reside en que no se almacena complementamente la matriz rala, sino sus entradas no nulas. Sin embargo el 25 % de una matriz "gingantesca" puede tal vez no caber en memoria principal.

Se demuestra así que, el enfoque sobre el proceso de Karmarkar adoptado por esta investigación teóricamente rinde frutos. En [1] se muestra como estas cavilaciones funcionan bien en la práctica, además de hacer análisis de tiempos de ejecución

Se debe observar de la fase II del proceso de Karmarkar, que el esfuerzo computacional está dominado por el cálculo del vector c_p (la proyección ortogonal de c) dado por

$$c_p = [I - P^T(PP^T)^{-1}P]\bar{c}^T,$$

(consúltese el primer artículo de esta serie). Debemos entonces, encontrar la inversa de PP^T o bien resolver un sistema lineal de la forma [9]

$$(PP^T)x = P\bar{c}. \tag{10}$$

Sustituyendo [10] $P = \begin{bmatrix} AD \\ \mathcal{U}_n \end{bmatrix}$, se obtiene

$$PP^T = \begin{bmatrix} AD^2A^T & AD\mathcal{U}_n \\ (AD\mathcal{U}_n)^T & \mathcal{U}_n\mathcal{U}_n \end{bmatrix} = \begin{bmatrix} AD^2A^T & 0 \\ 0 & n \end{bmatrix}. \tag{11}$$

(Es importante hacer énfasis aquí en que la matriz A que se menciona en P, es la matriz H' de la ecuación 3).

Nótese de (11) que la matriz PP^T no es necesariamente rala, aún cuando A y consecuentemente P lo sean. Un manejo despreocupado en la solución de (10), echaría por tierra el ahorro de espacio en memoria principal que se ha venido pregonando, además de que obligaría a diseñar rutinas para efectuar operaciones elementales con matrices utilizando las estructuras **(EU)** y **(EL)**.

Se debe notar que el sistema (10) es equivalente a

$$\begin{bmatrix} I & P^T \\ P & 0 \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} \bar{c} \\ 0 \end{bmatrix}. \tag{12}$$

En efecto, nótese que (12) es equivalente a

$$\begin{cases} y + P^Tx = \bar{c}, \\ Py = 0, \end{cases}$$

de esta forma

$$y = \bar{c} - P^Tx, \tag{13}$$

y por tanto al multiplicar ambos miembros de (13) por P, se obtiene $0 = Py = P\bar{c} - PP^Tx$, que es justamente (10). Es justo aclarar que esta idea (ecuación (12)), fue tomada de [6].

Es pertinente hacer algunos comentarios sobre este enfoque. Lo primero es que se evita diseñar algoritmos para efectuar productos matriciales utilizando las estructuras de datos **(EL)** y **(EU)**. Esto reduce el tiempo computacional. Por otro lado, la matriz

$$\tilde{P} := \begin{bmatrix} I & P^T \\ P & 0 \end{bmatrix}, \tag{14}$$

hereda la estructura rala de P, y ésto es algo deseable. De hecho si $\delta(A)$ es la densidad de la matriz $A_{m\times n}$ del problema original Maximizar $\{\, cx : Ax \geq b,\ x \geq 0 \,\}$, es posible estimar la densidad del la matriz $\tilde{P}$. El resultado se resume en la siguiente proposición.

[9] Obśervese que esta es la "ecuación normal" para el problema de *mínimos cuadrados*: hallar x que minimiza $||P^Tx - \bar{c}^T||$. Véase [8].

[10] Se usará por comodidad D en lugar de D_k.

Proposición 2 La densidad de la matriz $\tilde{P}$ dada en (14) satisface

$$\delta(\tilde{P}) \leq \frac{4mn\delta(A) + 14m + 14n + 8}{(3m + 3n + 4)^2}. \tag{15}$$

Demostración: Se debe notar primero que

$$\delta(P) \leq \frac{2mn\delta(A) + 4m + 4n + 1 + 2(m + n + 1)}{2(m + n + 1)(m + n + 2)} = \frac{2mn\delta(A) + 6m + 6nA + 3}{2(m + n + 1)(m + n + 2)},$$

entonces dado que $\tilde{P} = \begin{bmatrix} I & P^T \\ P & 0 \end{bmatrix}$, se tiene que:

$$\delta(\tilde{P}) \leq \frac{2[2mn\delta(A) + 6m + 6n + 3] + 2(m + n + 1)}{(3m + 3n + 4)^2},$$

y esto es justamente lo que se quería demostrar. □

De nuevo se puede analizar numéricamente el comportamiento de $\delta(\tilde{P})$ conforme aumenta el tamaño de la matriz A. Se notará lo conveniente que resulta el uso de las estructuras **(EL)** y **(EU)** para manejar esta matriz. Obsérvese, además que si $m = n$, entonces

$$\lim_{n \to +\infty} \delta(\tilde{P}) \leq 4/36 = 1/9 \approx 0{,}111,$$

y de nuevo, el estimado $4mn < (m+n+1)^2$, permite concluir que $\delta(\tilde{P})$ es aproximadamente $1/9$ para m y n "grandes".

La principal desventaja de este enfoque es el incremento en el uso de memoria principal al reemplazar P por $\tilde{P}$. Pareciera estar en contraposición con nuestra lucha a "ultranza" por el ahorro de este recurso. Sin embargo, dado que $\tilde{P}$ es rala, éste es un precio razonable que se puede pagar, y que redunda en una mejora de la velocidad del algoritmo final.

Nótese por ejemplo, que si la matriz tecnológica del problema original es de tamaño 50×50, la matriz $\tilde{P}$ correspondiente es de tamaño 304×304, aproximadamente unas 37 veces más grande que la original. Sin embargo, dado que en este caso $\tilde{P}$ tiene densidad aproximada de 0.123, se puede ahorrar un 87.7 % del espacio en memoria principal. De esta forma al utilizar estructura de datos como **(EU)**, la matriz $\tilde{P}$, ocupa solamente unas 4.5 veces más espacio que la matriz original.

La "variante" del algoritmo de Karmarkar, que se propone en esta investigación, será llamada **(K & MR)** (Karmarkar y matrices ralas), y puede ser explicada ahora con más propiedad. Las tres fases del método de Karmarkar (consulte [2]) permanecen idénticas. La modificación estriba en el uso de estructuras de datos sensibles a las características de las matrices ralas presentes a lo largo del proceso. En efecto, el punto álgido de esta concepción se sitúa en la solución del sistema (10), utilizando un enfoque (como el adoptado) que pretende un ahorro considerable de espacio en memoria principal, sin que ésto vaya en detrimento significativo de la rapidez del algoritmo final.

(K & MR) es una implementación eficiente del algoritmo de Karmarkar, en el sentido de que permite manejar problemas *más grandes* que los tolerados por una implantación rasa (usando arreglos bidimensionales) de este algoritmo. La consigna que ha marcado la directriz en el desarrollo de esta investigación, ha sido construir una versión del algoritmo Karmarkar aplicable a programa lineales "grandes", administrando eficientemente la memoria principal del computador. Esto no quiere decir que no se haya cuidado la eficiencia de los procedimientos involucrados, no obstante, cuando se debió escoger entre velocidad y memoria RAM, casi siempre se sacrificó velocidad. Un aspecto en que se evidencia perfectamente este lineamiento, es en el manejo de la matriz tecnológica original. En efecto, dicha matriz se carga directamente de disco duro, cada vez que sea necesaria y, como se explica seguidamente, esto se da en cada iteración requerida en las fases II y III. Se debe reconocer que día con día las limitaciones de memoria RAM van disminuyendo y esto puede ser una crítica al enfoque adoptado aquí, sin embargo las ideas plasmadas en este trabajo permitirán utilizar con más eficiencia cualquier cota de memoria RAM que se proponga.

La matriz $\tilde{P}$, se construye usando como piedra angular, la matriz tecnológica original A. Dado que se debe resolver el sistema (10), la invocación del procedimiento que obtiene la factorización L–U de $\tilde{P}$, destruye esta matriz, por lo que se hace necesario construirla después de cada invocación de esta rutina. Al analizar la matriz $\tilde{P}$, se observa que A, está "presente" 4 veces en ésta, no obstante basta con cargarla una sola vez del disco duro. Este razonamiento no es tan obvio como suena, cuando se usa la estructura de dados **(EU)**. El procedimiento para cargar $\tilde{P}$ utilizando **(EU)** puede ser el siguiente:

1. Colóquese la matriz $A_{m\times n}$ entre las filas $2m+2n+3$ y $3m+2n+2$ de $\tilde{P}$, empezando en al columna 1 y terminado (obviamente) en la columna n. Llamaremos a este bloque (principal), (BP).

   Es importante dejar 3 campos disponibles después de colocar cada fila de A en $\tilde{P}$, dado que la filas, comprendidas entre la $2m + 2n + 3$ y la $3m + 2n + 2$ no han sido construidas del todo. En efecto, existen tres entradas (por cada una de estas filas) que pueden ser no cero: la que determina I_m (que obviamente es no nula), la correspondiente en $\alpha_{m\times 1}$ y la determinada por $-b_{m\times 1}$.

2. Calcúlesen los vectores α y β utilizando la matriz $\tilde{P}$ construida hasta ahora. Dado que A está contenida en $\tilde{P}$, este cálculo requiere simplemente acceder las entradas apropiadas de $\tilde{P}$.

3. Complétense las filas comprendidas entre las $2m + 2n + 3$ y $3m + 2n + 2$, utilizando los vectores α y b.

4. Mediante el uso del bloque (BP) de $\tilde{P}$, constrúyase el resto de la matriz.

5. Para terminar de formar $\tilde{P}$, se debe multiplicar la i-ésima columna del bloque formado por las filas comprendidas entre la $2m + 2n + 3$ y $3m + 2n + 2$ y las columnas ubicadas entre la 1 y la $2(m + n + 1)$, por la i-ésima entrada de x_k (la solución de turno).

Es importante notar que a la hora de resolver el sistema en donde se involucra $\tilde{P}$, se puede empezar la eliminación de Gauss a partir de la fila $2m + 2n + 3$, dado que las filas precedentes no necesitan ser procesadas.

En la fase III del método de Karmarkar, también es necesario resolver sistemas de ecuaciones para purificar la solución obtenida en la fase II. En este caso, sin embargo, la matriz (que llamaremos R) de cada sistema, es más sencilla que $\tilde{P}$, y de hecho también más pequeña. Dado que R también utiliza la matriz tecnológica original A, es necesario cargarla directamente desde el disco duro, esto se puede hacer colocándola de una vez en las primeras m filas y las primeras n columnas de R. Dado que R no es rectangular, se le puede completar hasta obtener una matriz cuadrada, utilizando renglones adicionales generados aleatoriamente, pero cuidando que tengan "suficientes" entradas nulas, de suerte que la estructura **(EU)** siga siendo apropiada.

(K & MR) tiene una virtud en la que se ha hecho poco énfasis, a saber la obtención concomitante de la solución para los problemas dual y primario. En efecto, la fase I del método de Karmarkar que busca transformar un problema dado en la forma canónica a la forma estándar de Karmarkar, indirectamente permite la obtención de la solución del problema dual. Si tanto el problema primal como el dual son óptimos, se sabe que ambos objetivos óptimos son iguales (esto es en virtud de las condiciones de optimalidad de Kuhn–Tucker, consúltese [4]). Para obtener entonces la solución óptima del problema dual, bastará con utilizar las posiciones ubicadas entre la $m + n + 1$ y la $2m + n$ de la solución óptima para el problema transformado (presentado en el segundo artículo de esta serie). En [1] el lector encontrará los detalles de **(K & MR)**., además ahí se efectuan experimentos tendientes a valorar las posibilidades de **(K & MR)**, comparándolo con el método del símplex.

Otro aspecto que se debe mencionar es la utilización del procesamiento en paralelo para acelerar la obtención de soluciones. Sería agradable que algún lector pueda estar interesado en continuar con esta línea de investigación.

Referencias

[1] J. F. Avila Herrera (1993) *Una implementación eficiente del algoritmo de Karmarkar*. Tesis de Maestría ITCR, Cartago, C.R..

[2] J. F. Avila Herrera (1995) "Método de Karmarkar", *Revista de Matemáticas* 1

[3] M. S. Bazaraa y J. J. Jarvis (1984) *Programación Lineal y Flujo de Redes, 1era Edición*. Limusa, México.

[4] M. S. Bazaraa, J. J. Jarvis and H. Sherali (1990) *Linear Programming and Network Flows, 2nd Edition*. John Wiley & Sons, N.Y..

[5] R. L. Burden y J. D. Douglas (1985) *Análisis Numérico*. Grupo Editorial Iberoamérica, México.

[6] I. E. Duff and J. K. Reid (1989) *Direct Methods for Sparse Matrices*. Oxford Science Publications, Oxford.

[7] P. E. Gill, W. Murray, M. A. Saunders and M. H. Wright (1986) *Mantaining LU Factors of a General Sparse Matrix*. Department of Operations Research, Stanford University, California.

[8] G. H. Golub and C. F. Van Loan (1989) *Matrix Computations, 2nd Edition*. The Johns Hopkins University Press, USA.

[9] N. Karmarkar (1984) "A new polynomial-time algorithm for Linear Programming", *Combinatorica* 4

[10] M. Nuñez y V. Kong (1992) *El problema de la Mezcla de Alimentos*. Reporte Técnico, ITCR.