Trabájo Práctico 3

PageRank

17 de noviembre de 2013 Métodos Numéricos

Integrante	LU	Correo electrónico
Escalante, José	822/06	joe.escalante@gmail.com
Osinski, Andrés	405/07	andres.osinski@gmail.com
Raskovsky, Iván Alejandro	57/07	iraskovsky@dc.uba.ar

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

Índice

1. Abstract **2**

2. Introducción Teórica **3**

2.1. Page Rank 3

2.2. Cálculo alternativo de $x^{(k+1)} = P_2 x^{(k)}$ 3

2.3. Lema: $P^t\vec{x}$ preserva la Norma 1 de $\vec{x}$ 3

3. Desarrollo **5**

3.1. Detalles de Implementación 5

3.1.1. Enfoque Inicial - Etapa Python 5

3.1.2. Implementación - Etapa C++ 5

4. Resultados **6**

5. Discusión y Conclusiones **8**

A. Referencias **9**

1. Abstract

En este trabajo nos concentraremos en analizar la teoría detrás de cómo hace Google para indexar los sitios web.

Dado a conocer a través de un paper en 1998, el algoritmo Page Rank, se convirtió en una de las claves del suceso del motor de búsqueda Google. Su implementación se basa en la creación de un ranking en el cual se pondera con un cierto criterio cada una de las páginas.

En el armado del ranking recaen conceptos de Algebra Lineal, los cuales veremos en detalle a lo largo de este informe. En particular veremos un método iterativo para calcular el ranking y una variación (extraída de un paper), la cual empíricamente optimiza la cantidad de iteraciones.

Al final presentaremos resultados de experimentaciones que nos parecieron pertinentes, mostrando que efectivamente la variación del método iterativo sí optimiza la cantidad de iteraciones.

Palabras clave:

- Page Rank
- Metodo de Potencia
- Extrapolación Cuadrática
- QR

2. Introducción Teórica

2.1. Page Rank

La manera en cómo se modela el problema del ranking es usando una matriz de adyacencias. Sea $W \in \mathbb{R}^{n \times n}$ donde n es la cantidad de sitios indexados, luego el elemento w_{ij} es igual 1 si existe un link de la página i a la página j y 0 en caso contrario. A su vez los links autoreferenciados se ignoran por lo que en diagonal tenemos todos valores nulos.

Ahora con W podemos extraer la cantidad de links salientes de cada página, simplemente sumando los elementos de la fila correspondiente, llamemos n_j al grado de la página j donde $n_j = \sum_{i=1}^{n} w_{ij}$

2.2. QR y Reflecciones de Householder

Existen variadas formas de descomponer una matriz, en este trabajo en particular usaremos QR, el cual consiste en descomponer una matriz A en una matriz ortogonal Q y una triangular superior R de manera que $A = QR$. Al tener descompuesta A en esa forma y teniendo un sistema $Ax = b$, la obtención del vector solución se realiza resolviendo el sistema $Rx = Q^t b$.

A su vez existen distintos procedimientos para poder hallar la descomposición QR de una matriz, una de ellas son las Reflecciones de Householder, las cuales consisten en ir poniendo ceros debajo de la diagonal en cada caso del procedimiento. Este procedimiento sólo se puede llevar a cabo si es que la matriz tiene mas filas que columnaso a la sumo igual cantidad.

Yendo en detalle: Sea $A \in \mathbb{R}^{m \times n}$ una matriz con $m \geq n$

2.3. Cálculo alternativo de $x^{(k+1)} = P_2 x^{(k)}$

Veamos primero cómo utilizando el algoritmo de Kamvar podemos optimizar el espacio requerido en memoria para el almacenamiento de la matriz P_2 y el tiempo de ejecución requerido para hacer la multiplicación entre matrices y vectores.

Queremos ver que el algoritmo propuesto por [**?**, Algoritmo 1] es equivalente a la operación $\vec{y} = A\vec{x}$, para $A = (cP' + (1-c)E)^t$, donde P' es la matriz estocástica por filas de transiciones de links ajustada para considerar saltos aleatorios en páginas sin outlinks, y E es la matriz uniforme de teletransportación con valor $\frac{1}{n}$ en cada celda.

Para ello, expandimos las ecuaciones de ambos y veremos que las mismas producen el mismo cálculo.

Primero, la matrix P^t se desarrolla como

$$(cP + (1-c)E)^t \vec{x}$$

Y la matrix de [**?**, Algoritmo 1] como

$$cP^t\vec{x} + (\|\vec{x}\|_1 - \|\vec{y}\|_2)\vec{v}$$

donde $\vec{y}$ es el vector resultante de $cP^t\vec{x}$ y $\vec{v}$ es el vector de probabilidad uniforme de valor $\frac{1}{n}$ en cada elemento. Luego planteamos la equivalencia

$$\begin{aligned} (cP + (1-c)E)^t \vec{x} v &= cP^t\vec{x} + (\|\vec{x}\|_1 - \|\vec{y}\|_2)\vec{v} \\ cP^t\vec{x} + (1-c)E^t\vec{x} &= cP^t\vec{x} + (\|\vec{x}\|_1 - \|\vec{y}\|_2)\vec{v} \\ (1-c)E^t\vec{x} &= (\|\vec{x}\|_1 - \|\vec{y}\|_2)\vec{v} \end{aligned}$$

2.4. Lema: $P^t\vec{x}$ preserva la Norma 1 de $\vec{x}$

Sea P^t una matriz estocástica por columnas, luego los elementos de cada columna suman 1.

Luego P^t describe una transformación lineal de $\vec{x}$ donde la suma de los valores de cada x_i se reparte en los y_i resultantes (por ser cada y_i una combinación lineal de los x_i.) Como cada columna de P suma 1, y cada elemento de x se termina multiplicando por los elementos de una columna, y además los valores de P y x son positivos, entonces la ecuación

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

es equivalente a

$$\sum_{i=1}^{n} |y_i|$$

Luego P^t preserva norma 1.

Volviendo al problema, si observamos que la norma 1 de y es

$$\begin{aligned} \|\vec{y}\|_1 &= \left\|cP^t\vec{x}\right\|_1 \\ &= c\,\|\vec{x}\|_1 \end{aligned}$$

entonces podemos ver que

$$\begin{aligned} \|\vec{x}\|_1 - \|\vec{y}\|_1 &= \|\vec{x}\|_1 - c\,\|\vec{x}\|_1 \\ &= (1-c)\,\|\vec{x}\|_1 \\ &= (1-c)\,\|\vec{x}\|_1 \end{aligned}$$

por ende

$$(\|\vec{x}\|_1 - \|\vec{y}\|_1)\vec{v} = (1-c)\,\|\vec{x}\|_1\,\vec{v}$$

entonces el método de algortimo 1 tiene la forma

$$cP^t\vec{x} + (1-c)\,\|\vec{x}\|_1\,\vec{v}$$

Si observamos la segunda mitad de la definición de P^t, es decir, $(1-c)E^t$, veremos que el producto a la izquierda por $\vec{x}$ resulta en una matrix con la forma

$$E^t\vec{x} = \begin{bmatrix} \frac{1-c}{n}\sum_{i=1}^{n}|x_i| \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \frac{1-c}{n}\sum_{i=1}^{n}|x_i| \end{bmatrix} = \begin{bmatrix} \frac{1-c}{n}\,\|\vec{x}\|_1 \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \frac{1-c}{n}\,\|\vec{x}\|_1 \end{bmatrix} = (1-c)\frac{1}{n}\,\|\vec{x}\|_1$$

Con ello concluimos que los dos términos del algoritmo de Kamvar son equivalentes a la matriz A de transiciones. ■

3. Desarrollo

3.1. Detalles de Implementación

3.1.1. Enfoque Inicial - Etapa Python

En principio para evitarnos detalles de manejo de memoria y contar con una mayor de expresividad de lenguaje, implementamos el trabajo en Python.

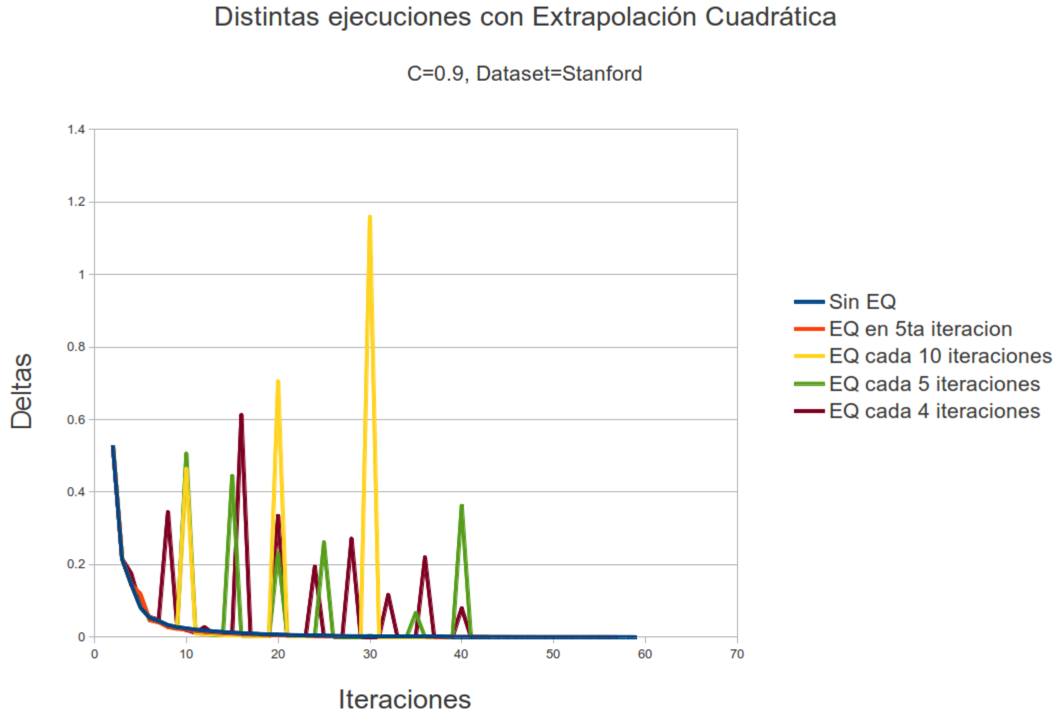
Usando librerías como Numpy y Scipy, pudimos hacer uso de matrices esparsas y operarlas cómodamente de manera eficiente. Tan sólo unas horas de trabajo y terminamos una implementación que devolvía resultados que parecían correctos. Inclusive con datasets enormes, como los que se pueden encontrar en la página de Stanford, el programa en Python tardaba pocos segundos por iteración y en cuestión de minutos armaba el ranking.

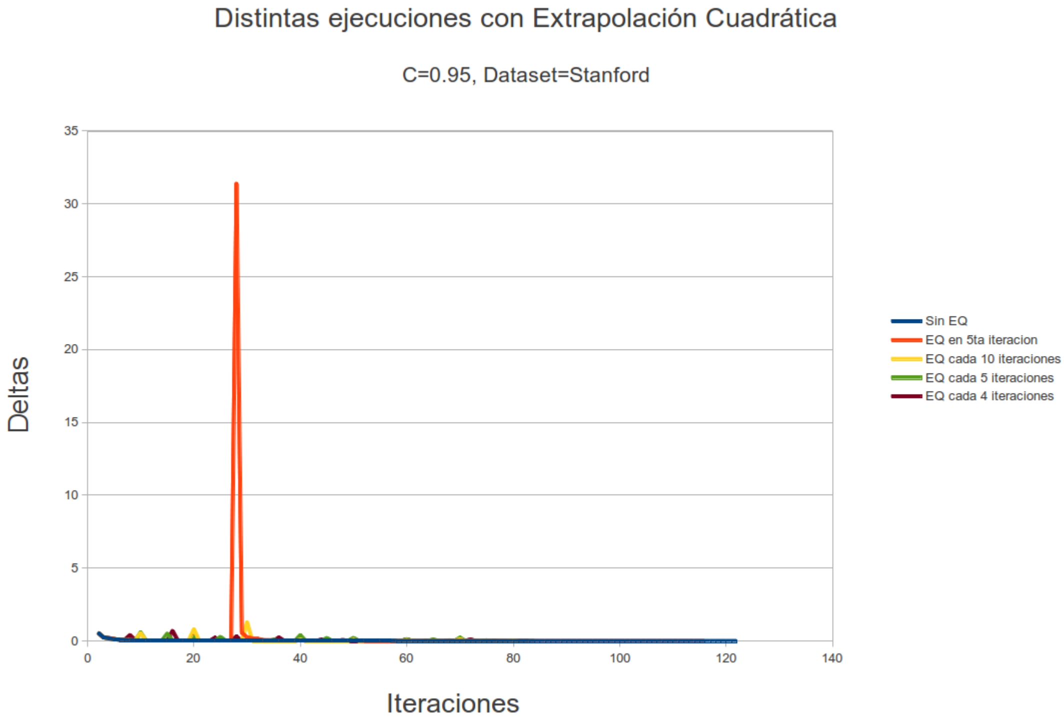
3.1.2. Implementación - Etapa C++

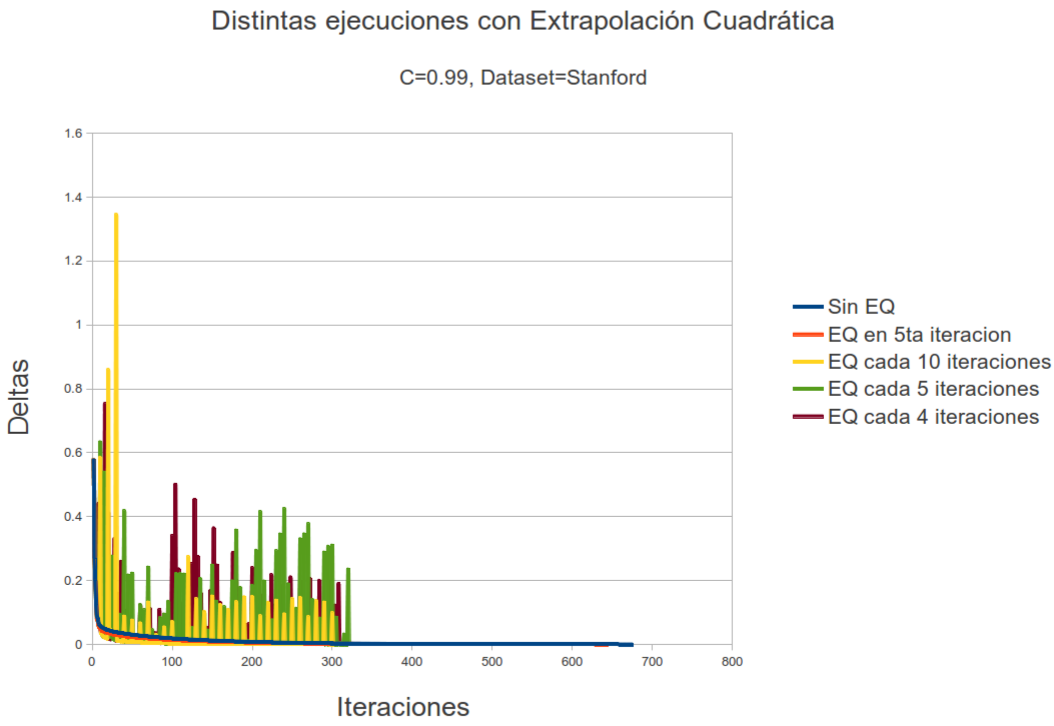
Una vez habiendo comprobado que la idea de cómo implementar este trabajo funcionaba, es que usamos el código de Python a manera de pseudocódigo para el de C++.

En C++ para armar las matrices esparsas usamos STL y la función *map*. A diferencia de la implementación en Python donde hacemos uso indiscriminado de la riqueza de las librerías, nos vimos forzados a acomodar las operaciones de manera tal que tengamos que implementar solamente las operaciones exclusivamente necesarias.

4. Resultados






Distintas ejecuciones con Extrapolación Cuadrática
C=0.99, Dataset=Stanford
Deltas
Iteraciones
Sin EQ
EQ en 5ta iteracion
EQ cada 10 iteraciones
EQ cada 5 iteraciones
EQ cada 4 iteraciones
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
100
200
300
400
500
600
700
800

5. Discusión y Conclusiones

A. Referencias

Wikipedia Burden