DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II

Métodos Numéricos
Segundo Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Iván Arcuschin	678/13	iarcuschin@gmail.com
Martín Jedwabny	885/13	martiniedva@gmail.com
José Massigoge	954/12	jmmassigoge@gmail.com
Iván Pondal	078/14	ivan.pondal@gmail.com

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. Introducción

El objetivo de este Trabajo Práctico es implementar diferentes algoritmos de resolución de sistemas de ecuaciones lineales y experimentar con dichas implementaciones en el contexto de un problema de la vida real.

El problema a resolver es hallar la isoterma $500C^o$ en la pared de un Alto Horno. Para tal fin, deberemos particionar la pared del horno en puntos finitos, y luego resolver un sistema de ecuaciones lineales, en el cual cada punto de la pared interior y exterior del Horno es un dato, y las ecuaciones para los puntos internos satisfacen la ecuación del calor.

Los experimentos realizados se dividen en dos partes: Comportamiento del sistema y Evaluación de los métodos. En la primera parte, analizaremos con distintas instancias de prueba y se estudiará la proximidad de la isoterma buscada respecto de la pared exterior del horno. En la segunda parte, analizaremos el tiempo de computo requerido para la resolución del sistema en función de la granularidad de la discretización y analizaremos el escenario en el cual las temperaturas de los bordes varían a lo largo del tiempo.

2. Modelo

2.1. Rankings de Páginas Web

Hoy en día la cantidad de páginas web en todo el mundo asciende a una cantidad de 4.79 billones (sólo las indexadas). Es por esta razón que los buscadores (Search Engines) cumplen un rol tan importante en el uso diario de internet desde hace muchos años.

Dichos buscadores nos permiten realizar busquedas de páginas web mediante distintos criterios, facilitando el acceso a la información.

Un posible criterio (bastante utilizado) es considerar que las páginas web populares son las más buscadas. De esta forma, el buscador puede ofrecernos en orden descendiente de popularidad los resultados obtenidos, esperando que encontremos más rápido lo que buscamos.

Siguiendo esta intuición, se han elaborado diferentes algoritmos para "rankear" las páginas web. A continuación presentaremos el famoso método PageRank, utilizado por Google en sus comienzos.

2.1.1. PageRank

El algoritmo de PageRank[1] se define para un conjunto de páginas Web $= \{1, \ldots, n\}$ de forma tal de asignar a cada una de ellas un puntaje que determine la importancia relativa de la página respecto de las demás.

Llamemos x_j al puntaje asignado a la página $j \in Web$, que es lo que buscamos calcular.

Ahora, un link saliente de la página j a la página i puede significar que i es una página importante. Pero bien podría ser que j sea una página muy poco importante, por lo que deberíamos ponderar sus links salientes para decidir la importancia de las páginas a las que apunta.

Luego, vamos a considerar que la importancia de la página i obtenida mediante el link de j es proporcional a la importancia de j e inversamente proporcional al grado de j. Entonces, si $L_k \in Web$ es el conjunto de páginas web que apuntan a la página k:

$$x_k = \sum_{j \in L_k} \frac{x_j}{n_j}, \quad k = 1, \ldots, n \tag{1}$$

En este algoritmo, hallaramos los x_k modelando el problema como una cadena de Markov, a la cual llamaremos Matriz de Transición, y que construiremos de la siguiente forma:

1. Sea G el grafo de la Web, dónde cada vértice es una página web y un eje de v a u significa que la página v tiene link saliente hacia u.

2. Luego, sea $W \in \{0,1\}^{n \times n}$ la *matriz de conectividad* de G, tal que la celda $\{i, j\}$ tiene un 1 si hay un link saliente de la j-ésima página a la i-ésima página (los *autolinks* son ignorados, o lo que es lo mismo, $\forall\ 1 \leq i \leq n\ W_{i,i}$).

3. Si definimos $n_j = \sum_{i=1}^{n} W_{i,j}$ como el grado de j (la cantida de links salientes), entonces podemos definir la matriz $P \in \mathbb{R}^{n \times n}$, tal que la $P_{i,j} = 1/n_j * W_{i,j}$, y P es estocástica por columnas.

   Además, notese que resolver el sistema dado por 1 es equivalente a encontrar un $x \in \mathbb{R}^n$ tal que $Px = x$. Es decir, encontrar el autovector asociado al autovalor 1 de P tal que $x_i > 0$ y $\sum_{i=1}^{n} x_i = 1$.

4. Ahora, puede pasar que para algún j, $n_j = 0$ lo que indicaría que la página j no tiene ningún link saliente. Para remediar estos casos, vamos a modificar P utilizando la idea del *navegante aleatorio*, de forma tal que para un j sin links salientes, la probabilidad de que el navegante salte a cualquier otra página i es $1/n$.

   Entonces, $P_1 = P + D$, dónde $D = vd^t$, $d \in \{0,1\}^n$ tal que $d_j = 1$ si $n_j = 0$, y $d_j = 0$ en caso contrario, y $v \in \mathbb{R}^n$ tal que $v_j = 1/n$.

[1] Kurt Bryan and Tanya Leise. The linear algebra behind google. SIAM Review, 48(3):569–581, 2006.

5. Entonces, P_1 es estocástica por columnas, pero puede que no sea regular. Para que sí lo sea, extendemos el concepto anterior a todas las páginas (fenómeno de *teletransportación*).

   Luego, $P_2 = c * P_1 + (1 - c) * E$, donde $\forall\ 1 \leq i, j \leq n,\ E_{i,j} = 1/n$, y $c \in (0, 1)$. Llamamos a c coeficiente de teletransportación.

   Lo que nos queda es un matriz P_2 estocástica por columnas y $\forall\ 1 \leq i, j \leq n,\ (P_2)_{i,j} > 0$.

Una vez que tenemos la Matriz de Transición, generaremos el puntaje para cada página buscando el autovector w del autovalor 1 de P_2, tal que $P_2 w = w$ y w sea un vector de probabilidades (normalizado con norma 1).

Es decir, generar el ranking de páginas equivale a aplicar el método de la potencia a la matriz P_2 y una vez hallado el w mencionado, ordenar los puntajes de mayor a menor:

$$ranking = \{p_1, \ldots, p_n\}, \text{donde } \forall\ i = 1, \ldots, n-1,\ w_{p_i} \geq w_{p_{i+1}} \tag{2}$$

2.1.2. PageRank con matriz Esparsa

Partiendo del hecho de que, aproximadamente, cada pagina Web tiene 7 links salientes[2] surge la posibilidad de optimizar la estructura que representa el Grafo dirigido de la Web, debido a la baja densidad de la matriz de conectividad, W, asociada a dicho Grafo. Esta optimizacion no es solo una posibilidad, sino una necesidad dado que la cantidad de Webs indexadas tiene un requerimiento prohibitivo en terminos de memoria si la representamos, ingenuamente, como un vector de vectores. Nuestra hipotesis es que, utilizando una estructura de datos optimizada, no solo obtendremos mejoras en terminos de complejidad espacial, sino tambien en terminos de complejidad temporal.

A partir de este analisis, nos propusimo analizar tres estructuras de datos que representan matrices esparsas:

1. *Dictionary of Keys (dok)*
2. *Compressed Sparse Row (CSR)*
3. *Compressed Sparse Column (CSC)*

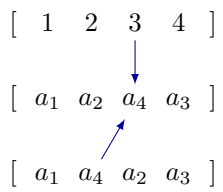
Sea $A \in \mathbb{R}^{n \times n}$ una matriz esparsa, y sea m es la cantidad de valores distintos de 0 de A.

1. Dictionary of Keys (dok): Esta estructura esta representada como un vector de diccionarios, en donde cada posicion, i, del vector puede considerarse como una fila, y la key del diccionario contenido dentro de esa posicion representa una columna, j, siendo el valor asociado a esa key, en el diccionario, el valor en la matriz contenido en esa fila y columna (A_{ij}). Solo aquellas A_{ij} que tienen valores distintos de 0 tienen una key en los diccionarios.

Veamos un ejemplo. Sea $B \in \mathbb{R}^{4 \times 4}$ la siguiente matriz:

$$B = \begin{pmatrix} 10 & 0 & 0 & -2 \\ 3 & 9 & 0 & 0 \\ 0 & 7 & 8 & 0 \\ 3 & 0 & 4 & 5 \end{pmatrix}$$

El dictionary of keys que representa a B es:



[2] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Extrapolation methods for accelerating pagerank computations. In Proceedings of the 12th international conference on World Wide Web, WWW '03, pages 261–270, New York, NY, USA, 2003. ACM.

[1;10]	[1;3]	[2;9]	11	8	5

Esta claro que intercambiando columnas por filas, como posiciones del vector contenedor, y filas por columnas como keys, representamos la misma matriz.

Las ventajas de usar esta estructura de datos son las siguientes:

1. Facilidad para crearse de forma dinamicamente, es decir, si a priori no conocemos la forma de la matriz a representar.
2. La operacion matriz esparsa por vector es sencilla.

Por otro lado, sus desventajas son:

1. Los accesos a los valores no tienen complejidad temporal constante.
2. Las operaciones entre matrices esparsas, sea la suma o la multiplicacion, son engorrosas.

2. Compressed Sparse Row (CSR): Esta estructura utiliza tres vectores, dos de tamano m, llamemosles *val* y *col_ind* respectivamente, y uno de tamano $n+1$, llamemosle *row_ptr*.

- En *val* guardamos los valores de A distintos de 0, recorriendo A por filas, es decir fijando las filas y avanzado por las columnas.
- En la posicion k de *col_ind* guardamos el indice j del valor A_{ij} contenido en la posicion k del vector *val*.
- Por ultimo en *row_ptr* guardamos indices del vector *val* en los cuales se encuentran el primera valor de cada fila, de tal forma que para la fila i, el valor k contenido en *row_ptr[i]*, nos dice que *val[k]* corresponde al primer valor de la fila i y el valor k' contenido en *row_ptr[i+1]*, nos dice que *val[k'-1]* corresponde al ultimo valor de la fila i. Vale la pena aclarar que si $k = k'$ entonces la fila no tiene valores distintos de 0.

Utilizemos B como la matriz del punto anterior:

$$B = \begin{pmatrix} 10 & 0 & 0 & -2 \\ 3 & 9 & 0 & 0 \\ 0 & 7 & 8 & 0 \\ 3 & 0 & 4 & 5 \end{pmatrix}$$

La misma seria representado de la siguiente manera:

- *val:*

10	-2	3	9	7	8	3	4	5

- *col_ind:*

1	5	1	2	2	3	1	3	5

- *row_ptr:*

1	3	5	7	10

Las ventajas de usar esta estructura de datos son las siguientes:

1. Eficiente a la hora de realizar operaciones aritmeticas entre matrices esparsas (suma, multiplicacion).
2. La operacion matriz esparsa por vector es sencilla.

Por otro lado, sus desventajas son:

1. Dificultad a la hora de crearse de forma dinamica, ya que debemos recalcular/redimensionar los tamanos de los vectores.
2. Cambios en la esparcidad son costosos, misma razon que el punto anterior.

3. Compressed Sparse Column (CSC): Muy parecidad a la *Compressed Sparse Row (CSR)*. Tambien utiliza tres vectores, dos de tamano m, llamemosles *val* y *row_ind* respectivamente, y uno de tamano $n+1$, llamemosle *col_ptr*.

- En *val* guardamos los valores de A distintos de 0, recorriendo A por columnas, es decir fijando las columnas y avanzado por las filas.
- En la posicion k de *row_ind* guardamos el indice i del valor A_{ij} contenido en la posicion k del vector *val*.
- Por ultimo en *col_ptr* guardamos indices del vector *val* en los cuales se encuentran el primera valor de cada columna, de tal forma que para la columna j, el valor k contenido en *col_ptr[j]*, nos dice que *val[k]* corresponde al primer valor de la fila j y el valor k' contenido en *col_ptr[j+1]*, nos dice que *val[k'-1]* corresponde al ultimo valor de la col j. Vale la pena aclarar que si $k = k'$ entonces la columna no tiene valores distintos de 0.

Nuevamente utilizemos B como ejemplo:

$$B = \begin{pmatrix} 10 & 0 & 0 & -2 \\ 3 & 9 & 0 & 0 \\ 0 & 7 & 8 & 0 \\ 3 & 0 & 4 & 5 \end{pmatrix}$$

La misma seria representado de la siguiente manera:

- *val:*

10	3	3	9	7	8	4	-2	5

- *row_ind:*

1	2	4	2	3	3	4	1	4

- *col_ptr:*

1	4	6	8	10

Las ventajas de usar esta estructura de datos son las siguientes:

1. Eficiente a la hora de realizar operaciones aritmeticas entre matrices esparsas (suma, multiplicacion).
2. La operacion matriz esparsa por vector es sencilla.

Por otro lado, sus desventajas son:

1. Dificultad a la hora de crearse de forma dinamica, ya que debemos recalcular/redimensionar los tamanos de los vectores.
2. Cambios en la esparcidad son costosos, misma razon que el punto anterior.

Contexto de Uso Antes de definir que estructura de datos vamos a utilizar para representar matrices esparsas, debemos definir el contexto de uso de la misma. Sabemos que el algoritmo de PageRank utiliza el metodo de la Potencia, usando la matriz P_2 definida previamente. Sin embargo la P_2 no es esparsa, inclusive todos sus valores son distintos de 0, por lo cual utilizar la implementacion del Page Rank descripta en la Seccion 2.1.1 no es posible .

Para poder hacer frente a esta dificultad, utilizamos el algoritmo propuesto por Kamvar et al.[Algoritmo 1][3] para calcular $x^{(k+1)} = P_2 x^k$, cuyos pasos son los siguientes:

1. $y = cPx$
2. $w = ||x||_1 - ||y||_1$
3. $y = y + wv$

donde c, P y v corresponden al escalar, la matriz y el vector descriptos en la Seccion 2.1.1.
Tenemos que ver que el algortimo descripto calcula efectivamente $x^{(k+1)} = P_2 x^k$:

Proposición 1. *El Algoritmo 1 calcula* $x^{(k+1)} = P_2 x^{(k)}$

Demostración. Por definicion sabemos que

$$\begin{aligned} x^{(k+1)} &= P_2 x^{(k)} \\ &= (cP_1 + (1-c)E)x^{(k)} \\ &= (cP + cD + (1-c)E)x^{(k)} \\ &= (cP + cD + (1-c)E)x^{(k)} \\ &= (cP + cvd^t + (1-c)v1^t)x^{(k)} \\ &= cPx^{(k)} + cvd^t x^{(k)} + (1-c)v1^t x^{(k)} \end{aligned}$$

Por otro lado, el Algoritmo 1 nos dice que:

$$x^{(k+1)} = cPx^{(k)} + (||x^{(k)}||_1 - ||cPx^{(k)}||_1)v$$

Luego basta ver que:

$$\begin{aligned} cPx^{(k)} + (||x^{(k)}||_1 + ||cPx^{(k)}||_1)v &= cPx^{(k)} + cvd^t x^{(k)} + (1-c)v1^t x^{(k)} \\ (||x^{(k)}||_1 - ||cPx^{(k)}||_1)v &= cvd^t x^{(k)} + (1-c)v1^t x^{(k)} \\ ||x^{(k)}||_1 v - ||cPx^{(k)}||_1 v &= cvd^t x^{(k)} + v1^t x^{(k)} - cv1^t x^{(k)} \end{aligned}$$

Sabemos que $\forall\ 1 \leq i \leq n\ x_i^{(k)} \geq 0$, ya que x es un vector de probabilidades. Entonces $||x^{(k)}||_1 = \sum_{i=1}^{n} x_i^{(k)}$.
Tambien sabemos que $1^t x^{(k)} = \sum_{i=1}^{n} 1x_i^{(k)}$, por lo tanto $||x^{(k)}||_1 = 1^t x^{(k)}$.
Por otro lado sabemos que $0 \leq c \leq 1$, lo que implica que $|c| = c$.
Teniendo en cuenta esta informacion, la igualdad nos queda de la siguiente forma:

$$\begin{aligned} v1^t x^{(k)} - c||Px^{(k)}||_1 v &= cvd^t x^{(k)} + v1^t x^{(k)} - cv1^t x^{(k)} \\ -c||Px^{(k)}||_1 v &= cvd^t x^{(k)} - cv1^t x^{(k)} \end{aligned}$$

Tomando como factor comun v y c:

$$\begin{aligned} -c||Px^{(k)}||_1 v &= cv(d^t x^{(k)} - 1^t x^{(k)}) \\ -||Px^{(k)}||_1 &= d^t x^{(k)} - 1^t x^{(k)} \\ -||Px^{(k)}||_1 &= (d^t - 1^t)x^{(k)} \end{aligned}$$

[3] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Extrapolation methods for accelerating pagerank computations. In Proceedings of the 12th international conference on World Wide Web, WWW '03, pages 261–270, New York, NY, USA, 2003. ACM.

Sabemos, por definicion de d, que d^t solo tiene 1 y 0 como valores, por lo tanto al realizar $(d^t - 1^t)$ obtendremos un vector fila, llamemosle z, cuyos valores seran 0 o -1. Lo que implica que $z_j = 0$ cuando $d_j^t = 1$, situacion que se da cuando, para ese j, $\sum_{i=1}^{n} P_{ij} = 0$. Caso contrario $z_j = -1$ cuando $d_j^t = 0$, situacion que se da cuando, para ese j, $\sum_{i=1}^{n} P_{ij} = 1$. Por lo tanto al mutiplicar $zx^{(k)}$ obtendremos un escalar que sera la suma de los opuestos de los $x_j^{(k)}$ que cumplan que, para ese j, $\sum_{i=1}^{n} P_{ij} = 1$, ya que los otros seran mutiplicados por 0.

Por otro lado, partiendo del hecho de que P es estocastica por columnas, sabemos que al realizar $Px^{(k)}$, aquellas columnas, j, que cumplan que $\sum_{i=1}^{n} P_{ij} = 0$, forzaran a que los $x_j^{(k)}$ no sean parte de ninguna componente del vector resultante, ya que en todas las multiplicaciones de filas de P por el vector $x^{(k)}$, estos $x_j^{(k)}$ seran multiplicados por 0. Caso contrario cuando las columnas, j, cumplan que $\sum_{i=1}^{n} P_{ij} = 1$, estas forzaran a que la suma de todos los $x_j^{(k)}$ presentes en las distintas componentes del vector resultante sea igual a $x_j^{(k)}$. Precisamente al aplicar $||.||_1$ al vector $Px^{(k)}$, el resultado sera un escalar que sera la suma de todas los $x_j^{(k)}$ que cumplan la condicion antes descripta. Por ultimo si al resultado de aplicar $||.||_1$ lo multiplicamos por -1 obtenemos un escalar que sera la suma de los opuestos de los $x_j^{(k)}$ que cumplan que, para ese j, $\sum_{i=1}^{n} P_{ij} = 1$.

Por todo lo expuesto anteriormente, podemos concluir que $-||Px^{(k)}||_1 = (d^t - 1^t)x^{(k)}$, lo que implica que El Algoritmo 1 calcula $x^{(k+1)} = P_2 x^{(k)}$. □

Luego, sabemos que la unica operacion que vamos a realizar con la matriz esparsa es multiplicarla por un vector. Tambien sabemos que la construccion de la matriz es de forma dinamica, ya que vamos leyendo los links salientes de cada pagina de forma secuencial.

A partir de esta informacion concluimos que la estructura de datos que mas se adecua a nuestras necesidades es el *Dictionary of Keys (dok)*, en donde las posiciones del vector representan las filas y las keys de los diccionarios las columnas.

Esta eleccion se debe al hecho de que podemos actualizar la matriz de forma sencilla a medida de que vamos leyendo los datos de entrada, y a su vez la multiplicacion por un vector se realiza de forma sencilla, vease Seccion Implementacion Esparsa (CAMBIAR CON LA REF).

2.2. Rankings en competencias deportivas

Elegir un sistema de puntos que sea justo para todos los participantes en un deporte no es una tarea sencilla. Existen muchos factores que afectan el resultado de una competencia como lo pueden ser el orden en el que deben competir entre si los equipos, generando desbalances respecto las capacidades de cada uno. Es por esto que a continuación presentaremos el modelo GeM [4] que busca modelar los resultados de forma tal que estos factores impacten lo menos posible en el posicionamiento final de la tabla de puntajes.

Con el fin de experimentar con distintos modelos, a lo largo del informe trabajaremos sobre los resultados del Torneo de Primera División 2015[5], donde utilizaremos el sistema de ranking estándar de la AFA como punto de comparación.

2.2.1. Generalized Markov chains Method (GeM)

Definición del método

El método GeM es el resultado de tomar el algoritmo PageRank y mediante pequeñas modificaciones utilizar su potencial para establecer un ranking de equipos. Análogo a PageRank, los equipos pasan a formar parte de un grafo dirigido con pesos, donde cada nodo representa un equipo y los pesos de cada arista reflejan el resultado de los partidos jugados entre los vértices conectados.

Formalmente, el modelado se realiza de la siguiente manera:

[4] Angela Y. Govan, Carl D. Meyer, and Rusell Albright. Generalizing google's pagerank to rank national football league teams. In Proceedings of SAS Global Forum 2008, 2008.

[5] Campeonato de Primera División 2015, *Julio H. Grondona*
`http://www.afa.org.ar/html/9/estadisticas-de-primera-division`

1. Representamos el torneo como un grafo con pesos dirigidos de n nodos, donde n es igual a la cantidad de equipos que participan. Cada equipo tiene su respectivo nodo y las aristas contienen como peso la diferencia positiva entre los nodos conectados.

2. Definimos la matriz de adyacencia $A \in \mathbb{R}^{n \times n}$.

$$A_{ij} = \begin{cases} w_{ij} & \text{si el equipo } i \text{ perdió contra } j \\ 0 & \text{caso contrario} \end{cases}$$

   Donde w_{ij} es la suma total de diferencia positiva de puntaje sobre todos los partidos en los que i perdió contra j.

3. Definimos la matriz $H \in \mathbb{R}^{n \times n}$.

$$H_{ij} = \begin{cases} A_{ij} / \sum_{k=1}^{n} A_{ik} & \text{si hay un link de } i \text{ a } j \\ 0 & \text{caso contrario} \end{cases}$$

4. Definimos la matriz GeM, $G \in \mathbb{R}^{n \times n}$ con $u, v, a, e \in \mathbb{R}^n$ y $c \in \mathbb{R}$.

$$G = c(H + au^t) + (1 - c)ev^t$$

$$\sum_{k=1}^{n} v_k = 1 \qquad \sum_{k=1}^{n} u_k = 1 \qquad \forall_{i=1..n} e_i = 1$$

$$0 \leq c \leq 1 \qquad a_i = \begin{cases} 1 & \text{si la fila } i \text{ de } H \text{ es un vector nulo} \\ 0 & \text{caso contrario} \end{cases}$$

5. Por último tenemos que el ranking de los equipos estará definido por el vector $\pi \in \mathbb{R}^n$ tal que

$$\pi^t = \pi^t G$$

o si tomamos la transpuesta en ambos lados

$$G^t \pi = \pi$$

De esta forma al igual que con PageRank, calculando el autovector π obtenemos nuestro ranking. Este modelo permite cierta flexibidad a partir del u, v y c que tomemos.

El vector de probabilidad u se aplicará en el caso de que un equipo se encuentre invicto, esto es el equivalente a que en PageRank un sitio no tenga ningún link entrante, por lo tanto su tratamiento es el mismo, se le asigna a la fila correspondiente el vector con las probabilidades de saltar a otro nodo. Por lo tanto, el vector u nos permite definir con qué probabilidades un equipo invicto perdería contra el resto de los participantes. En el caso de PageRank, este es un vector de distribución uniforme, donde es igual la posibilidad de saltar a cualquiera de los otros nodos, una posible alternativa sería definirlo como un vector cuyas probabilidades se basen en algún ranking anterior.

El vector de probabilidad v, nos da otro tipo de personalización que es la del *navegante aleatorio*. Esta es la probabilidad de que un equipo cualquiera independientemente de los resultados registrados, pierda contra el resto de los equipos. En PageRank, esto lo veíamos como la posibilidad de que estando navegando el grafo, uno se *teletransportará* a otro nodo independientemente de las conexiones de los mismos. Este vector por defecto también suele tomar el valor de la distribución uniforme.

Por último tenemos nuestro valor c que actua como un factor de amortiguación donde lo que se modifica es cuánto afecta el *navegante aleatorio* al resultado final, donde con $c = 0$, únicamente influye el *navegante aleatorio* y con $c = 1$ se elimina el efecto del mismo.

Modelado del empate

Una particularidad de este sistema que se puede observar en la definición del mismo es que no contempla los partidos donde hubo empate. Un empate equivale a que no exista un perdedor y por ende no se modifica el peso de ningún nodo. Para deportes donde el empate no es algo frecuente esto no sería un

problema, pero si tomamos como ejemplo el fútbol, donde los empates son algo mucho más común, el ignorar estos partidos afecta notablemente el ranking.

Podemos tomar como ejemplo los Torneos Argentinos de Primera División y observar el porcentaje de partidos empatados sobre el total que se jugaron.

Campeonato de Primera División	Partidos	Empates	Empates/Partidos
2015	390	126	0.32
2014	190	45	0.23
2013/14	380	117	0.31

Relación entre partidos jugados y empates totales

Como podemos ver, la proporción de partidos empatados no es menor, con lo cual analizaremos cómo impactaría esto en el método GeM. Como mencionamos en la definición del modelo, los pesos de las aristas en el grafo dependen en parte de la diferencia de puntaje con la que gana un equipo sobre otro.

Al encontrarnos con un empate no existe diferencia alguna, por lo tanto no implica ningún cambio en el sistema. Esto podría defavorecer a los equipos que más empates tuvieran en la temporada, ya que sólo los partidos donde hubieran ganado o perdido afectarían su lugar en la tabla de puntajes. Además podría suceder que un equipo perdiese contra un contrincante que no tuviera mucho peso pero empatara contra un puntero, y esto debería verse reflejado ya que de otra manera, sólo quedaría registrado su partido perdido.

Una posible forma de representar los empates aprovechando la estructura del modelo actual sería cuando se produce uno, reflejar en el grafo el equivalente a que ambos equipos hubieran perdido entre si, asignando como diferencia de puntaje el valor con el que empataron. De esta forma no solo incluiríamos este escenario al cálculo del ranking final, si no que además el puntaje con el que empataron tendría peso.

2.2.2. Puntaje AFA

A modo de comparación con el método GeM, siendo que trabajaremos sobre los resultados del Torneo de Primera División, utilizaremos también el sistema de puntajes que propone la AFA[6]. El mismo es bastante sencillo, donde lo que establece es que todos los equipos comienzan con puntaje **0** y se les suma **1** punto en caso de empate, **3** por victoria y **0** por derrota.

A primera vista, dada que la modalidad de este torneo es de todos contra todos, donde cada equipo juega una vez contra el resto, el resultado debería reflejar de forma justa el desempeño de cada participante. Sin embargo, para este torneo en particular se aplicó una fecha adicional de *clásicos*, donde cada equipo vuelve a jugar contra su respectivo clásico. Este es un punto donde se desequilibran un tanto las cosas ya que un participante podría tener asignado como clásico un puntero, mientras que otro al último en la tabla de posiciones. Para este sistema de puntuación ambos partidos serían tratados al igual quel resto del torneo.

[6] `http://www.afa.org.ar/upload/reglamento/Reglamento_PrimeraDivision_2015.pdf`

3. Implementación

3.1. Scripts Rankings Deportivos

Para los dos modelos siendo estudiados, GeM y el sistema de puntaje de la AFA, decidimos implementarlos com scripts para MATLAB/OCTAVE, dado que lo que nos interesaba era analizar los resultados de los mismos y no analizar su tiempo de cómputo u otro atributo relacionado a su implementación.

3.1.1. GeM

El script posee varios parámetros de entrada que pasamos a describir a continuación:

- **in_filename:** Dirección de archivo con datos de entrada.
- **out_filename:** Dirección de archivo donde escribir el resultado final.
- **team_codes_filename:** Archivo opcional con el número de equipo asociado a su nombre correspondiente.
- **c:** Parámetro de amortiguación descrito en la sección 2.2.1 (Por defecto 0.85).
- **date_limit:** Campo opcional con la cantidad de fechas a tomar en cuenta (Por defecto toma todas las fechas disponibles).

Ahora explicaremos el código desarrollado:

1. Leemos la cantidad de equipos y partidos para después cargar todos los partidos disponibles y generar nuestra matriz A.

```
Cargar numero de partidos y equipos
Crear matriz A de tamanio equipos*equipos llena de 0s
Mientras partidos > 0
  Cargo numero de fecha del partido junto a los equipos y el
     resultado
  Si habia limite de fecha y ya se cumplio
    partidos = 0
  Si no
    Si el primer equipo perdio
      A[numero primer equipo][numero segundo equipo] += diferencia
          de puntos
    Si el segundo equipo perdio
      A[numero segundo equipo][numero primer equipo] += diferencia
          de puntos
    Fin si
  Fin si
  Decremento partidos en uno
Fin mientras
```

2. Generamos la matriz H.

```
Crear matriz H de tamanio equipos*equipos llena de 0s
Crear vector a de tamanio equipos lleno de 0s
Para i de 1 a equipos
  sumaFila = suma total de elementos en fila i de A
  Si la suma > 0
    La fila i de H seran todos los elementos de la fila i de A
    cada uno dividido por sumaFila
  Si no
```

```
    Pongo un 1 en la posicion i del vector a
  Fin si
Fin para
```

3. Generamos la matriz G.

```
Crear vector e de tamanio equipos lleno de 1s
Crear vector u para equipos invictos con todos elementos 1/equipos
Crear vector v para teletransportacion con todos elementos 1/
   equipos
Crear matriz G como resultado de hacer
G = c*[H + a*transponer(u)] + (1 - c)*e*transponer(v)
```

4. Busco el autovector asociado al autovalor $\lambda = 1$.

```
Generar matrices V y l de autovectores y autovalores de mi matriz
   G transpuesta
V es una matriz con autovectores como columnas
l es una matriz con autovalores en su diagonal
i = 1
Mientras valorAbsoluto(l[i][i] - 1) > 0.0001
  Incremento i en una unidad
Fin mientras
Crear vector x correspondiente a la columna i de la matriz V
```

5. Normalizo el vector solución.

```
x = valorAbsolutoACadaElemento(x)/sumaElementos(
   valorAbsolutoACadaElemento(x))
```

6. Genero matriz de solución.

```
Crear matriz S de tamanio equipos*2 y llena de 0s
Para i de 1 a equipos
  S[i][1] = i
  S[i][2] = x[i]
Fin para
```

7. Ordeno y escribo la matriz solución.

```
Ordenar crecientemente por segundo elemento de filas la matriz S (
   ranking)
Si tengo archivo con nombres de equipos
  Creo mapa teamcodes con cada numero de equipo asociado a su
     nombre
Fin si

Si tengo nombres de equipos cargados
  Para i de 0 a equipos - 1
    Escribo numero, nombre y ranking del equipo numero equipos - i
  Fin para
Si no
  Para i de 0 a equipos - 1
    Escribo numero y ranking del equipo numero equipos - i
  Fin para
Fin si
```

3.1.2. Puntaje AFA

Para el script utilizado para calcular el puntaje según el criterio de la AFA los parámetros son los siguientes:

- **in_filename:** Dirección de archivo con datos de entrada.
- **out_filename:** Dirección de archivo donde escribir el resultado final.
- **team_codes_filename:** Archivo opcional con el número de equipo asociado a su nombre correspondiente.
- **date_limit:** Campo opcional con la cantidad de fechas a tomar en cuenta (Por defecto toma todas las fechas disponibles).

Pasamos a describir el código:

1. Leemos la cantidad de equipos y partidos para después cargar todos los partidos disponibles y generar nuestra matriz S.

```
Cargar numero de partidos y equipos
Crear matriz S de tamanio equipos*2 llena de 0s
Mientras partidos > 0
  Cargo numero de fecha del partido junto a los equipos y el
     resultado
  Si habia limite de fecha y ya se cumplio
    partidos = 0
  Si no
    Si hubo empate
      S[numero primer equipo] += 1
      S[numero segundo equipo] += 1
    Si gano el primer equipo
      S[numero primer equipo] += 3
    Si gano el segundo equipo
      S[numero segundo equipo] += 3
    Fin si
  Fin si
  Decremento partidos en uno
Fin mientras
```

2. Ordeno y escribo la matriz solución.

```
Ordenar crecientemente por segundo elemento de filas la matriz S (
   ranking)
Si tengo archivo con nombres de equipos
  Creo mapa teamcodes con cada numero de equipo asociado a su
     nombre
Fin si

Si tengo nombres de equipos cargados
  Para i de 0 a equipos - 1
    Escribo numero, nombre y ranking del equipo numero equipos - i
  Fin para
Si no
  Para i de 0 a equipos - 1
    Escribo numero y ranking del equipo numero equipos - i
  Fin para
Fin si
```

4. Experimentación

En esta sección, se detallan los diferentes experimentos que realizamos para medir la eficiencia y resultados de los algoritmos implementados.

4.1. Page Rank

4.1.1. Instancias de prueba

Para generar grafos sobre los cuales experimentar tuvimos que obtener varias muestras. Por un lado, se utilizaron recursos de SNAP (http://snap.stanford.edu/) para las instancias mas grandes de grafos que utilizamos para medir tiempo y la convergencia del algoritmo Page Rank. Además, generamos instancias originales mediante las herramientas provistas por la cátedra otras extra generadas por nosotros.

4.1.2. Convergencia del algoritmo

Realizamos una serie de experimentos con el propósito de analizar la convergencia de la solución provista por el algoritmo a través de las iteraciones que el mismo realiza en su ciclo principal.
Para hacer esto, modificamos el algoritmo (no su procedimiento) con el fin de ir guardando las soluciones temporales en cada paso.
Con estos datos, medimos la Norma Manhattan (L1) entre las soluciones de cada par de iteraciones sucesivas, definida como:

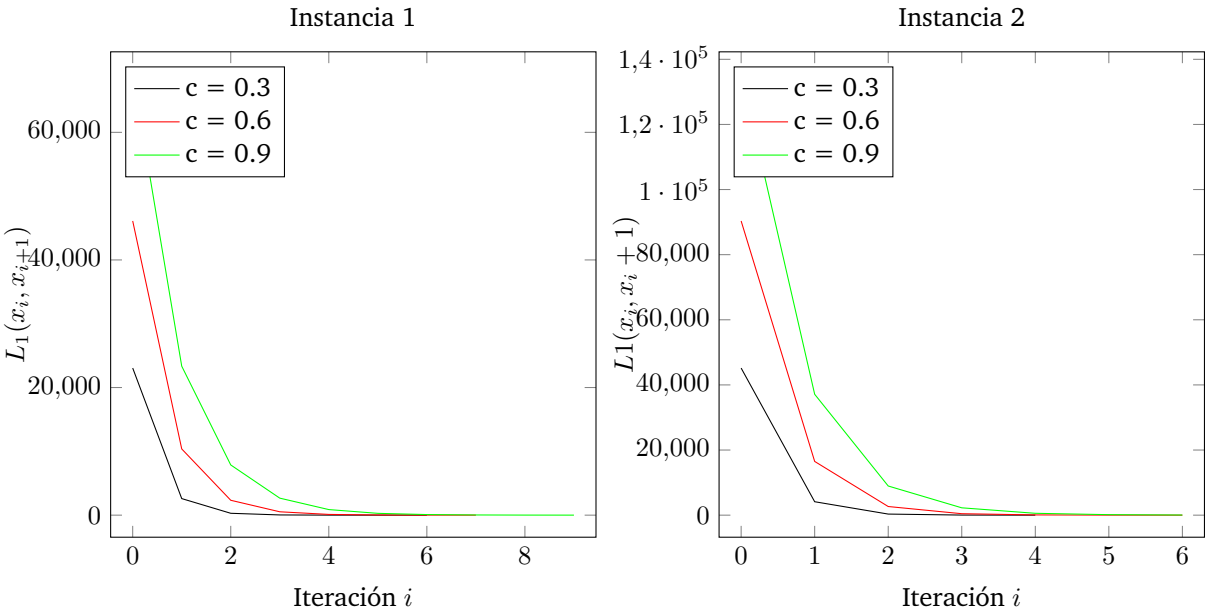
$$L_1(x, y) = \sum_{i=1}^{n} |x[i] - y[i]|$$

Siendo n la cantidad de nodos del grafo siendo estudiado, $x \in \mathbb{R}^n$ la solución del algoritmo en una iteración j e $y \in \mathbb{R}^n$ la de la iteración $j + 1$.
Esta es una forma intuitiva y logica de estudiar la convergencia, ya que al tomar la suma de los valores absolutos entre los componentes de cada vector, lo que realmente estamos haciendo es calcular cuanto varia la solución que estamos calculando a través de las iteraciones. Además, podemos asegurar que por el procedimiento de nuestro algoritmo esta diferencia va a ir disminuyendo, y cuando sea lo suficientemente chica el algoritmo va a terminar.

Estudiamos la convergencia a través de dos instancias de tamaño mediano-grande obtenidos de SNAP, mas precisamente:

- Instancia 1: 6301 nodos y 20777 ejes.
- Instancia 2: 10876 nodos y 39994 ejes.

Por lo que podemos decir que son considerables y relativamente esparsas (por ejemplo, el primero podria tener cerca de 40 millones de ejes si fuera completo, teniendo en cuenta que es un grafo dirigido).
A su vez, fuimos variando el componente c del algoritmo Page Rank entre 0.3, 0.6 y 0.9, el cual controla la importancia del viajero aleatorio, es decir, a menor c aumenta la probabilidad de que el usuario vaya a una página aleatoria desde la actual.
Veamos los resultados:



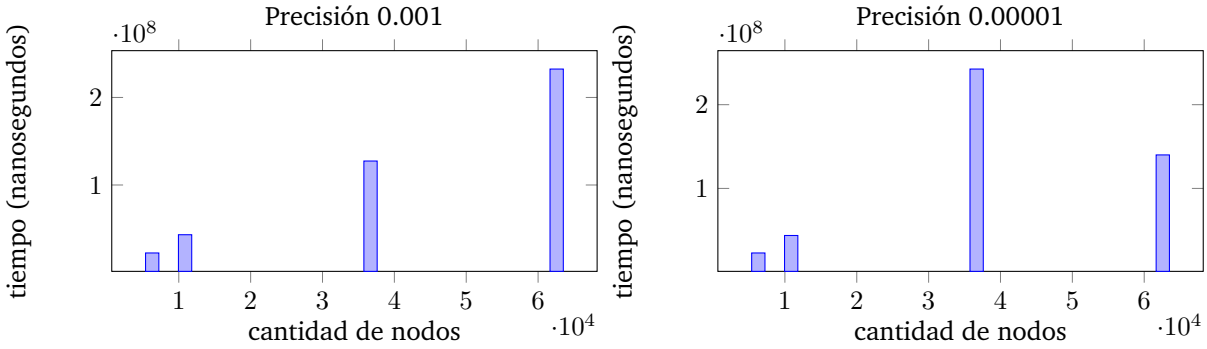
Siendo $L1(x_i, x_i + 1)$ (el eje y del gráfico) la distancia Manhattan entre los autovectores generados por el algoritmo entre las iteraciones i e $i + 1$.
Como podemos ver el algoritmo converge rápidamente, tomando a lo sumo 9 iteraciones para terminar de procesar instancias de datos relativamente grandes. Los resultados son parecidos para ambas instancias.
Adicionalmente, vemos que Page Rank converge de forma más rápida para valores de c más chicos, es decir, casos en los cuales es más probable que el usuario vaya a una página cualquiera desde la actual. Cuando esto sucede, la matriz de transición tiene valores más parejos dentro de cada columna y podemos decir entonces que el algoritmo soluciona el problema más rápido porque el sistema es más estable.

4.1.3. Tiempo de ejecución

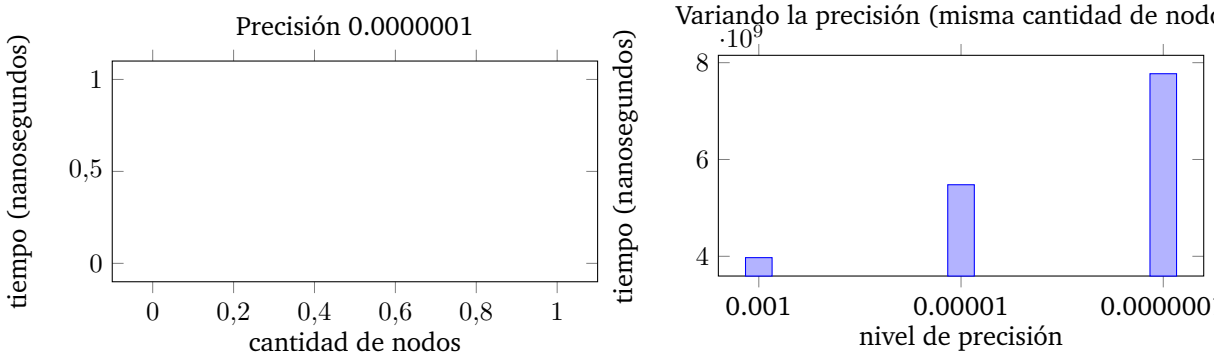
En esta sección, analizamos el tiempo de cómputo que emplea Page Rank.
Para tomar los tiempos, utilizamos la libreria chronos de C++ y pasamos las mediciones a nanosegundos. Todos las medidas fueron hechas bajo las mismas condiciones (procesos abiertos, computadora, alimentación y nivel de optimizaciones del compilador).
Con este fin, tomamos varias instancias de SNAP con distinta cantidad de nodos y vértices de forma que podamos ver la evolución de los tiempos en función del tamaño de la entrada.
Adicionalmente fuimos variando la precisión utilizada por el algoritmo, es decir, la cota que controla cuando la diferencia entre dos iteraciones del algoritmo sucesivas son lo suficientemente cercanas como para dejar de correrlo.
El factor c de Page Rank fue configurado en 0.85, ya que no afecta el tiempo de cómputo.
Las instancias utilizadas fueron:

- Instancia 1: 6301 nodos y 20777 ejes.
- Instancia 2: 10876 nodos y 39994 ejes.
- Instancia 3: 36682 nodos y 88328 ejes.
- Instancia 4: 62586 nodos y 147892 ejes.
- Instancia 5: 281903 nodos y 2312497 ejes.

Los resultados fueron los siguientes:



datafile.dat 0 152.9 0.2 159.3 0.4 165.2 0.6 170.5 0.8 174.8 1 178.2 1.2 180.5 1.4 182 1.6 183 1.8



183.9 2.0 184.5

4.1.4. Calidad de los resultados

Para evaluar la calidad de las soluciones provistas por Page Rank las comparamos contra el algoritmo InDeg, que ordena las páginas según la cantidad de links que van hacia ellas.
Por el lado de Page Rank, no utilizamos la versión Esparsa del algoritmo, ya que el tamaño de la entrada no lo requiere.
Las instancias que usamos para hacer estas comparaciones fueron:

- Instancia 1: 13 nodos y 18 ejes.
- Instancia 2: 13 nodos y 31 ejes.
- Instancia 3: 5 nodos y 20 ejes (grafo completo).

De forma de poder analizar los resultados facilmente, nos limitamos a instancias chicas.

Para la instancia 1, el valor c del algoritmo Page Rank fue configurado en 0.3, 0.6 y luego en 0.9 con el fin de introducir la posibilidad de que el navegante salte a otra página de forma aleatoria y ver cuánto afecta al resultado final. Esto no es asi para la instancia 3 ya que al ser un grafo donde la probabilidad de ir a las demás páginas es igual para todas, el concepto de navegante aleatorio es irrelevante porque es igual de probable que vaya a cualquier página desde el principio.

Los resultados fueron:

Instancia 1

Page Rank(c = 0.3)

Ranking	Página	Importancia
0	2	$8{,}95 \cdot 10^{-2}$
1	4	$8{,}38 \cdot 10^{-2}$
2	8	$8{,}01 \cdot 10^{-2}$
3	10	$8{,}01 \cdot 10^{-2}$
4	6	$8 \cdot 10^{-2}$
5	7	$7{,}62 \cdot 10^{-2}$
6	9	$7{,}61 \cdot 10^{-2}$
7	3	$7{,}42 \cdot 10^{-2}$
8	0	$7{,}2 \cdot 10^{-2}$
9	1	$7{,}2 \cdot 10^{-2}$
10	5	$7{,}2 \cdot 10^{-2}$
11	11	$7{,}2 \cdot 10^{-2}$
12	12	$7{,}2 \cdot 10^{-2}$

Page Rank(c = 0.6)

Ranking	Página	Importancia
0	2	0,14
1	4	0,11
2	10	$9{,}8 \cdot 10^{-2}$
3	8	$9{,}8 \cdot 10^{-2}$
4	6	$9{,}52 \cdot 10^{-2}$
5	7	$7{,}62 \cdot 10^{-2}$
6	9	$7{,}35 \cdot 10^{-2}$
7	3	$6{,}53 \cdot 10^{-2}$
8	0	$4{,}76 \cdot 10^{-2}$
9	1	$4{,}76 \cdot 10^{-2}$
10	5	$4{,}76 \cdot 10^{-2}$
11	11	$4{,}76 \cdot 10^{-2}$
12	12	$4{,}76 \cdot 10^{-2}$

Page Rank(c = 0.9)

Ranking	Página	Importancia
0	2	0,2
1	4	0,15
2	8	0,13
3	10	0,13
4	6	0,12
5	7	$8{,}11 \cdot 10^{-2}$
6	9	$7{,}05 \cdot 10^{-2}$
7	3	$5{,}76 \cdot 10^{-2}$
8	0	$1{,}18 \cdot 10^{-2}$
9	1	$1{,}18 \cdot 10^{-2}$
10	5	$1{,}18 \cdot 10^{-2}$
11	11	$1{,}18 \cdot 10^{-2}$
12	12	$1{,}18 \cdot 10^{-2}$

InDeg

Ranking	Página	In Degree
0	2	4
1	8	3
2	10	3
3	4	2
4	6	2
5	9	2
6	3	1
7	7	1
8	0	0
9	1	0
10	5	0
11	11	0
12	12	0

Por un lado, Page Rank obtiene un ranking con valores lógicos en el sentido de que el nodo con mas ejes hacia él es el de mayor importancia y los nodos con cantidad de links parecidos reciben un ranking similar.
Como podemos ver, los resultados de Page Rank no varian mucho, sus valores son afectados pero los rankings varian muy levemente. Esto es debido a que la cantidad de ejes no es de un tamaño tan grande como para que la variación de c afecte el ranking.
A su vez, InDeg da los resultados que esperabamos, ordenando según la cantidad de links entrantes.
En comparación, los algoritmos dan resultados parecidos de a secciones pero sí tiene diferencias en los primeros puestos. Esto se debe a que Page Rank detecta que por las probabililidades del recorrido que un navegante puede hacer, no necesariamente las páginas con mas links entrantes van a ser las más visitadas a futuro.

Aun asi, veamos otro ejemplo donde la cantidad de ejes sea un poco mayor:

Instancia 2

Page Rank(c = 0.3)

Ranking	Página	Importancia
0	1	$9{,}02 \cdot 10^{-2}$
1	2	$8{,}73 \cdot 10^{-2}$
2	8	$8{,}12 \cdot 10^{-2}$
3	6	$7{,}86 \cdot 10^{-2}$
4	4	$7{,}79 \cdot 10^{-2}$
5	10	$7{,}76 \cdot 10^{-2}$
6	7	$7{,}67 \cdot 10^{-2}$
7	9	$7{,}55 \cdot 10^{-2}$
8	5	$7{,}21 \cdot 10^{-2}$
9	3	$7{,}21 \cdot 10^{-2}$
10	0	$7{,}03 \cdot 10^{-2}$
11	11	$7{,}03 \cdot 10^{-2}$
12	12	$7{,}03 \cdot 10^{-2}$

Page Rank(c = 0.6)

Ranking	Página	Importancia
0	2	0,13
1	1	0,12
2	8	0,11
3	6	$8{,}77 \cdot 10^{-2}$
4	10	$8{,}72 \cdot 10^{-2}$
5	4	$8{,}46 \cdot 10^{-2}$
6	7	$7{,}89 \cdot 10^{-2}$
7	9	$7{,}18 \cdot 10^{-2}$
8	3	$5{,}53 \cdot 10^{-2}$
9	5	$5{,}39 \cdot 10^{-2}$
10	0	$4{,}22 \cdot 10^{-2}$
11	11	$4{,}22 \cdot 10^{-2}$
12	12	$4{,}22 \cdot 10^{-2}$

Page Rank(c = 0.9)

Ranking	Página	Importancia
0	2	0,21
1	8	0,15
2	10	0,12
3	1	$9{,}94 \cdot 10^{-2}$
4	6	$8{,}73 \cdot 10^{-2}$
5	4	$8{,}22 \cdot 10^{-2}$
6	9	$7{,}56 \cdot 10^{-2}$
7	7	$7{,}36 \cdot 10^{-2}$
8	3	$4{,}71 \cdot 10^{-2}$
9	5	$2{,}83 \cdot 10^{-2}$
10	0	$1{,}04 \cdot 10^{-2}$
11	11	$1{,}04 \cdot 10^{-2}$
12	12	$1{,}04 \cdot 10^{-2}$

InDeg

Ranking	Página	In Degree
0	1	5
1	8	5
2	2	4
3	4	3
4	6	3
5	7	3
6	9	3
7	10	3
8	3	1
9	5	1
10	0	0
11	11	0
12	12	0

En este caso ya podemos ver que el ranking varía más. Esto se debe a que ahora la cantidad de ejes es mayor con respecto a la cantidad de vértices. Por lo tanto, cuando vamos variando c, el algoritmo devuelve diferentes respuestas porque ahora la distribución de los links cobra más relevancia a comparación del factor ”navegante aleatorio”.
A diferencia del caso anterior, los resultados ya no son tan parecidos a los del algoritmo InDeg. De hecho, en solo un caso coincide PageRank con InDeg en el primer puesto del ranking. Lo que sugiere que la distribución de los ejes y su cantidad afectan la impredecibilidad del resultado.
Además, los ejes están distribuidos de tal manera que no haya grupos grandes aislados o grafos completos donde un grupo de vértices tenga una distribución de probabilidad muy pareja y nada de relación con otras componentes conexas.
Para ilustrar este caso veamos el siguiente ejemplo de un grafo completo:

Instancia 3

Page Rank

Ranking	Página	Importancia
0	0	0,2
1	1	0,2
2	2	0,2
3	3	0,2
4	4	0,2

InDeg

Ranking	Página	In Degree
0	0	4
1	1	4
2	2	4
3	3	4
4	4	4

Los resultados indican lo lógico, como es un grafo donde la probabilidad de ir a cualquier lado es igual, tanto Page Rank como InDeg resuelven el problema asignandole igual importancia a cada página. Por lo tanto concluimos que tanto la cantidad de ejes como la distribución son los factores que más contribuyen a los resultados de Page Rank.

4.2. GeM

4.2.1. Variando el parámetro c

Este experimento consite en variar el parámetro c y analizar como impacta esta variación en los resultados obtenidos al generar el ranking con GeM.

Recordemos que c es el coeficiente de amortiguación, que regula que tanto afecta el *navegante aleatorio* al resultado final.

Como es intuitivo de pensar, nuestra hipotesis en este experimento es que al tender c a cero aumenta la influencia del *navegante aleatorio* en el puntaje de cada página y, por lo tanto, más se parecen los puntajes de todas la páginas. Es decir, menos dejan de importar los links salientes de las distintas páginas del grafo original.

Por otro lado, cuando c tiende a uno se debilita la influencia del *navegante aleatorio*, causando que el puntaje dependa solo de la matriz $H + au^t$, descrita en la sección 2.2.1.

Se muestran a continuación dichos resultados para diferentes valores de c:

Posicion	Equipo	Puntaje
1	Vélez Sarsfield	0.033333
2	Unión	0.033333
3	Gimnasia y Esgrima (LP)	0.033333
4	Estudiantes (LP)	0.033333
5	Defensa y Justicia	0.033333
6	Crucero del Norte	0.033333
7	Colón	0.033333
⋮	⋮	⋮
24	Lanús	0.033333
25	San Lorenzo	0.033333
26	Rosario Central	0.033333
27	River Plate	0.033333
28	Racing Club	0.033333
29	Quilmes	0.033333
30	Olimpo	0.033333

Posicion	Equipo	Puntaje
1	Boca Juniors	0.051301
2	San Lorenzo	0.044563
3	River Plate	0.044200
4	Racing Club	0.040067
5	Aldosivi	0.039451
6	Rosario Central	0.039114
7	Quilmes	0.036699
⋮	⋮	⋮
24	Godoy Cruz	0.028356
25	Argentinos Juniors	0.028208
26	Temperley	0.027904
27	Crucero del Norte	0.027353
28	Nueva Chicago	0.026346
29	Atlético de Rafaela	0.025776
30	Colón	0.025577

Cuadro 1: A izquierda: puntajes obtenidos con $c = 0$, a derecha: puntajes obtenidos con $c = 0{,}3$

Posicion	Equipo	Puntaje
1	Boca Juniors	0.086019
2	Aldosivi	0.065353
3	River Plate	0.063500
4	San Lorenzo	0.062035
5	Rosario Central	0.048473
6	Racing Club	0.047878
7	San Martín (SJ)	0.043956
⋮	⋮	⋮
24	Godoy Cruz	0.017516
25	Temperley	0.016045
26	Crucero del Norte	0.016039
27	Argentinos Juniors	0.015398
28	Nueva Chicago	0.014232
29	Atlético de Rafaela	0.011388
30	Colón	0.010276

Posicion	Equipo	Puntaje
1	Boca Juniors	0.095290
2	Aldosivi	0.075151
3	River Plate	0.068142
4	San Lorenzo	0.065265
5	Rosario Central	0.050875
6	Racing Club	0.048824
7	San Martín (SJ)	0.047118
⋮	⋮	⋮
24	Godoy Cruz	0.014148
25	Crucero del Norte	0.013132
26	Temperley	0.012487
27	Argentinos Juniors	0.011286
28	Nueva Chicago	0.011239
29	Atlético de Rafaela	0.007559
30	Colón	0.006003

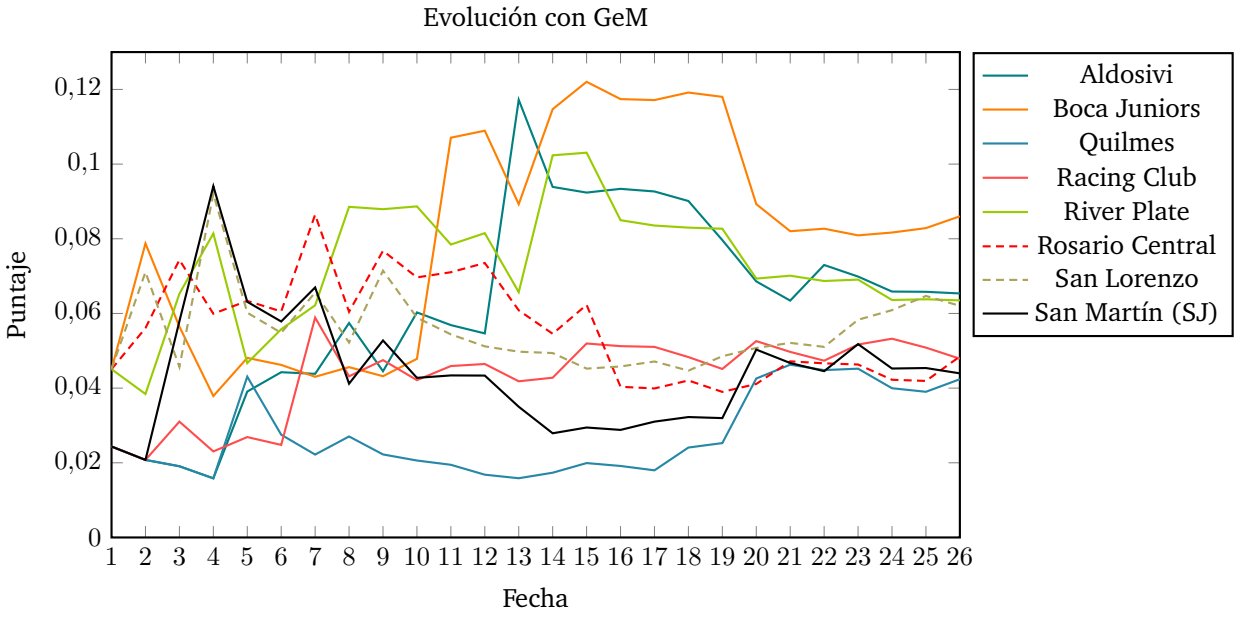
Cuadro 2: A izquierda: puntajes obtenidos con $c = 0{,}85$, a derecha: puntajes obtenidos con $c = 1$

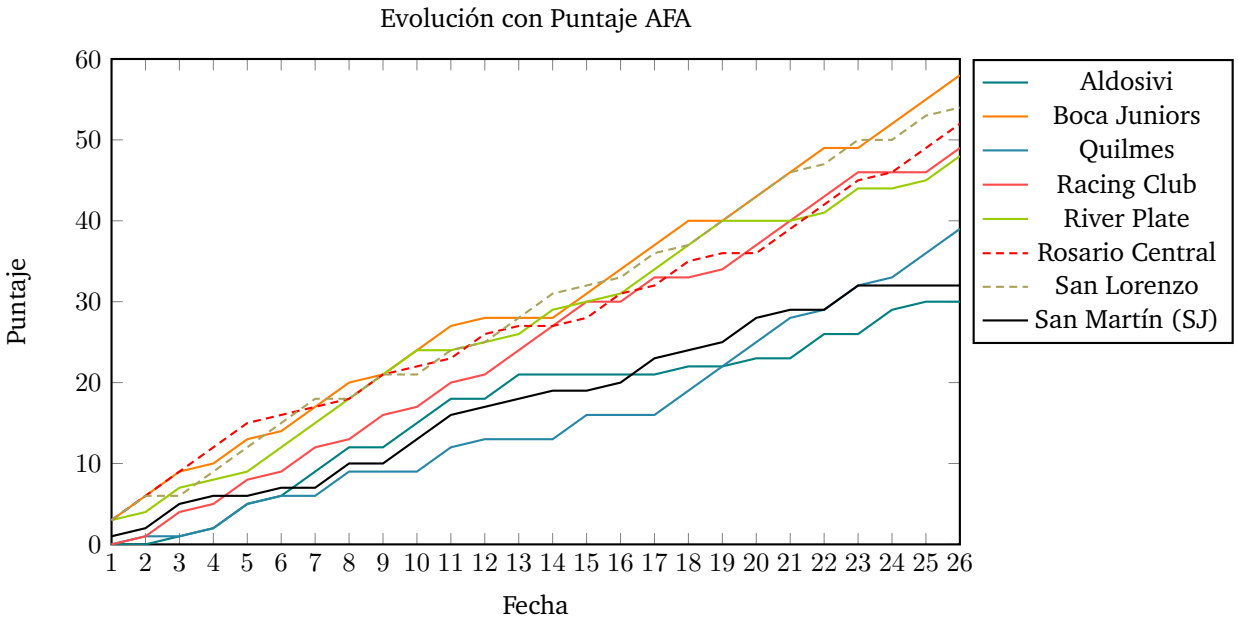
Analizando los resultados tenemos que:

- Para $c = 0$, la tabla de puntajes se condice con la hipótesis planteada. Los puntajes de los equipos no solo son parecidos, sino que son iguales. Y dicho puntaje es $0{,}0\hat{3} = 1/30 = 1/n$, dónde n es la cantidad de equipos totales.
- A medida que el c aumenta (desde $0{,}3$ a 1) las posiciones van convergiendo. Ya con $c = 0{,}3$, 6 de los 7 primeros equipos aparecen en los primeros 7 puestos con $c = 1$, y los últimos 7 equipos con $c = 0{,}3$ aparecen en los últimos 7 puestos con $c = 1$.
- A su vez, la diferencia entre los resultados con $c = 85$ y $c = 1$ es muy chica: de los 14 equipos mostrados en el Cuadro 2 solo dos de ellos cambian de posición (Crucero del Norte y Temperley).

En base a lo analizado, podemos concluir que los resultados del experimento corroboran las hipotesis planteadas.

4.2.2. Evolución del ranking




Evolución con Puntaje AFA
Puntaje
Fecha
0
10
20
30
40
50
60
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
Aldosivi
Boca Juniors
Quilmes
Racing Club
River Plate
Rosario Central
San Lorenzo
San Martín (SJ)

5. Conclusión

En este trabajo pudimos no solo modelar el sistema planteado, sino que apreciar y aprovechar las propiedades del mismo para así resolverlo con los métodos estudiados observando también las características de ellos.

Por un lado mediante la forma en la que construímos nuestro sistema probamos que se podía resolver con Eliminación Gaussiana sin pivoteo. Además produjimos una versión mejorada del algoritmo de eliminación donde aprovechando la propiedad de banda de la matriz del sistema, redujimos drásticamente la cantidad de operaciones necesarias para resolverla.

Así mismo, cabe destacar que al realizar operaciones con aritmética finita, tanto para la solución de los sistemas como para el cálculo de la isoterma donde la reutilización de datos arrastra error, no podemos garantizar que los resultados obtenidos sean exactos, pero dado que realizamos varias instancias de prueba con distintas metodologías y tomando números de condición aceptables, pudimos ver que los valores que obtuvimos eran coherentes a su contexto.

Luego, en lo que respecta el cálculo de la isoterma, al plantear diversas metodologías tuvimos la posiblidad de analizar y discutir los resultados de las mismas, donde en particular pudimos observar cómo al utilizar la búsqueda binaria podíamos llegar al grado de precisión que deseásemos y que para el método por promedio, al aumentar la cantidad de particiones mejoraba la aproximación, mientras que usando la regresión lineal, esta se ajustaba más a una función lineal que no reflejaba el comportamiento de la fórmula de calor, convergiendo así a un valor distinto tanto al del promedio como el de la búsqueda binaria.

Mediante estas aproximaciones, habiendo establecido previamente nuestro criterio para evaluar si una estructura se encontraba en peligro, llegamos a estimar qué sistemas eran seguros dentro de lo estipulado.

Para el análisis del tiempo de ejecución de una así como varias instancias del sistema modelado, vimos cómo se cumplían las complejidades teóricas de la resolución a través de Eliminación Gaussiana y LU. En este análisis corroboramos cómo si se trataba de una sola instancia la Eliminación Gaussiana presentaba una ventaja sobre LU, dado que el último debe calcular su factorización en su primer corrida, mientras que al subir el número de instancias el algoritmo para LU lograba un tiempo sumamente mejor que el de Eliminación Gaussiana, ya que con la factorización LU habiendo pagado un costo cúbico en la primer instancia, luego es del orden cuadrático contra el siempre cúbico de la Eliminación Gaussiana. Además en el análisis para el algoritmo de Eliminación Gaussiana con la optimización de banda llegamos a concluir que su tiempo de ejecución llegaba a reducirse al de orden cuadrático.

Por último, podemos mencionar algunos experimentos que podrían realizarse a futuro, como el aprovechamiento de la matriz banda en lo que es el algoritmo para la factorización LU, ya que esta optimización se realizó sólo para la Eliminación Gaussiana, junto a su correspondiente estudio de tiempo de ejecución. A su vez, quedó pendiente el realizar la mejora no únicamente en lo que son los tiempos de ejecución sino que el espacio que consume nuestro algoritmo dado que en la matriz banda gran parte de la misma permanece inalterada. También se podría haber profundizado en la experimentación del cálculo de la isoterma con sistemas donde la temperatura interna y externa no fueran constantes si no que tuvieran algún tipo de fluctuación donde se pudiera ver con más detalle cómo se comportaba cada método.