DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico N°1

Sistemas de ecuaciones lineales

Integrante	LU	Correo electrónico
Martinelli, Ezequiel	883/19	ezeaam@gmail.com
Mauro, Andrés	39/17	sebaastian_mauro@live.com
Tasat, Dylan	29/17	dylantasat11@gmail.com

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 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (++54 +11) 4576-3300

http://www.exactas.uba.ar

En este trabajo práctico analizaremos una implementación del algoritmo utilizado por Google llamado PageRank. Analizaremos sus fortalezas y debilidades ante diversos planteos elegidos estratégicamente. Plantearemos distintas propuestas y conclusiones para mejorar el algoritmo considerando sus puntos débiles.

Keywords: Eliminación Gaussiana, Page Rank, Google, Page Rank Variant.

Índice

1. Introducción

Los navegadores de internet utilizan varios criterios para darle prioridad a los resultados de una búsqueda, pero estos pueden ser inconsistentes devolviendo resultados de baja calidad. Entonces se utilizan alternativas como PageRank, un algoritmo utilizado por el motor de búsqueda de Google en sus inicios, nosotros vamos a estar analizando una variante del mismo en este informe.

Con este criterio se busca calcular el ranking de página, esto es un puntaje que nos dice qué tan "importante" es, utilizando el grafo de hipervínculos de la web.

Entonces primero queremos definir los links de una página, donde se representa la relación de un link desde una página j a una i con un 1 y la ausencia de este con un 0. De esta forma definimos una matriz W tal que:

$$w_{ij} = \begin{cases} 1 & \text{si existe un link de j a i} \\ 0 & \text{si no existe un link de j a i} \end{cases} \tag{1}$$

Cabe aclarar que no contaremos los links que tenga una página a ella misma, por lo tanto $w_{jj} = 0 \ \forall \ j \in 1, ..., n$.

Luego tenemos que definir la importancia para cada página. Para calcular este criterio vamos a tener en cuenta la importancia de las páginas que le apuntan por medio de un link, aumentando su importancia por cada uno, y la cantidad de enlaces que tienen, disminuyendo cada una su importancia añadida por cada enlace. Entonces determinamos su ranking con un valor numérico igual a la sumatoria de la clasificación de cada página que la apuntan divido la cantidad de enlaces que tienen. De esta forma para las n páginas el page ranking de la página i se expresa como:

$$x_i = \sum_{j=1}^{n} \frac{x_j}{c_j} w_{ij}$$

Donde c_j es la cantidad de links que salen de j ($c_j = \sum_{i=1}^{n} \ w_{ij}$) y x_j es el page ranking de la página j. Si no hay link de la página j a la página i $w_{ij} = 0$ y no suma al page ranking. Podemos definir el problema de encontrar cada uno de los page ranking como un sistema de ecuaciones $WDx = x$. D es una matriz diagonal tal que:

$$d_{jj} = \begin{cases} \frac{1}{c_j} & \text{si} \ \ c_j \neq 0 \\ 0 & \text{si} \ \ c_j = 0 \end{cases} \tag{2}$$

Al multiplicar W por D nos da como resultado una matriz R que tiene a cada elemento de su columnas dividido por el valor c_j correspondiente, por lo tanto $r_{ij} = \frac{w_{ij}}{c_j}$. Y resolver el sistema Rx = x es equivalente a encontrar cada uno de los x_i.

A partir de este sistema de ecuaciones se puede calcular el page ranking de cada una de las páginas, pero no refleja bien el comportamiento que puede tener un usuario. En realidad cuando estamos en una página podemos ir a otra totalmente aleatoria. Cada salto de una página a otra se puede realizar desde una página j a cualquiera con la que comparta un link o a cualquier otra.

Dada la probabilidad $p \in (0, 1)$ de que el usuario viaje de una página j a otra en sus links,

definimos una matriz A que vamos a utilizar para calcular el ranking de una página como:

$$a_{ij} = \begin{cases} \frac{1-p}{n} + p\frac{w_{ij}}{c_j} & \text{si} \quad c_j \neq 0 \\ \frac{1}{n} & \text{si} \quad c_j = 0 \end{cases} \tag{3}$$

Siendo $1/c_j$ la probabilidad de que escoja una página cualquiera entre sus c_j páginas y 1/n la probabilidad de escoger alguna cualquiera.

Este es el modelo del navegante aleatorio. Ahora el ranking de cada página se consigue resolviendo el siguiente sistema:

$$Ax = x$$

Veamos que si definimos:

$$z_j = \begin{cases} \frac{1-p}{n} & \text{si} \quad c_j \neq 0 \\ \frac{1}{n} & \text{si} \quad c_j = 0 \end{cases} \tag{4}$$

Podemos reescribir A como $pWD + z^T e$ siendo e un vector con un 1 en todas sus posiciones.

Definimos $B = pWD + z^T e$ Queremos ver que $a_{ij} = b_{ij}$

Veamos que como $(z^T e)_{ij}$ es la multiplicación de la fila i de z^T y la columna j de e. Pero como z^T es un vector columna y como matriz tiene n filas de 1 elemento y e tiene n columnas de un elemento que es 1. Entonces $\forall i \in 1, ..., n$

$$(z^T e)_{ij} = z_i^T$$

De esta forma cada una de las n filas de $z^T e \in \mathbb{R}^{nxn}$ son iguales al vector z. Por lo tanto cada columna j tiene únicamente (1 - p)/n si $c_j \neq 0$ y 1/n si $c_j = 0$.

Queremos ver que se cumple si $c_j = 0$. Como $(pWD)_{ij} = 0$ porque si $c_j = 0$ entonces $w_{ij} = 0$ y $(z^T e)_{ij} = 1/\text{n}$. Entonces $b_{ij} = (pWD)_{ij} + (z^T e)_{ij} = 1/n$.

Ahora si $c_j \neq 0$. Como pWD = pR y $r_{ij} = w_{ij}/c_j$ entonces $(pWD)_{ij} = p\frac{w_{ij}}{c_j}$ por como definimos R anteriormente, además $(z^T e)_{ij} = (1\text{-p})/\text{n}$, en ambos casos con $c_j \neq 0$. Entonces $b_{ij} = (pWD)_{ij} + (z^T e)_{ij} = (1-p)/n + p\frac{w_{ij}}{c_j}$.

Ahora puedo escribir b_{ij} como:

$$b_{ij} = \begin{cases} \frac{1-p}{n} + p\frac{w_{ij}}{c_j} & \text{si} \quad c_j \neq 0 \\ \frac{1}{n} & \text{si} \quad c_j = 0 \end{cases} \tag{5}$$

De esta forma $a_{ij} = b_{ij}\ \forall i, j \in 1, ..., n$. En concreto $A = B = pWD + z^T e$.

Ahora podemos reescribir la ecuación Ax = x como (I - pWD)x = γe con $\gamma = z^T x$

Para solucionar este sistema de ecuaciones vamos a utilizar el algoritmo de eliminación gaussiana[1]. Este busca triangular la matriz para que sea fácil encontrar una solución para el sistema.

No hace falta intercambiar filas si sabemos que nunca hay un 0 en la posición de la diagonal con la que estamos operando. Una forma de verlo es comprobando que la matriz tiene una factorización LU (cabe aclarar que no queremos ver si tiene una factorización PLU, es decir sin permutar la matriz); para probar esto podemos ver que la matriz es diagonal dominante,

[1]Accuracy and Stability of Numerical Algorithms, Nicholas Higham, SIAM, 2002. Página 158

es decir que:

$$|a_{ii}| > \sum_{J=1,j\neq i}^{n} |a_{i,j}| \ \forall i \in 1, ..., n$$

[2]

Entonces veamos qué es esa sumatoria para nuestra matriz I - pWD:

La matriz tiene solamente unos en la diagonal porque W tiene solamente ceros en esta y al restar las matrices no se le resta nada a la diagonal de la matriz identidad. Entonces lo que queremos probar es:

$$1 > \sum_{J=1,j\neq i}^{n} |(I - pWD)_{i,j}| \ \forall i \in 1, ..., n$$

Luego como las posiciones fuera de la diagonal se les resta pW_{ij}/c_j si $c_j \neq 0$ o 0 si $c_j = 0$, puede pasar que w_{ij} sea 0 o 1 y $p \in (0, 1)$. Entonces como $|(I - pWD)_{i,j}| > 0$ en el caso $c_j \neq 0 \wedge w_{ij} = 1$ entonces

$$\sum_{J=1,j\neq i}^{n} |(I - pWD)_{i,j}| \leq \sum_{J=1,j\neq i}^{n} |0 - p/c_j| \leq \sum_{J=1,j\neq i}^{n} p/c_j$$

Además como c_j es la suma de los links que tiene la página j $c_j < n$ y sabiendo que $\sum_{J=1,j\neq i}^{n} 1 \leq \sum_{J=1}^{n} 1$ entonces:

$$\sum_{J=1,j\neq i}^{n} p/c_j \leq \sum_{J=1,j\neq i}^{n} p/n \leq (p/n) \sum_{J=1}^{n} 1 = (p/n)n = p < 1$$

Porque $p \in (0, 1)$.

Entonces la matriz tiene factorización LU y puede factorizarse con el algoritmo de eliminación gaussiana sin pivoteo.

[2] Accuracy and Stability of Numerical Algorithms, Nicholas Higham, SIAM, 2002. Página 170

2. Desarrollo

Podemos observar que la matriz $I - pWD$ tiene un tamaño de $(páginas)^2$, nuestro primer desafío va a ser construir una estructura que nos permita almacenar y operar con esta matriz de forma eficiente, para lo cual nos aprovecharemos del hecho de que la cantidad de 1's va a ser significativamente inferior a la cantidad de 0's.

Como idea decidimos definir una clase $SparseMatrix$ que represente nuestra matriz, y establecimos que debe tener una interfaz muy similar a cualquier clase que represente matrices, pero internamente va a utilizar una estructura que aproveche la gran cantidad de ceros para reducir la complejidad de las operaciones.

Estas son las operaciones básicas que va a tener nuestra $SparseMatrix$:

- Crear una matriz: por conveniencia se va a considerar inicializada con ceros luego de su creación y las dimensiones de la matriz se establecen en este momento y no se van a poder modificar.
- Asignar un valor a la posición (i, j).
- Obtener el valor actual en la posición (i, j).
- Obtener tamaño.
- Sumar y restar con otra matriz (del mismo tamaño).
- Multiplicar por un escalar.
- Multiplicar por otra matriz (con las dimensiones que correspondan para que valga la multiplicación).

Las operaciones mencionadas se pueden encontrar normalmente en cualquier implementación de una matriz, sin embargo, a medida que fuimos avanzando con el desarrollo establecimos algunas funciones adicionales que rompen el encapsulamiento que venimos respetando, pero nos van a permitir aprovechar al máximo nuestra implementación sparse para optimizar las tareas que necesitamos realizar. Estas son:

- Asignar un valor a la posición (i, j) con precondición que estemos seguros que hay un cero en esa posición de la matriz, esto es más eficiente que la asignación mencionada previamente porque evita tener que buscar en nuestra estructura si se encuentra el elemento (i, j) y puede proceder directamente a su creación. Es muy útil para inicializar las matrices, especialmente porque la mayoría de nuestras operaciones van a funcionar por copia.
- Multiplicar cada columna por un elemento, esto es el equivalente a multiplicar por una matriz diagonal a derecha y lo vamos a utilizar para calcular $(pW)D$.
- Realizar la operación $fila_i(A) = fila_i(A) - k * col_j(A)$, esta es la operación básica que realiza la Eliminación Gaussiana en cada paso y va a ser la clave para que nuestro algoritmo sea eficiente aprovechando la matriz Sparse.

2.1. Generación de la matriz $I - pWD$

El primer paso de nuestro algoritmo va a ser generar la matriz que representa el sistema de ecuaciones que deseamos resolver.

Comencemos con la matriz W, este es nuestro parámetro principal y vamos a aprovechar la lectura del archivo para ir agregando los links como 1's en la matriz, en la posición que corresponda, para esto vamos a utilizar la función definida previamente que nos permite asignar elementos con precondición que haya un cero en esa posición, lo que va a ser cierto porque la matriz está inicialmente vacía, y no hay links repetidos en el archivo de entrada.

Sin embargo, ¿tiene sentido poner unos?, instantáneamente después de cargar el archivo estaríamos multiplicando esta matriz por p, sin embargo una matriz de todos 1's multiplicada por un escalar es lo mismo que haber puesto p en vez de 1 cuando cargamos la matriz, esto nos permite ahorrar un poco de tiempo.

También aprovechamos la carga del archivo para generar d, como un vector de tamaño "cantidad de páginas" que en el elemento j tiene la cantidad de 1's de la columna j, esto se hace fácil ya que cada vez que leemos un link con posición (i, j) podemos sumar 1 a la posición j del vector.

Luego para cada elemento de d vamos a aplicar $d_i = 1/d_i$, lo que nos va a dar como resultado que d sea exactamente la diagonal de la matriz D mencionada en la introducción. Sabemos que multiplicar una matriz A por una diagonal D es lo mismo que aplicar $col_i(A) = col_i(A) * d_{ii}$ a cada una de las columnas, para esto vamos a utilizar nuestra operación definida previamente en la estructura $SparseMatrix$ que recibe la matriz y el vector con los elementos de la diagonal de D y lo resuelve de forma eficiente.

Finalmente ya obtuvimos nuestra matriz pWD, ahora solo queda hacer I menos eso, para no complicar las cosas decidimos generar la matriz identidad y luego aplicar la operación de resta de nuestra implementación, quizás se podría hacer algo un poco más eficiente considerando que es una resta bastante especial, sin embargo no es tan costoso por lo tanto fuimos por esta opción.

2.2. Eliminación Gaussiana

Una vez generada nuestra matriz, el próximo paso es triangularla para poder resolver el sistema, para esto vamos a utilizar el algoritmo de Eliminación Gaussiana, el cual consta de realizar operaciones de filas que no alteran el resultado del sistema para ir poniendo ceros debajo de la diagonal.

Para ser más precisos, el algoritmo recorre cada una de las columnas de la matriz, luego para cada fila debajo de la diagonal aplicamos la operación $fila_i(A) = fila_i(A) + c * fila_j(A)$ donde $fila_i(A)$ va a representar a la fila actual, y $fila_j(A)$ a la fila correspondiente al elemento de la diagonal de la columna actual. Finalmente c se define como $c := \frac{A_{ij}}{A_{jj}}$, esto nos va a asegurar que el elemento A_{ij} se convierta en cero, y como esto se repite para todos los elementos debajo de la diagonal, vamos a terminar llenando la parte inferior de la diagonal con ceros.

Para que este algoritmo tan simple funcione correctamente debemos hacer algunas consideraciones:

- Que los elementos por los que estamos dividiendo no se van a anular, pero como vimos en la introducción, el sistema tiene una única solución y por lo tanto va a tener una factorización LU sin ceros en la diagonal de U.
- Cada vez que apliquemos una de las operaciones de fila, vamos a tener que hacer $b_i = b_i - c * b_j$ para no alterar el resultado del sistema, siendo b el vector con los números que se encuentran a la derecha del igual en nuestro sistema de ecuaciones, este vector representa al γe mencionado en la introducción y por lo tanto comienza lleno de unos.
- La operación $fila_i(A) = fila_i(A) + c * fila_j(A)$ la definimos previamente como una de las operaciones que nos provee la $SparseMatrix$ y si la matriz es suficientemente sparse va a ser más eficiente que $O(n)$, en consecuencia, si la matriz no es sparse probablemente vayamos a tener un costo mayor en complejidad.
- Hay que tener en cuenta que si seguimos el orden de ejecución que se puede ver en la figura 1 nos va a asegurar que las operaciones de filas no inserten valores no-nulos en las celdas que ya nulificamos.

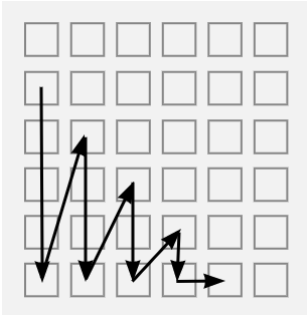

Figura 1: Orden de ejecución de la Eliminación Gaussiana

2.3. Resolución del sistema

Una vez que resolvimos la triangulación de la matriz, ahora tenemos una matriz con ceros debajo de la diagonal y un vector b resultante de las operaciones de filas aplicadas en el proceso de triangulación.

Una vez tenemos esto podemos realizar fácilmente en costo de $O(n^2)$ la resolución del sistema con la siguiente fórmula:

$$x_i = (b_i - \sum_{j=i+1}^{n} a_{ij} * x_j)/a_{ii} \tag{6}$$

2.4. Implementación de las operaciones

Hasta ahora definimos una interfaz de la $SparseMatrix$ y operamos con la misma, sin embargo va a depender mucho de la implementación de esta matriz la complejidad resultante del algoritmo.

Lo que estamos seguros es que vamos a tener una estructura de datos que sólo almacene los valores no-nulos debido a que la matriz es sparse.

Vamos a ver lo que hacen las distintas operaciones, y sus complejidades:

- Crear una matriz:
  - Inicializamos la estructura de datos vacía (porque comienza llena de ceros).
  - Guardamos el tamaño de la matriz.

  Complejidad: O(Inicializar ED[3])

- Asignar un valor a la posición (i, j).
  - Buscamos si existe el elemento en nuestra estructura de datos.

  Si encontramos el elemento:

  Si el valor a asignar es cero:

  - Borramos el elemento en nuestra estructura de datos

  Si el valor a asignar es distinto de cero:

  - Modificamos el valor por el nuevo

  Si no encontramos el elemento:

  Si el valor a asignar es cero:

  - No hacemos nada

  Si el valor a asignar es distinto de cero:

  - Creamos un nuevo elemento en la estructura de datos

  Complejidad: O(Buscar ED) + O(Borrar ED) + O(Modificar ED) + O(Crear ED)

- Obtener el valor actual en la posición (i, j).
  - Buscamos si existe y si no devolvemos cero.

  Complejidad: O(Buscar ED)

- Sumar y restar con otra matriz $(A \pm B)$.
  - Creamos la matriz resultado.
  - Insertamos las celdas de A en el resultado (copiamos A).
  - Iteramos los no-nulos de B y los sumamos/restamos al resultado.

  Complejidad: O(Crear matriz) + O(No-nulos de A) * O(Asignación ciega) + O(No-nulos de B) * (O(Asignación) + O(Obtener valor))

- Multiplicar por un escalar.[4]
  - Creamos la matriz resultado.
  - Recorremos las celdas e insertamos en el resultado el valor de la celda por el escalar en el resultado.

  Complejidad: O(Crear matriz) + O(No-nulos de A) * O(Asignación ciega)

[3] ED = operación en la estructura de datos.

[4] Esta operación no se utiliza en la versión final pero la dejamos por completitud.

- Multiplicar por otra matriz.[5]
  - Creamos la matriz resultado.
  - Iteramos cada fila de A y hacemos $< fila_i(A), col_i(B) >$ y lo insertamos en el resultado.

  Complejidad: O(n^3) * (O(Obtener valor) + O(Asignación ciega))

- Asignación ciega (i, j).
  Si el valor es distinto de cero:
  - Agregamos una celda con el valor correspondiente.

  Complejidad: O(Crear ED)

- Multiplicar cada columna por un elemento.
  - Para cada celda, si está en la posición (i, j), entonces multiplicamos su valor por el elemento j del vector parámetro.

  Complejidad: O(No-nulos de A) * (O(Asignación ciega) + O(Consultar elemento del vector)[6])

- Realizar la operación $fila_i(A) = fila_i(A) - c * col_j(A)$.
  - Para cada columna k, si el elemento de la fila j es no-nulo, lo multiplicamos por c, se lo restamos al valor de la fila i, y finalmente lo asignamos en la posición (i, k).

  Complejidad: O(n) * (O(Asignación) + O(Obtener valor))

  Implementación alternativa: probamos otra opción en función de la cantidad de celdas en vez de recorrer las columnas pero resultó ser mucho más ineficiente que la versión previa a medida que se iba haciendo menos sparse con las operaciones de la Eliminación Gaussiana, la dejaremos como referencia:

  - Para cada celda de la matriz, si la fila es j, buscamos cuanto es el valor de la fila i para esa columna y asignamos el resultado de la cuenta.

  Complejidad: O(No-nulos de A) * (O(Asignación) + O(Obtener valor))

Luego lo que más nos va a interesar a nosotros es la complejidad de la Eliminación Gaussiana, ya que probablemente supere en complejidad a todo lo demás.

O(Eliminación Gaussiana) = O(n^2) * O($fila_i(A) = fila_i(A) - c * col_j(A)$) = O($n^2$) * O($n$) * (O(Asignación) + O(Obtener valor)) = O(n^3) * (O(Asignación) + O(Obtener valor))

[5] Esta operación no se utiliza en la versión final pero la dejamos por completitud.
[6] En std:vector es O(1): https://cplusplus.com/reference/vector/vector/operator[]/

Inicialmente intentamos utilizar la primera implementación que se nos ocurrió almacenar un vector de tuplas del tipo $(i, j, valor)$, y en este vector van a estar guardados todos los elementos que tengan valor distinto de cero.

Como la matriz es sparse consideramos que esta implementación podría llegar a ser suficientemente buena, sin embargo luego de algunas pruebas llegamos a la conclusión que era muy costoso al punto de que los casos de prueba proporcionados por la cátedra que deberían tardar unos 30 segundos, parecía que fueran a tardar varias horas.

Luego de analizar la situación, concluimos que las complejidades eran O(Obtener valor) = O(n) y O(Asignación) = O(n), lo que lleva a que nuestro algoritmo de Eliminación Gaussiana termine con una complejidad de O(n^4).

Finalmente decidimos utilizar la implementación de tablas de hash de $C++$ que se llama unordered_map, donde el índice va a ser una túpla (i, j) con la posición del elemento, y el valor va a ser de tipo double.

La tabla de hash tiene la característica de tener consulta y asignación en O(1) amortizado, ya que estamos haciendo la amortización al recorrer toda la parte inferior de la matriz más toda la fila actual en el algoritmo de Eliminación Gaussiana, podemos afirmar que en la práctica obtenemos unas complejidades de O(Obtener valor) = O(1) y O(Asignación) = O(1) amortizadas, y por lo tanto terminamos en un algoritmo de Eliminación Gaussiana de O(n^3).

Puede parecer que no hay nada especial en esto, ya que esta es la complejidad standard para la Eliminación Gaussiana sin matrices sparse, sin embargo, la constante que multiplica al n^3 va a ser extremadamente mejor porque la mayoría de las operaciones no hacen nada si el elemento es cero, y como la matriz es sparse vas a tener prácticamente la mayoría de iteraciones que comparan si el valor es cero y no hacen nada más.

3. Experimentación

3.1. Generación de instancias

Para las siguientes experimentaciones utilizamos distintos conjuntos de instancias específicas para los objetivos de cada experimento. Para el análisis cuantitativo utilizamos algunas de las instancias y soluciones provistas por la cátedra (test_30_segundos, test_15_segundos, test_aleatorio_desordenado, test_completo, test_trivial).
Para el análisis de la complejidad temporal de nuestra solución generamos instancias particulares, que no modelan ninguna situación específica pero nos permiten evaluar complejidad en cuestión. Estas instancias denominadas çomplejidad temporal"no son más que matrices en donde variamos la cantidad de páginas que tienen contienen (es decir columnas de la matriz) y a su vez la densidad de la misma (cuanta celdas contienen valores distintos a 0), de esta manera generamos instancias que poseen n paginas (con $n \in \{1, 25, 50, 75, 100\}$)y la cantidad de celdas variando entre $\{0, 0{,}25n^2, 0{,}5n^2, 0{,}75n^2, n^2\}$
Para el análisis cualitativo en el que vamos a estudiar los rankings obtenidos en función de la estructura del grafo (que modela la web) y del valor de P, propusimos tres dataset distintos que representan distintas situaciones posibles

- 3 Principales: las instancias de este dataset representan una web en la cual hay 3 páginas principales con pociones claras en el ranking (1er 2do 3er puesto) ya que esta determinada la cantidad de páginas que apuntan a ellas. Y el Resto de links de la web esta distribuido de manera aleatoria y uniforme en resto de páginas. En la figura 2 se muestra una representación gráfica de estas instancias.
  En la construcción de estas instancias se determino que las páginas que ocupen el 1er, 2do y 3er puesto sean las páginas 1, 2 y 3 respectivamente. A su vez, al momento de determinar la cantidad de links que las apuntan fueron asignados en función de los links totales, el 50 % para la página 1, el 25 % para la página 2 y el 10 % para la página 3. El 15 % restante de links en la web fue asignado con una distribución uniforme al resto de páginas en la web

- 3 Principales y un Anillo: En este dataset se modela el caso en el que la web contiene 3 páginas principales con claros puestos en el ranking, pero también contiene un sub conjunto de páginas que no son apuntadas por otras (es decir el grafo no es conexo), pero entre si la i-ésima página del sub conjunto apunta a la i+1 y a la i-1 pero a su vez estas dos apuntan a la i-ésima. De esta forma se construye un anillo de páginas que se puntan entre si y finalmente, todas las páginas del anillo apuntan a una central. La figura 3 ilustra un ejemplo de este caso.
  En la construcción de estas instancias, se dejo fija la cantidad total de páginas y la cantidad de links (links no pertenecientes al anillo) en la web con valores 1000 y 501 respectivamente. También se tomo la decisión de que las páginas que conforman al anillo sean las últimas de la matriz y en particular, la página central es la última de todas (es decir la página 1000). De esta forma para variar el tamaño del anillo variamos a partir de que página comienza a formarse. Cabe aclarar que la cantidad de links asignados a

las páginas que no forman parte del anillo fue distribuida de igual maneja al dataset anterior

- 3 Principales apuntando a: las instancias de este dataset representan una web con 3 páginas claramente principales (construidas de la misma forma que en los dataset anteriores) las cuales apuntan a otra página elegida al azar entre el resto de la web, dentro del dataset se contemplan los casos en que la pagina elegida es apuntada por la página 1, la 1 y la 2 o las 3 principales. Para facilitar el análisis del ranking, se determino que la página apuntada sea la página 4 en todos los casos. En estas instancias se fijo la cantidad de páginas totales en 1000 y la cantidad total de enlaces en 500. En la figura 4 se ejemplifica este caso

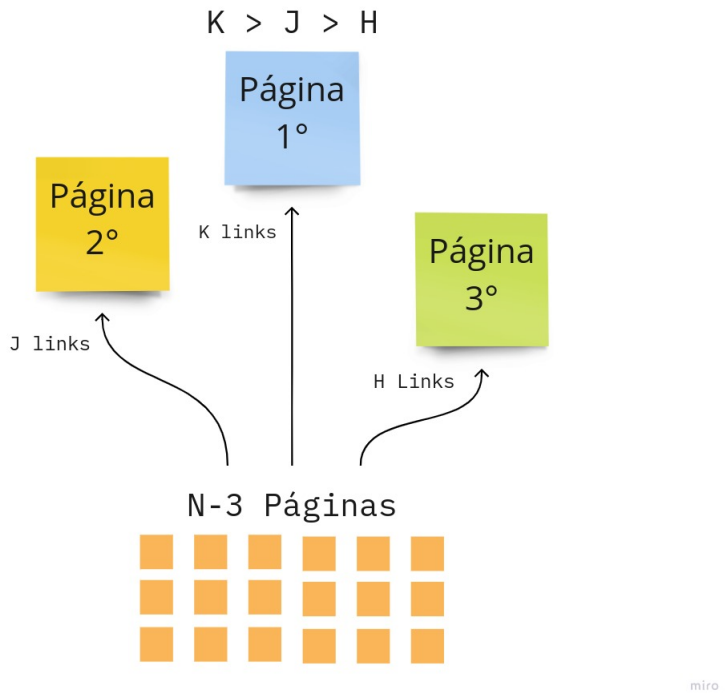

Figura 2: Una web con N páginas de las cuales 3 son apuntadas por una cantidad fija y el resto son apuntadas por una cantidad aleatoria páginas

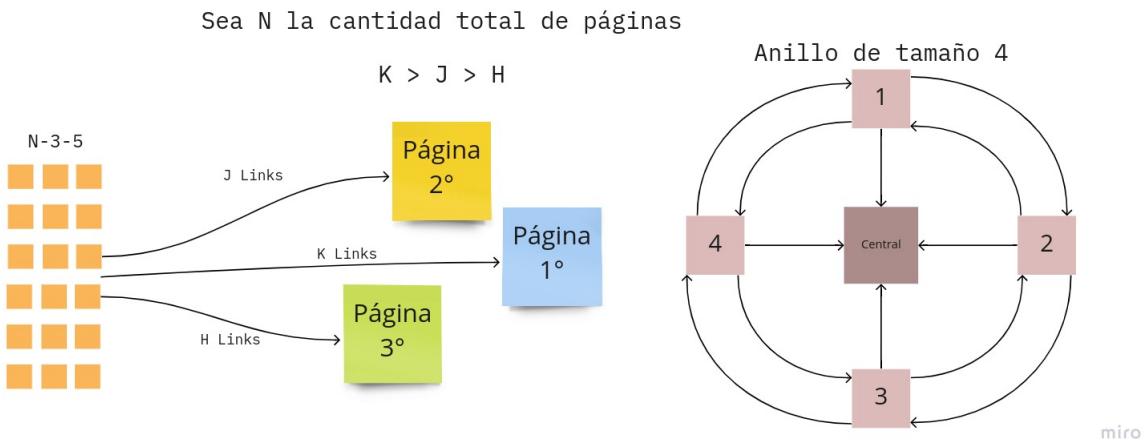

Figura 3: Una web 3 páginas principales y un anillo de tamaño 4

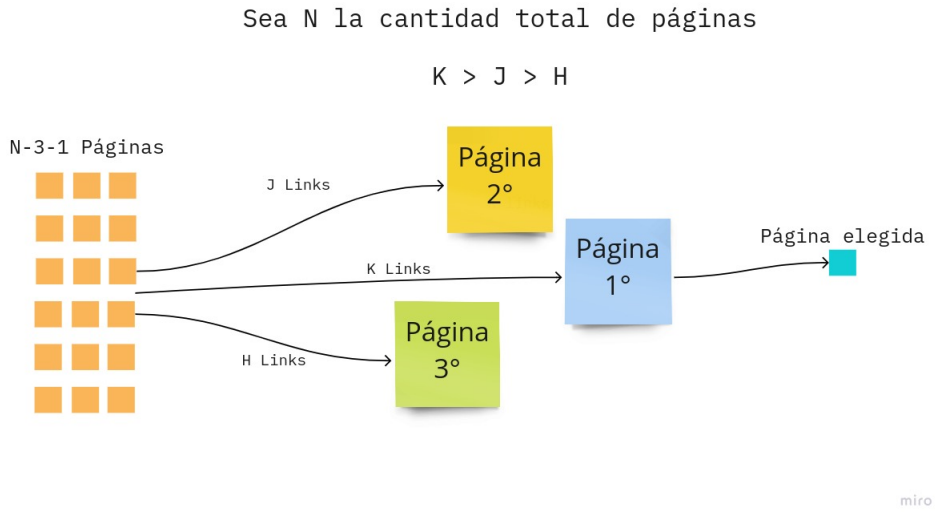

Figura 4: Una web 3 páginas principales donde la pagina más apuntada linkea a una aleatoria

3.2. Ejecución de experimentos

3.2.1. Experimento 1

Como primer experimento vamos a realizar un análisis de la complejidad temporal de nuestra solución, con el fin de comprobar la escalabilidad de este método como solución al problema de generar un ranking de páginas web a partir de los enlaces entre ellas.

Con este experimento tenemos dos hipótesis:

- Primero, cuando la matriz es sparse nuestro algoritmo va a tener una complejidad similar a n^3.
- Segundo, cuando la matriz no es sparse la complejidad va a empeorar hasta el punto de ser similar a n^4.

Para esto ejecutaremos nuestro código con instancias de diferentes tamaños (cantidad de páginas y enlaces) 5 veces cada una y tomaremos la media del tiempo total que le llevo al algoritmo. Luego vamos a graficar los tiempos en función de la cantidad de nodos de la matriz y su cantidad de links.

Este análisis se realizara con instancias del dataset çomplejidad temporal".

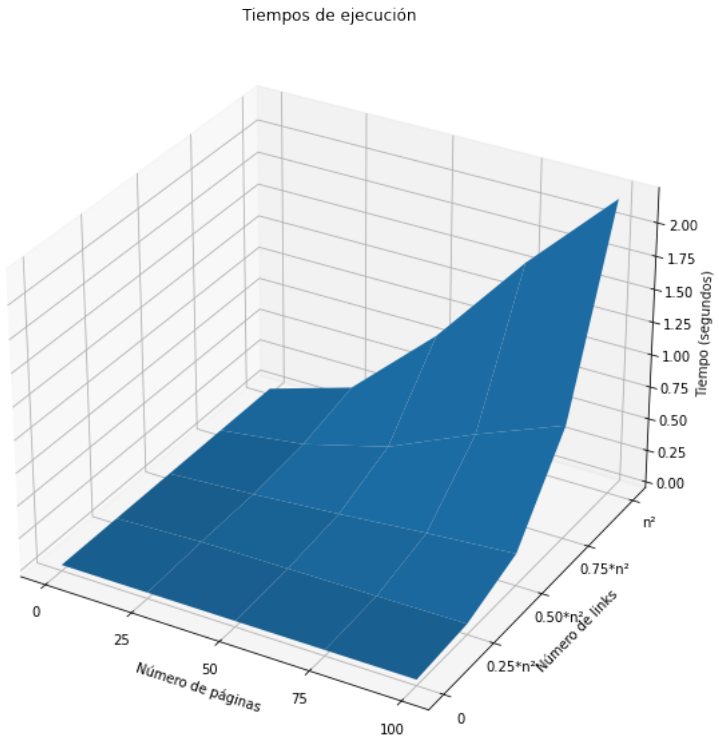

Figura 5: Complejidad temporal

El gráfico resultante muestra con claridad como la complejidad del algoritmo pareciera se polinomial (de exponente mayor a 2) en función tanto de la cantidad de páginas en la web

como respecto de la cantidad de links que contiene Cabe aclarar que el desvío estándar de cada una de estas ejecuciones fue menor que 0,012450, lo cual nos indica que el gráfico fue generado con datos bastante consistentes. También podemos ver que mientras menos sparce es la matriz, la complejidad respecto de la cantidad de páginas pareciera seguir siendo polinomial pero de mayor grado.

3.2.2. Experimento 2

Como segundo experimento nos proponemos probar empíricamente que la solución del sistema $I - pWD = \gamma e$ efectivamente es solución del sistema original $|Ax - x| = 0$. Nuestra hipótesis es que efectivamente los rankings encontrados por nuestra solución son lo suficientemente cercanos a autovalores del A.

Para llevar a cabo este experimento vamos a resolver el sistema corriendo el algoritmo que implementamos con instancias. Con herramientas de python vamos a armar la matriz A del sistema original, luego la multiplicamos por los resultados encontrados y probamos que la diferencia entre estos sea menor a $\epsilon = 1 * 10^{-4}$ al correr las instancias del dataset "3 Principales".

```
9.838638884666667e-05
count    3.000000e+03
mean     1.522058e-05
std      1.176954e-05
min      3.814386e-09
25%      8.505050e-06
50%      1.459979e-05
75%      1.631162e-05
max      9.838639e-05
dtype: float64
```

Figura 6: Resultados de calcular $|Ax - x| = 0$

En la figura 6 se pueden apreciar la media (mean), el desvío estándar (std) y la mediana (50 %) del experimento. Todos los resultados tenían una diferencia menor al ϵ propuesto y, por lo tanto, nuestros resultados son lo suficientemente cercanos a los del sistema original.

Cabe notar que si bien la máxima diferencia de los experimentos se acerca peligrosamente al ϵ propuesto este parece un outlier porque al ver la media, la mediana y la varianza este valor se aleja mucho del resto, además el 75 % sigue alejándose lo suficiente como para reforzar esta idea. Esto indica algo importante sobre nuestro algoritmo, a pesar de que este da un promedio muy cercano a la realidad del problema podemos tener resultados que se acercan peligrosamente a superar margen propuesto y por lo tanto algunos de nuestros resultados pueden no ser tan precisos como querríamos.

3.2.3. Experimento 3

Como parte de la experimentación cualitativa procederemos a calcular el error absoluto de los rankings encontrados por nuestra solución respecto de los que nos brindo la cátedra. Nuestra hipótesis es que el error debe ser muy cercano a cero.

Para realizar este experimento vamos a tomar los test de la cátedra, correrlos con nuestra implementación y le restamos a cada uno de nuestros resultados el dado por la cátedra (la resta se hace posición a posición del ranking). Luego tomamos el módulo del resultado y vemos que sea menor a $\epsilon = 1 * 10^{-4}$.

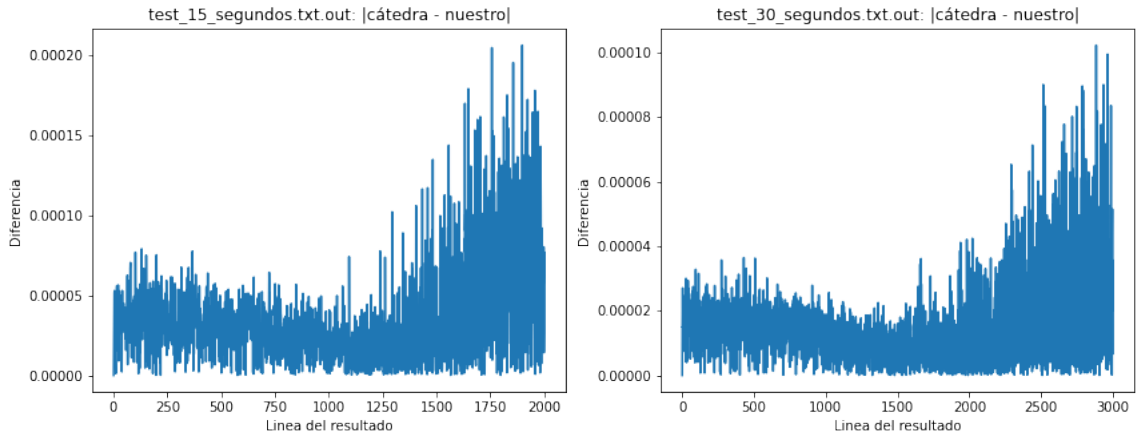

Figura 7: Error absoluto respecto de las instancias de la cátedra

En la figura 7 se muestra el error absoluto posición a posición del ranking generado nuestra solución respecto a los ranking brindados por la cátedra para las instancias test_15_segundos.txt y test_30_segundos.txt graficamos únicamente estos ya que los errores hallados para las otras instancias fue 0. En este gráfico podemos comprobar que el erro absoluto hallado para cada posición no supero $1 * 10^{-4}$. Una conclusión que podemos verificar del gráfico es que las ultimas posiciones son las que marcan mayor error ya que son los elementos que se ubican más a la derecha en la matriz, por ende a la hora de la triangulación sus elementos fueron más manipulados y por ende acumulando mayor error.

3.2.4. Experimento 4

En el siguiente experimento nos proponemos analizar el ranking resultante de nuestra solución en instancias aparentemente triviales, a modo de comprobar empíricamente que los resultados son los esperados. A su vez nos disponemos a realizar un análisis de dichos resultados con el fin de entender como se ven afectadas las puntuaciones de las páginas más relevantes con respecto a la cantidad total de links en la web. Para esto, correremos el ejecutable con instancias que poseen 1000 páginas y variaremos la cantidad de links entre 200, 400, 600, 800, 1000.

Estos Experimentos se realizaran sobre instancias del dataset 3 Principales, ya que modelan un escenario en el cual claramente hay tres páginas cuyas posiciones deberían ser los primeros tres puestos.

Esperamos que en el ranking resultante de nuestra solución, la diferencia de puntaje entre las páginas 1 y la 3 incremente en función del aumento de la cantidad de links totales de la web. Esto debería suceder porque en nuestras instancias el porcentaje de links que puntan a cada una de las 3 páginas es el mismo, con lo cual al incrementar los links una mayor cantidad de los nuevos apuntan página 1 en comparación con los que apuntan a la página 3.

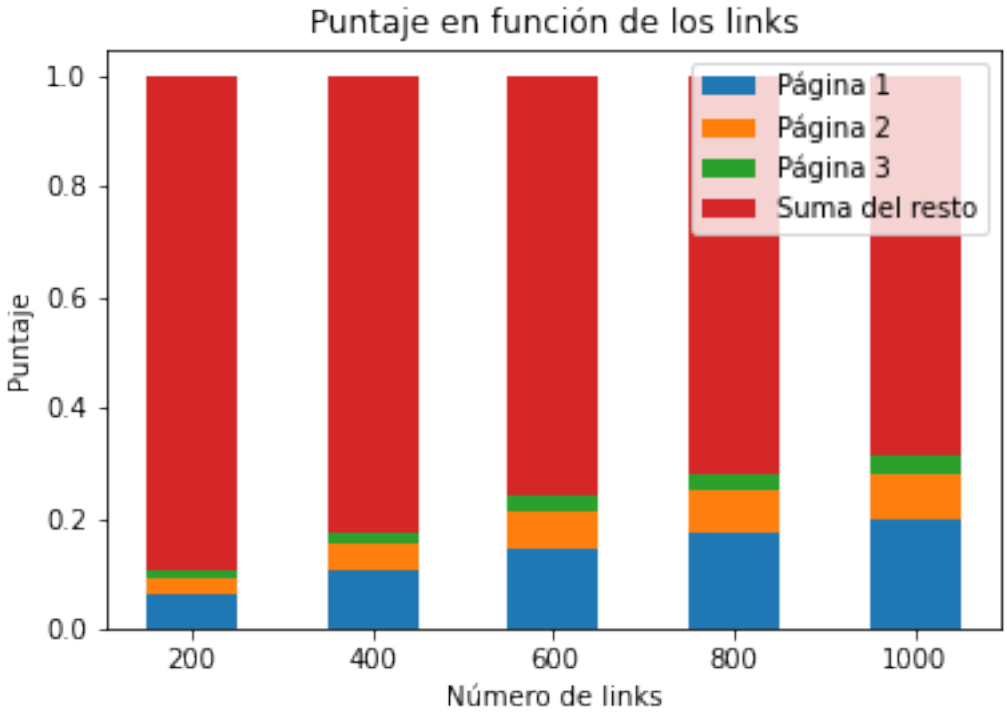

Figura 8: Puntaje del ranking variando la cantidad de links

Como se muestra en la figura 8 por como esta definido el ranking de las páginas, a medida que aumentamos la cantidad de links. Se puede apreciar que la diferencia entre el ranking de las páginas 1 y 3 aumenta. Esto denota que la relevancia de las páginas aumenta en mayor o menor medida respecto a la cantidad de links. En particular, mientras el porcentaje de links totales que apunte a una sea mayor su diferencia con el resto crecerá y, siempre y cuando el porcentaje de links que apunten a cada una no cambie, el orden del ranking será el mismo.

3.2.5. Experimento 5

En el siguiente experimento nos proponemos analizar como afecta al ranking resultante la variable P. Con el fin de comprender la relevancia de contemplar la navegación aleatoria por la web y averiguar cual es el peso que nuestra solución da a los enlaces en donde las probabilidades de seguirlos varía.

Para ello, correremos nuestro ejecutable con distintas instancias del dataset 3 Principales que tienen 1000 páginas y 500 links variado el valor de P entre 0, 0.2, 0.4, 0.6, 0.8, 1. Elegimos ese deta set en particular porque creemos que el escenario que plantea se asemeja a la realidad.

Nuestra hipótesis es que entre más grande sea el P, aumentara el puntaje de las páginas 1, 2 y 3, dado que mientras mayor sea la probabilidad de seguir un link, más importantes son a

la hora de formar el ranking. Por otro lado mientras menor sea el valor de P (es decir menos probabilidad de tomar un link) el puntaje de las páginas principales debería parecerse al de resto de páginas, ya que el peso de cada uno de esos link seria menor.

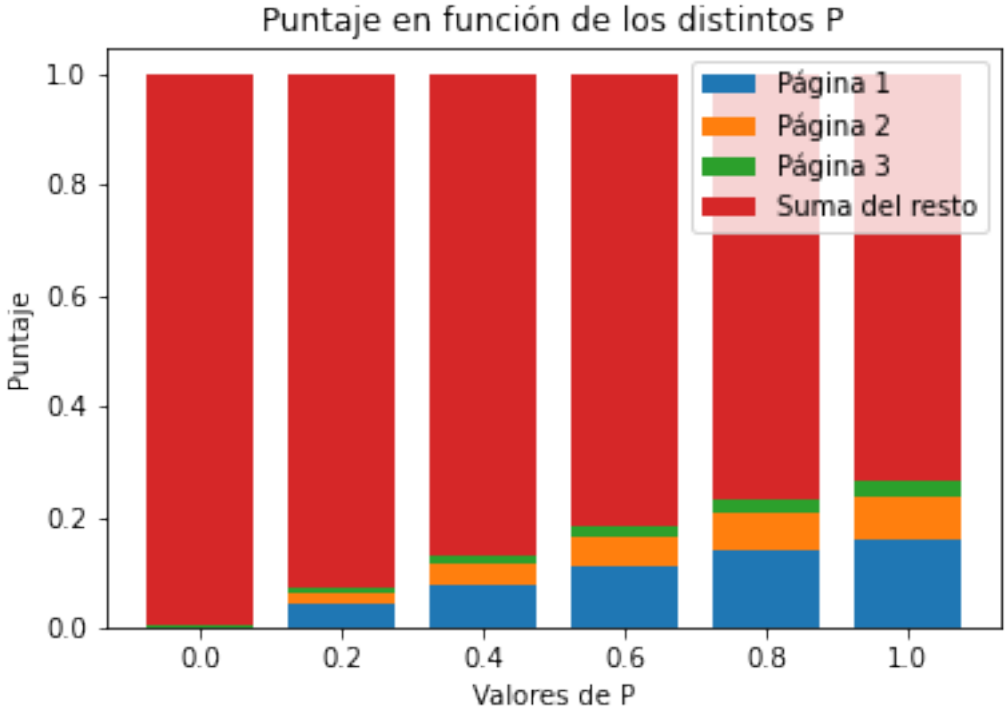

Figura 9: Puntajes de las páginas fijando la cantidad de páginas en 1000, la cantidad de enlaces en 500 y variando P

En la figura 9 podemos apreciar que para una misma instancia el valor de las páginas principales aumentan en ranking a medida que aumenta el P. Al mismo tiempo podemos apreciar que desde p = 0.2 hasta p = 1 el las páginas 1, 2 y 3 mantienen su orden en el ranking siendo la 1 siempre la mayor y la 3 siempre la menor independientemente del valor de P (no nulo), mientras que con un P nulo los puntajes de las páginas son muy similares entre sí; esto también se refleja en la suma del resto siendo este mayor entre menor sea el valor de P.

Con nuestros resultados podemos concluir que el puntaje de las páginas que reciben mayor cantidad de links crece más que resto. Por el otro lado entre mayor sea el valor de P, las páginas no principales pierden puntuación total frente a las principales, manteniéndose las páginas principales como las más relevantes. De esta forma las páginas no principales resultan menos beneficiadas en el ranking entre más cerca del 1 este la probabilidad P, mientras que las principales resultan más beneficiadas teniendo más relevancia en el ranking.

3.2.6. Experimento 6

En este experimento vamos analizar como se comporta nuestro algoritmo de rankeo ante, el particular escenario que plantean las instancias del data set "3 Principales y un Anillo". Estos escenarios nos interesan porque plantean una situación posible en la que el ranking resultante

no se corresponda con el deseado. Para averiguar que tan robusto ante estos casos es nuestro algoritmo, vamos a variar el tamaño del anillo dejando fija la cantidad páginas, links fuera del anillo y p; cabe aclarar que las páginas 1, 2 y 3 reciben 250, 125 y 50 links respectivamente. Luego analizaremos con cuanta facilidad se puede posicionar la pagina central de anillo en el podio.

Creemos que si bien para que el ranking de la página central del anillo sea mayor que el resto de páginas no relevantes (lo que significa ubicarla en el cuarto puesto) este anillo no deberá ser muy grande (respecto del tamaño total de la web), intuimos que para posicionarlo dentro de los primeros dos puestos el tamaño del anillo debería ser considerablemente grande. O más específicamente j/2 páginas en el anillo, siendo j la cantidad de links apuntando a la página 2.

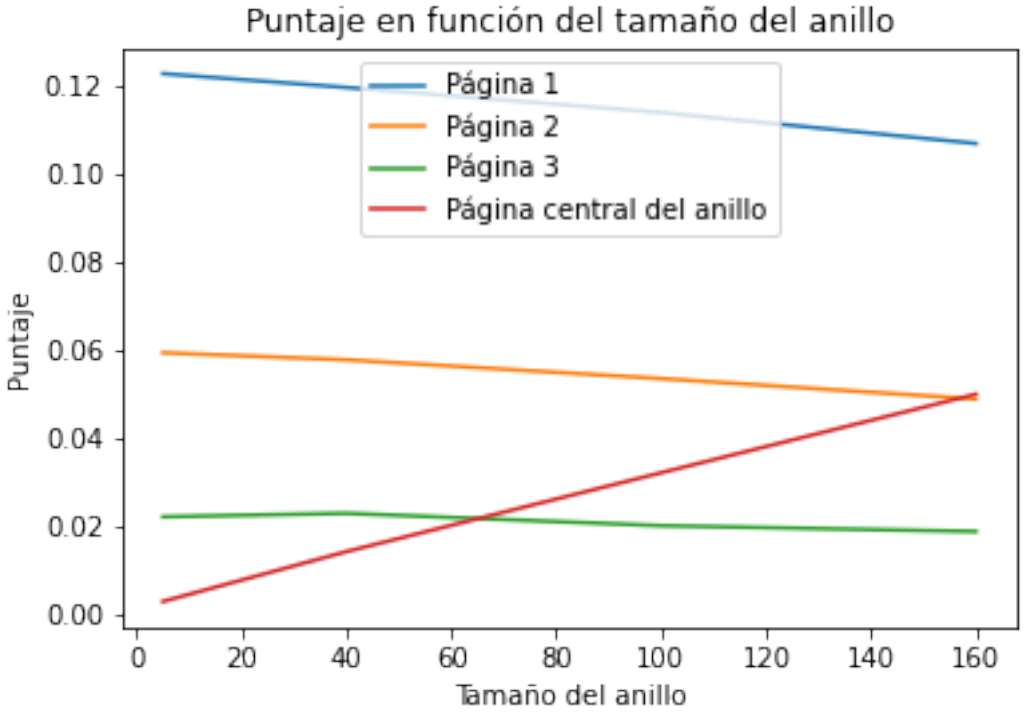

Figura 10: Puntajes de las páginas fijando la cantidad de páginas en 1000, la cantidad de enlaces en 500 y P = 0,7

Por lo visto en la experimentación mientras más aumenta el tamaño del anillo el ranking las páginas principales empieza a disminuir entre más relevante se hace la página central consolidándola en un cuarto puesto de manera rápida. Por el otro lado la página central no llega al segundo puesto en el tamaño de anillo esperado, nuestra intuición no era errónea en que el tamaño del anillo debía ser grande en (proporción a la cantidad total de páginas en al web), pero asumimos que sería más chico de lo que resulto tras la experimentación.

Entonces nuestra conjetura sería que debe recibir una cantidad de links parecida a la de la página 2, pero tampoco sucede. Mientras que la página central y la página 2 reciben la misma cantidad de links, sus puntajes se acercan pero no llegan a ser igual, mientras que recién estando cerca de un anillo de tamaño 160 es cuando cambia el orden entre estas dos. Al ver esto notamos que el peso de los links del anillo es menor que los de las páginas con

distribución uniforme.

Entonces si queremos modificar de forma artificial el ranking de los primeros puestos con la introducción de un anillo esto nos va a conllevar un costo mayor, en cantidad de n páginas, al de estar apuntando con n links a una página específica con páginas menos relevantes generadas a partir de una distribución uniforme.

3.2.7. Experimento 7

Con este experimento pretendemos analizar el comportamiento de nuestra solución ante otras instancias particulares, en concreto las situaciones modeladas con el data set "3 Principales apuntando a". Donde resulta que las páginas mejor rankeadas apuntan a su vez a una página no tan puntuada, nos proponemos estudiar el peso y la relevancia que nuestra solución da a estos links y a su vez averiguar como varía la puntuación de las páginas principales en estos casos particulares. Para ello, correremos nuestra solución con las 3 instancias del data set. Nuestra hipótesis es que, como el puntaje de las páginas principales es mucho mayor que el puntaje del resto, estos enlaces tendrán un peso muy considerable. Esto sumado a que el puntaje de la página que apunte a la aleatoria disminuirá (ya que parte de el será compartido con la pagina apuntada) podría no solo destacar a la página apuntada respecto del resto, sino que incluso podría posicionarla por encima de las que originalmente eran principales.

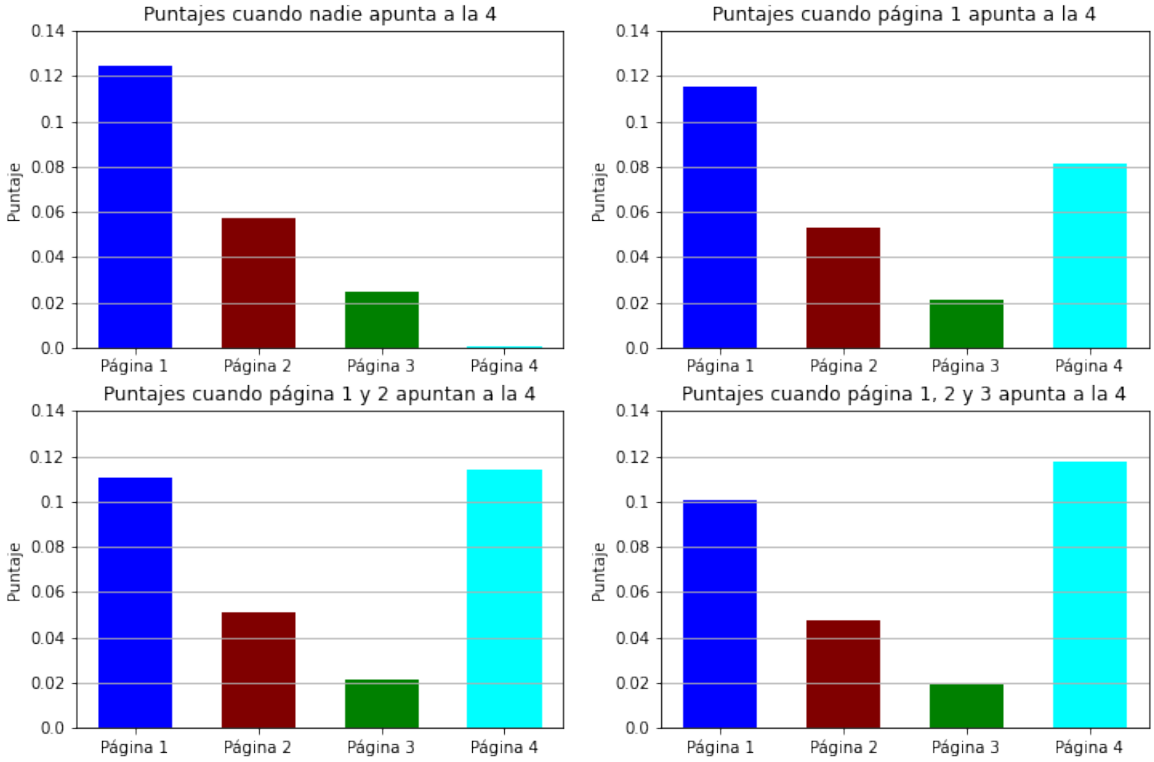

Figura 11: Puntajes de las páginas 1, 2, 3 y 4 variando los enlaces a la página 4

Analizando los resultado obtenidos, notamos de que si bien nuestra hipótesis fue acertada

los resultados fueron aún más trascendentales.
En la primer fila de imágenes de la figura 11 se puede observar que con solo el enlace de la pagina 1 a la 4 basta para posicionar esta última en el segundo puesto por encima de las páginas 2 y 3, esto nos indica que el peso de ese enlace es mayor que la suma de los pesos de los enlaces que apuntan a la página 2. En las siguiente fila de imágenes de la figura 11 se observa como los enlaces del restos de las páginas relevantes posicionan a esta última en el primer puesto y a su vez, que la puntuación que recibe la página 4 es claramente superior a la suma del puntaje de las páginas 2 y 3, esto explica porque en la ultima figura de todas se ve que el link de la pagina 3 a la 4 no añade mucho más puntaje a la 4. También se puede observar como el puntaje de las páginas decrece al apuntar a la página 4, esto se debe a que ceden parte de su puntaje a la página apuntada.

4. Conclusiones

El algoritmo funciona muy bien teniendo en cuenta lo simple que es, sin embargo tiene de fondo una teoría bastante fuerte que lo respalda, eso no significa que no tenga problemas, como demostramos en el Experimento 6, puede ser manipulado para darle un alto puntaje a una página en específico, también vimos en el Experimento 7 cómo las páginas importantes van a tener mucho poder de manipulación en los resultados del ranking y esto puede generar negocios ilícitos a través de la publicidad en páginas importantes para subir tus resultados en el buscador.

Resultó bastante difícil concretar una implementación de matriz sparse que sea suficientemente eficiente para correr el algoritmo en tiempo razonable, especialmente porque, a medida que va progresando la Eliminación Gaussiana, la matriz se va llenando con cosas no-nulas en la parte superior de la diagonal y esto hace que se reduzca la cantidad de ceros, lo que provoca que no podamos aprovechar los métodos que son muy rápidos con una matriz muy sparse pero muy lentos si no lo es.

La variante del caminante aleatorio que estamos utilizando en este trabajo demuestra sus resultados de forma evidente en el Experimento 5, donde un P bajo provoca que se acerque a un algoritmo que salta aleatoriamente entre páginas para hacer el ranking, lo que resulta en un resultado poco informativo, pero en el otro extremo, un P muy alto representa la variante original del algoritmo, donde siempre se siguen los links, lo que genera muchísimo sesgo hacia las páginas importantes y termina generando problemas como en los Experimentos 6 y 7.

5. Referencias

1. Elminacion Gaussiana: https://www.uv.es/~diaz/mn/node29.html

2. Accuracy and Stability of Numerical Algorithms, Nicholas Higham, SIAM, 2002.

3. Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. Computer networks and ISDN systems, 30(1-7):107117, 1998.