Generación de Órdenes Totales mediante el Vector de Perron de Cadenas de Markov Irreducibles

Silvio Vileriño *(svilerino@gmail.com)*, Sacha Kantor *(sacha.kantor+exactas@gmail.com)* y Nahuel Lascano *(laski.nahuel@gmail.com)*

Resumen

La elaboración de órdenes es uno de los problemas más frecuentes que se da en nuestra sociedad. Virtualmente todos los ámbitos conocidos (deportes, laboral, lúdico, salarial, etc) enfrentan este problema, que será más o menos complejo según el contexto. A lo largo de este trabajo estudiamos este problema en el ámbito de las páginas web: ¿cuándo una página debería estar por encima de otra?

Presentamos un modelo de generación de rankings orientado al ordenamiento de páginas web para resultados de motores de búsqueda. Pasamos entonces a dar un método numérico que nos asegura la existencia de una solucón para el modelo dado, para luego implementar el mismo y analizarlo empíricamente tanto en aspectos computacionales como del dominio del problema. Por último, analizamos le extensión de este modelo al ámbito de las competiciones deportivas. Presentamos el modelo, lo implementamos y analizamos sus comportamientos empíricos respecto de los rankings reales utilizados por las competencias, así como con casos abstractos ideales para poder constratar la calidad de los resultados contra algo más determinístico que las competencias deportivas.

Palabras Clave

PageRank, Autovectores, Matriz Estocástica, Motores de Búsqueda

Índice

1. INTRODUCCIÓN TEÓRICA

EN el siguiente trabajo se aborda una problemática tan común a muchos ámbitos distintos: la generación de *Rankings* u órdenes. El objeto de estudio en nuestro caso será la elaboración de estos rankings para en el ámbito de las páginas web, para luego tratar de extrapolar los métodos encontrados y aplicarlos a los participantes de distintos eventos deportivos.

Esta introducción está orientada a explicar muy brevemente: las características y motivaciones detrás de la necesidad de generar estos rankings en el ámbito de las páginas web, el modelo propuesto orignalmente por el motor de búsqueda *Google* con una breve fundamentación matemática del mismo, para finalizar introduciendo los conceptos utilizados para adaptar dicho modelo a los eventos deportivos.

1.1. El Problema

En su publicación original en 1998 [**Brin1998**], Brin y Page comentan algunas de las características principales de su motor de búsqueda de la *World Wide Web* con el que apuntan a solucionar las problemáticas existentes y crecientes de la Internet de aquella época (que incluso al día de hoy siguen siendo problemas vigentes y estudiados).

En dicho documento se hace hincapié, entre otras cosas, en como el crecimiento exponencial tanto en el uso de internet como de la cantidad de información[a] han generado un verdadero problema a la hora de encontrar el contenido buscado por los usuarios. Sin mencionar que estos nuevos usuarios, en su gran mayoría, no son programadores ni administradores de red ni nada similar. Es decir, estos nuevos usuarios no tienen un conocimiento (ni se espera que lo tengan, si uno espera que el uso de internet sea masivo) para realizar *queries* complejas en algún lenguaje de búsqueda[b]. E incluso para usuarios con dicho conocimento, sería mucho más practico poder realizar la búsqueda y obtener los resultados correctos sin tener que recurrir a dicha sintaxis.

Junto con el crecimiento de la información, que a su vez se da desde distintas partes del mundo (generando una problemática geográfica de acceso a la información), el problema de las búsquedas de contenido específico se ve aún más difícil: ahora buscar la información correcta en un universo cada vez más grande y diverso requiere de métodos más elaborados.

En este contexto, entonces, diseñar un motor de búsqueda que pueda escalar junto con internet requiere enfrentar distintos desafíos: hay que ir descubriendo rápidamente los nuevos contenidos que aparecen y desaparecen de la red[c], usar eficientemente el espacio para almacenar índices (y opcionalmente documentos enteros), usar sistemas de indexado que puedan procesar todo este volumen gigantesco de información también de manera eficiente; y, obviamente, un sistema de consultas que pueda resolver las mismas de manera rápida con un ratio de entrada de hasta miles de consultas por segundo que devuelva resultados relevantes.

En resúmen, se observa que muchos de los problemas que fueron (y todavía siguen) surgiendo son consecuencia de la velocidad con la que esta escalando internet. El objetivo de este trabajo es estudiar un aspecto del último de los items mencionados en el párrafo anterior: el orden de los resultados de una consulta. No estudiaremos como indexar los resultados, ni nos consentraremos en realizar las consultas eficientemente respecto de la complejidad (espacial o temporal) sino que nos focalizaremos en el orden de los resultados. De manera abstracta y, a modo de ejemplo, si uno consigue de alguna manera ordenar todas las páginas web por orden de relevancia (para alguna definición de relevancia), luego resolver una consulta implicaría únicamente filtrar los resultados manteniendo el orden dado.

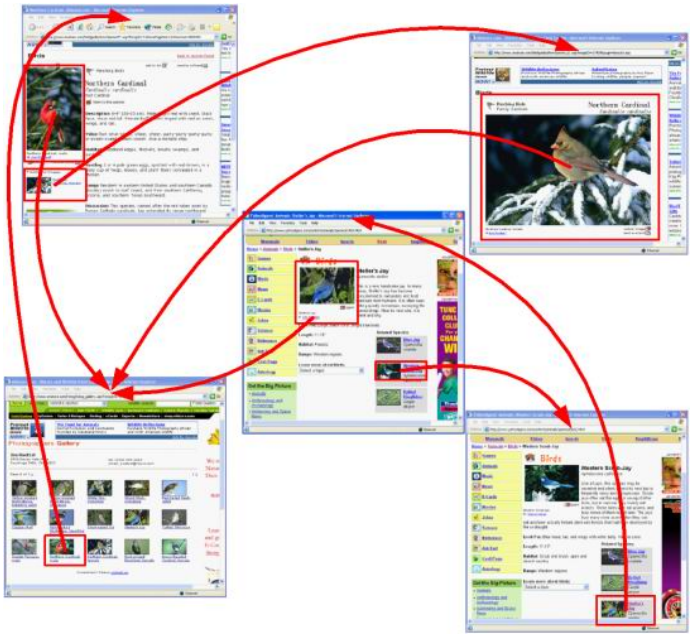


Figura 1. Páginas web y sus vínculos [**uchicago˙research**]

1.2. El Modelo

El modelo propuesto por Brin y Page para *ordenar* o crear un ranking fue llamado *PageRank*. El concepto

a. a.k.a: Páginas web.
b. Como por ejemplo *ANSI SQL*.
c. a.k.a. Crawling.

principal del mismo es el de asignar a todas las páginas
un puntaje basado en los hipervínculos o *links* de otras
páginas web que referencian a la página que está siendo evaluada. Estos links son denomminados *backlinks* de cada página.

Una forma gráfica de entender esto es observar a la web como un grafo dirigido de conectividad. Cada página web estaría representada por un nodo distinto y cada link sería un eje dirigido, yendo desde el nodo que representa a la página con el link hasta el nodo homónimo de la página referenciada por el mismo. El modelo de PageRank propone observar a estos ejes como un sistema de "votos", donde cada página "vota" mediante sus links a las páginas que consideran importantes.

Un orden muy simple sería, entonces, tomar el grado de entrada de cada nodo[a], u en palabras de lo que representa el grafo, la cantidad de backlinks de una página, y ordenar en orden descendente por dicho valor. Pero esta solución presenta un problema inmediato: todos los "votos" o ejes salientes tienen el mismo peso, sin importar de quién vienen. Si una página web tiene millones de links, todos sus votos valen lo mismo que una página que sólo tiene diez. A su vez, los votos de una página que no es votada por nadie valen lo mismo que los de una página que es votada por todos los demás.

Para solucionar, o al menos mitigar, este comportamiento, PageRank propone cuantificar los "votos" según de quién vienen, y de su puntaje. Esto inicialmente parecería un razonamiento circular ya que el puntaje de cada página dependerá del puntaje de las demás, que a su vez dependen del puntaje de esta página. Pero utilizando esta idea, con alguna modificación, se llegará a un sistema de ecuaciones lineales, el cual permite resolver justamente este tipo de cálculos donde hay muchas ecuaciones a ser satisfechas que utilizan las mismas variables (en este caso, los puntajes de las páginas).

Volviendo a la cuantificación mencionada, la misma se basa en calcular el puntaje de una página como la suma de los puntajes de las páginas de sus backlinks. Pero esto podría llevar a que una página con buen puntaje, reparta infinitos votos (mediante la generación de links) de gran valor. Para evitar esto se decide que el valor de los votos de una página sea distribuido equitativamente entre todos sus votos. Formalizando un poco matemáticamente, decimos que el puntaje x_k de una página k con el conjunto L_k de páginas que tienen algún link a k se define como:

$$x_k = \sum_{j \in L_k} \frac{x_j}{n_j} \tag{1}$$

donde n_j es el grado de salida de j (es decir, la cantidad de links que tiene la página j o, lo que es equivalente, los ejes salientes del nodo correspondiente a j).

En este modelo, se decide ignorar los links autoreferenciales (uno no puede subir su ranking votándose a si mismo), así como también considerar sólo la conexidad entre dos nodos en lugar de considerar a todos los distintos links que conectan en el mismo sentido el mismo par de páginas (es decir, no tenemos un multigrafo[b]).

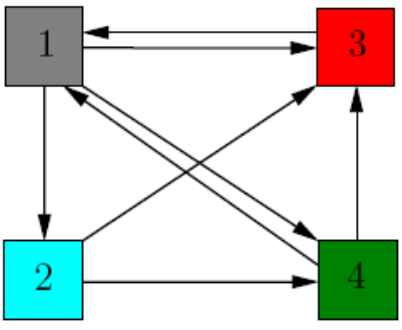


Figura 2. Grafo de Conectividad [**Bryan2006**]

Observemos en la figura 2 un ejemplo de un grafo de conectividad entre las páginas $1, 2, 3$ y 4:

Aplicando la ecuación 1 en este ejemplo, obtenemos que el siguiente sistema de ecuaciones:

$$\begin{cases} n_1 = 3 \\ n_2 = 2 \\ n_3 = 1 \\ n_4 = 2 \end{cases} \implies \begin{cases} x_1 = x_3/1 + x_4/2 \\ x_2 = x_1/3 \\ x_3 = x_1/3 + x_2/2 \\ x_4 = x_1/3 + x_2/2 \end{cases} \tag{2}$$

Y como ya se ha visto a lo largo de la materia, este sistema de ecuaciones es expresable como un sistema $Ax = x$ para $A \in \mathbf{R}^{4\times4}$ y $x \in \mathbf{R}^4$ (pues $\vec{x}$ es tanto el vector de coeficientes como el vector resultado):

a. Es decir, la cantidad de ejes entrantes al nodo.
b. Un grafo que permite tener varias aristas conectando los mismos nodos (en el mismo sentido, en el caso de un grafo dirigido).

$$\underbrace{\begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{x} \tag{3}$$

Esta matriz A obtenida, si se observa bien, es lo que podríamos llamar una matriz de conectividad pesada. Esto se nota al caracterizar el valor de cada coordenada de la matriz:

$$a_{ij} = \begin{cases} 0 & \text{si } j \text{ no tiene links a } i \\ 1/n_j & \text{si } j \text{ tiene links a } i \end{cases} \tag{4}$$

Entonces vemos que la matriz tiene valores no nulos siempre que haya un link de j a i, y en tal caso el valor de la matriz será algo similar a repartir el "único" voto del nodo j entre todas sus conexiones/ejes salientes (n_j)[a].

Partimos entonces de un modelo basado en digrafos[b] y una expresión matemática para el cálculo del puntaje de cada nodo, y llegamos al problema de resolver un sistema del tipo $Ax = x$, que es equivalente a resolver $Ax = \lambda x$ con $\lambda = 1$. Y esto último no es otra cosa que calcular el autovector asociado al autovalor 1 de la matriz A [**Burden2010**].

Claramente, antes de comenzar a resolver este problema hay que plantearse si el mismo tiene solución. En este caso, deberíamos poder asegurar la existencia de un autovalor 1. Por suerte para el modelo, se puede asegurar que existirá dicho autovalor si la matriz es *estocástica por columnas*[c] [**Bryan2006**], con lo cual para asegurar que existirá el x buscado podríamos tratar de ver si A es estocástica por columnas.

En [**Bryan2006**] se demuestra que la matriz A es estocástica por columnas si el grafo asociado está libre de *dangling nodes*, o nodos con grado de salida 0 (es decir, páginas web sin links). Matricialmente hablando, esto sería una columna de A con ceros únicamente. Así pues, encontramos una primera cosa que modelo debe solucionar para asegurar la existencia del autovalor 1.

Otro problema que surge es la posibilidad de que el autovalor 1 de esta matriz (suponiendo que se aseguró la ausencia de dangling nodes) tenga multiplicidad mayor a 1. Es decir, que tengamos múltiples soluciones para nuestro modelo (múltiples posibles rankings). Claramente esto es algo no deseado, ya que no tenemos un patrón para decidir que ranking es el correcto en caso de tener varios. Así pues, aquí encontramos otro inconveniente a solucionar: la existencia de rankings múltiples.

Dangling Nodes

Hasta ahora, hemos observado al modelo para puntuar a las páginas basados únicamente en los links entre las mismas, sin tomar en cuenta el comportamiento de los usuarios navegando en las páginas. PageRank, en su primera versión, propone que todo usuario siempre puede decidir ir a la barra de dirección e ir a cualquier página con la misma probabilidad. Obviamente, esto último no es del todo fino y puede ser mejorado, es mucho más probable que usuario escriba la dirección de un buscador web, o de alguna página relacionada con el mismo ámbito de la página actual en la que se encuentra [d]. Aún así, dado el contexto didáctico de este trabajo, utilizaremos esta versión en la cual se asume que el navegador elige con la misma probabilidad cualquiera de las páginas a las cuales "saltar" mediante la barra de direcciones (tal como lo hicieron Bryan y Page en 1998). A este proceso de "saltar" a cualquier otra página/nodo del grafo se lo denomina *teletransportación*.

Volviendo a nuestra matriz A, se utiliza el concepto de teletransportación de manera consistente con el modelo, de modo de convertir a A en una matriz sin dangling nodes y que represente a un grafo fuertemente conexo[e],

a. Decimos "único voto" pues en A no están los rankings de las páginas (los x_k). Esto se puede entender como que todas las páginas tienen el mismo poder de voto (igual a 1) pero lo reparte de manera equitativa entre todas las páginas a las cuales se víncula.

b. Grafos dirigidos.

c. Se dice que una matriz es estocástica por columnas si todos sus valores son no negativos y la suma de los valores de cualquier columna es igual a 1.

d. Cosas que *Google* seguramente comenzó a tomar en cuenta en siguientes versiones de su buscador.

e. Cosa que en la realidad no necesariamente pasa. Como ejemplo, alcanza con que exista una página web que no sea referenciada ni que referencie a otros sitios web. O algún tipo de documento web, como un *pdf*

o equivalentemente, convertirla a una matriz de Markov[a] irreducible. Luego la teoría de cadenas de Markov nos asegurará, entre otras cosas, la existencia del autovalor 1 (pues una matriz de Markov es estocástica por columnas por definición, y ya hemos comentado hace algunos párrafos que en tal caso existe el autovalor 1) y la unicidad del autovector asociado (nuestro $\vec{x}$).

La idea intutiva planteada por Brian y Page fue la del **navegante aleatorio**. Básicamente plantean que existe un usuario (o navegante) que va "viajando" entre los links de la estructura del grafo/web. Es decir, cuando llega a una página con muchos links, elige uno aleatoriamente y llega a una nueva página. Este proceso sigue indefinidamente. Pensando en el infinito, la proporción del tiempo que este navegante pasa en una página dada sería la medida de la importancia relativa de dicho sitio (respecto de las demás páginas). A mayor proporción, mayor importancia, por lo tanto, debería tener mayor puntaje. Entonces las cuestiones de los dangling nodes aquí se notan como un problema evidente, ya que una vez que llega a uno de ellos, el navegante no puede continuar. Aquí se utilizó la teletransportación para solucionar este inconveniente.

Se realiza entonces un **ajuste estocástico** sobre A, donde todas las columnas $\vec{0}$ son reemplazadas por columnas $(1/n)e$, para $e = \vec{1}$ y n la cantidad de nodos/páginas. De esta manera la matriz A se vuelve estocástica por columnas, ya que las columnas correspondientes a los dangling nodes ahora tienen n valores de $1/n$, y su sumatoria es 1 trivialmente. Así pues, se resolvió el inconveniente de los dangling nodes y se convirtió a A en una matriz de estocástica, y ya se mencionó que en este caso y si la misma estaba libre de dangling nodes, tenemos asegurada la existencia del autovalor 1.

Matricialmente hablando, tendríamos que:

$$S = A + \left(\frac{1}{n}e\right) a^T \qquad \text{Donde: } a_i = \begin{cases} 1 & \text{si el nodo } i \text{ es un dangling node} \\ 0 & \text{caso contrario} \end{cases} \tag{5}$$

Tomemos un ejemplo para que se vea precisamente como esto elimina los dangling nodes de un grafo. En la figura 3 tenemos el mismo ejemplo anterior, pero habiendo eliminado el link de 3 a 1, convirtiéndo el nodo 3 en un dangling node.

$$S = \overbrace{\begin{bmatrix} 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}}^{A} + \overbrace{\begin{bmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{bmatrix}}^{(1/4)e} \overbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}}^{a^T} = \begin{bmatrix} 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & \frac{1}{4} & \frac{1}{2} \\ \frac{1}{3} & 0 & \frac{1}{4} & 0 \\ \frac{1}{3} & \frac{1}{2} & \frac{1}{4} & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & \frac{1}{4} & 0 \end{bmatrix} \implies S \text{ es matriz de Markov}$$

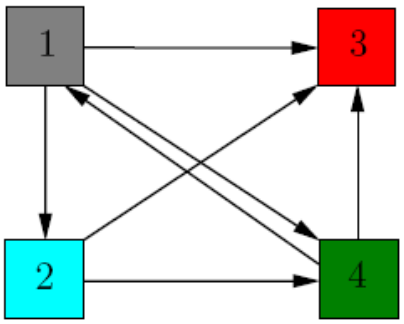


Figura 3. Dangling Nodes

Rankings Múltiples

Ahora bien, como se ha mencionado es deseable tener un único autovector x asociado al autovalor 1. La teoría nos indica que en caso de tener una matriz de Markov, que a su vez sea cuadrada e irreducible, entonces el vector estacionario[b] de la cadena de Markov asociada a esta matriz[c] existe y es único [**Meyer2000**][d] (y puede ser calculado con el método de la potencia, tema que desarrollaremos más adelante en este trabajo). Y justamente ese vector estacionario es el autovector asociado al autovalor 1 de la matriz que se construirá.

a. Una matriz de Markov es una matriz positiva y estocástica por columnas.
b. También llamado en la teoría **Vector de Perron**.
c. Matriz a la que se denomina *matriz de transición de probabilidades*
d. Tener en cuenta que en la referencia indicada, la matriz utilizada es estócastica por filas. Es decir, en este desarrollo estamos trabajando con la matriz de Markov transpuesta. Si llamamos a nuestra matriz de Markov M, la equivalencia con la matriz P de [**Meyer2000**] es $M^T = P$. Luego es trivial ver que el vector de Perron transpuesto es nuestro autovector x, ya que: $\pi^T P = \pi^T \iff \pi^T M^T = \pi^T \iff M\pi = \pi \implies x = \pi$

Para poder llegar a esta matriz de Markov de manera consistente con lo que se está modelando, Bryan y Page usan nuevamente el argumento del navegante aleatorio y la teletransportación. Básicamente sostienen que en realidad, dicho navegante, siempre puede ir a la barra de direcciones y saltar a otra página ("linkeada" o no en la página actual), esté o no en un página sin links (dangling node). Luego, el proceso de navegación se sigue repitiendo, es decir que el usuario sigue visitando los links, pero siempre con una probabilidad de ir a la barra de direcciones y teletransportarse a cualquier otra página.

Para modelar esto matemáticamente, se definió una nueva matriz M a partir de S de la siguiente manera:

$$M = \alpha S + (1-\alpha)\frac{1}{n}ee^T \qquad \alpha \in \mathbf{R} \wedge \alpha \in [0,1) \tag{6}$$

En este modelo, α es un parámetro que controla la proporción del tiempo (o probabilidad, depende como se lo quiera mirar) en la cual el navegante aleatorio sigue viajando a través de los links de las páginas (S) en contraposición a teletransportarse ($E = \frac{1}{n}ee^T$[a]). Básicamente, α nos determina que matriz de navegación tendrá más peso en M. A su vez, cada vez que el navegante toma una decisión, las probabilidades de ésta quedan modeladas en la matriz M, y son siempre las mismas para cada viaje del navegante (en este modelo no se asume nada sobre la dirección de un viaje en cuanto al viaje anterior, es decir que se representa un proceso estocástico sin memoria). Así pues, observamos como todos estos viajes son bien representados por la cadena de Markov de nuestra matriz M.

Veamos entonces que esta matriz es una matriz de Markov irreducible y cuadrada:

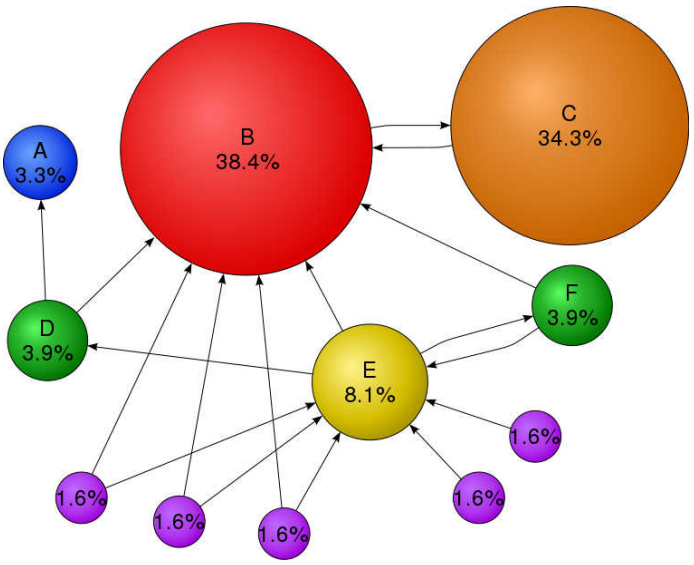


Figura 4. Grafo de conectividad con tamaño de nodos en función de PageRank [**wiki˙pagerank**]

- M es *estocástica por columnas*, pues es la combinación convexa de dos matrices estocásticas por columnas (S y E[b]).
- M es *positiva*. Observar que al definir M, $e_{ij} > 0 \quad \forall i,j$, luego como $\alpha < 1$ podemos asegurar que E será positiva. A su vez S es no negativa, por lo cual la suma $S + E$ seguro que será una matriz positiva (es decir, $m_{ij} > 0 \quad \forall i,j$).
- M entonces es una matriz de **Markov**, ya que su definición es la combinación de los dos items anteriores.
- M es *cuadrada*. Además de M haber sido construída a partir de A (que es cuadrada), M representa los posibles "saltos" entre 2 páginas web (o nodos de un grafo). Es decir, el conjunto de nodos de partida (filas) y de llegada (columnas) de un salto debe ser el mismo (en este modelo), por lo tanto $M \in \mathbf{R}^{n\times n}$.
- M es *irreducible*. El grafo resultante de interpretar a M como una matriz de conectividad (habrá un eje entre los nodos i y j si $p_{ij} \neq 0$) es completo pues M es positiva. Luego el grafo asociado a M es fuertemente conexo. A su vez M es una matriz cuadrada. Luego, con estas 2 condiciones se puede asegurar que M es irreducible [**Meyer2000**].

Es decir, se han dado todas las condiciones que nos permiten asegurar que que el autovector asociado al autovalor 1 existe y es único [**Meyer2000**]. Y dicho autovector será la solución a nuestro sistema, tal que cada componente x_j de $\vec{x}$ será el puntaje del nodo/página j. La forma de calcular este autovector será parte de este trabajo y será explicada más adelante.

1.3. Adaptación a Eventos Deportivos

En los eventos deportivos los rankings son claramente importantes. Uno puede observar a un torneo como un ranking que se va generando a lo largo del tiempo (a medida que progresa la competición) y que al finalizar el torneo determina quién es el ganador.

a. Observar que la matriz E es una matriz tal que $\forall i,j \quad e_{ij} = {}^1/_n$
b. Se puede observar trivialmente que E es estocástica por columnas.

Se observa entonces que cada competición, deporte, torneo y/o asociación tiene su propio sistema de rankings: el fútbol suele manejar un sistema de puntos 3-1-0 por partido Ganado-Empatado-Perdido, el basquet cotabiliza partidos ganados y perdidos (no hay empates), el tenis (a nivel ranking de jugadores, no de torneo) mantiene un sistema de puntos por nivel alcanzado en cada torneo anual.

Ahora bien, este es uno de los aspectos de los deportes. El otro aspecto (más allá del deporte en sí mismo) radica en cómo se transita la competición, qué partidos se dan y qué competidores se enfrentan. Por ejemplo, en el esquema clásico del torneo de fútbol se confecciona un calendario de partidos o *fixture* donde todos los equipos juegan contra todos los demás 2 partidos a lo largo de la competición. Sin embargo, en algunas ligas de basquet, el calendario de partidos propone dividir a los equipos en divisiones (agrupadas en 2 conferencias) según su ubicación geográfica donde cada equipo juega más partidos contra los equipos de su división que contra equipos de su conferencia, con los cuales juegan más veces que contra los equipos de la conferencia restante.

En la figura 5 se puede observar un ejémplo gráfico de esta asimetría. Se observa un grafo donde cada nodo representa uno de los 130 equipos de la liga universitaria de fútbol americano, los cuales juegan un total de 12 partidos cada uno por temporada. La gran mayoría de esos partidos se dan contra equipos de su misma emphpower conference, o división, en las cuales se agrupan todos las universidades. Esto se ve reflejado en el grafo ya que se observan 6 subgrafos muy densos (los cuales agrupan a los equipos de una misma división), y luego se ven muchas menos aristas que unen a equipos de distintas divisiones.

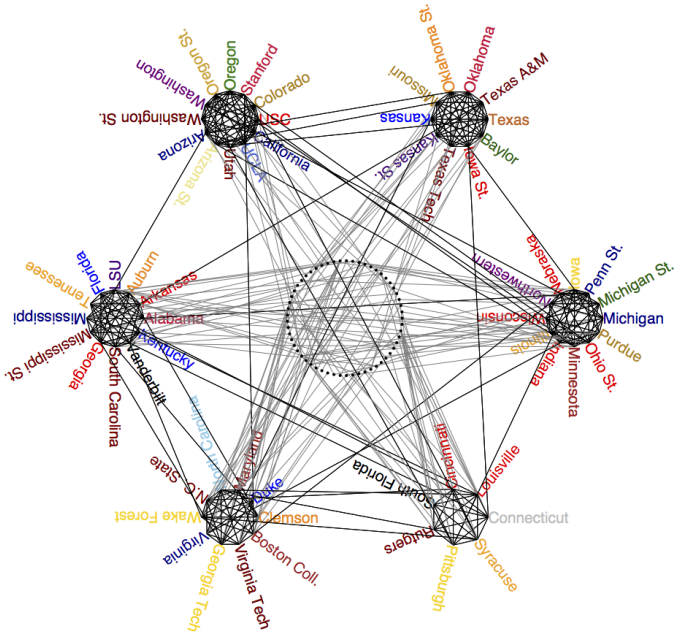


Figura 5. Grafo de Encuentros de la liga de fútbol americano universitaria [**college˙football˙graph**]

Estas diferencias se dan más y más mientras más competiciones deportivas se estudien, sumado a que cada cual tiene su propio ranking el cual supone tener en cuenta todas estas características de la competicón. Claramente, la pregunta a hacerse es si dichos rankings son realmente un reflejo de la realidad. Más aún, existen rankings confeccionados que establecen una relación de orden entre equipos del mismo deporte pero de distintas asociaciones. Por ejemplo, el ranking de clubes profesionales de fútbol. El mismo establece que el Barcelona de España[a] está muy por encima que el club Platense de Argentina[b]. ¿Y cómo se llega a esta conclusión si ambos equipos jamás jugaron entre sí? ¿Es dicho ranking correcto?

Incluso se pueden tener rankings no basados necesariamente en los resultados de la competencia, sino en la opinión de los periodistas, entrenadores, seguidores o los mismos competidores. Es decir, dada una competencia, pueden existir múltiples sistemas de ranking más allá del asociado a la competición (aunque este es el único, en principio, con injerencia en la determinación del ganador).

La elección de qué ranking usar es problemática debido a todos los intereses distintos que dependen del mismo (en el caso de los deportes profesionales). Y como se ha explicado, utilizar un ranking basado únicamente en partidos perdidos y ganados no alcanza ya que en muchas competiciones directamente hay equipos que no juegan contra otros (y en otros casos hay asimetrías en cuanto a la cantidad de partidos jugados contra distintos partidos). Además, se debe tener en cuenta que en ningún deporte las victorias son "transitivas": que A le haya ganado a B y B a C no implica necesariamente que A vaya a ganarle a C.

En este contexto, Govan, Meyer y Albright propusieron utilizar una adaptación del algoritmo de PageRank para generar rankings basados en los resultados de la competición [**Govan2008**].

Extrapolando la idea de PageRank, donde ser referenciado por una página incrementa el puntaje (o ranking) de la página referenciada de acuerdo al puntaje de quien nos referencia, los autores proponene que ganarle a un competidor bien "rankeado" vale mucho más que ganarle a alguien no tan bien ubicado en el ranking. Así pues, ya

a. http://www.fcbarcelona.com/
b. http://www.cap.org.ar/

vemos una primer diferencia. Ganar más partidos no asegura estar por encima de alguien que gano menos, sino que la calidad de los rivales influye también (esto contrasta con la mayoría de los rankings deportivos tradicionales).

La adaptación sugiere utilizar PageRank, pero en lugar de basarnos en una matriz de conectividad donde los pesos de los ejes dependen del puntaje y cantidad de ejes del nodo de partida, sugiere que los pesos se calculen en base al resultado final del deporte.

Así pues, la aplicación del método *GeM* consiste en cambiar un poco los datos de las matrices del algoritmo de PageRank ya explicado en la sección 1.2 teniendo en cuenta los resultados. El primer problema que surge es qué hacer con los empates, pues el grafo de PageRank y su correspondiente matriz de connectividad no modelan un vínculo no dirigido. Govan, Meyer y Albright realizaron su propuesta en base a deportes de los Estados Unidos, donde los empates directamente no pueden ocurrir, o la probabilidad de que ocurran es extremadamente baja. Así pues, utilizan el *algoritmo del avestruz*: ignoran el problema y no lo modelan. Entonces, utilizando la misma nomenclatura que en la sección anterior, las matrices de GeM quedan definidas como:

$$A \in \mathbf{R}^{n\times n} / a_{ij} = \begin{cases} w_{ij} & \text{Si } i \text{ ganó contra a } j \\ 0 & \text{Caso contrario} \end{cases} \quad \text{donde } w_{ij} \text{ es la diferencia del marcador} \tag{7}$$

$$H \in \mathbf{R}^{n\times n} / h_{ij} = \begin{cases} a_{ij} / \sum_{k=1}^{n} a_{kj} & \text{Si } i \text{ ganó contra } j \\ 0 & \text{Caso contrario} \end{cases} \tag{8}$$

$$S \in \mathbf{R}^{n\times n} / S = H + ua^T \qquad \text{Donde: } a_i = \begin{cases} 1 & \text{si } i \text{ no perdió ningún partido} \\ 0 & \text{caso contrario} \end{cases} \tag{9}$$

$$\alpha \in \mathbf{R}, M \in \mathbf{R}^{n\times n} / M = \alpha S + (1-\alpha) v e^T \tag{10}$$

En caso de enfrentarse más de 1 vez dos competidores i y j y que un mismo competidor haya ganado más de 1 encuentro (sin pérdida de generalidad, diremos que i le ganó a j más de una vez), entonces w_{ij} será la diferencia del marcador acumulada entre todos los encuentros.

Volviendo a las ecuaciones, observamos que son muy similares a las vistas en anteriormente. Las diferencias aparecen en la definición de H, y en la aparición de los vectores u y v. Estos vectores son ambos vectores de distribución de probabilidad, donde $\forall i, v_i = {}^1/_n$ y u será un vector de ajuste para equipos invictos análogo al vector utilizado en el caso de PageRank para el caso de los dangling nodes. Desarrollando un poco, cada u_i sería la probabilidad de que un equipo invicto pierda contra el equipo i. En una primera aproximación (la cual es la desarrollada por este trabajo), asumimos que $u = {}^{\vec{1}}/_n$, con lo cual podemos establecer las siguientes igualdades respecto de las ecuaciones de PageRank: $u = v = \frac{1}{n} e$, quedando:

$$S = H + \left(\frac{1}{n}\right) e a^T \tag{11}$$

$$M = \alpha S + (1-\alpha) \left(\frac{1}{n}\right) e^T \tag{12}$$

Es decir, llegamos a exactamente la misma expresión que en el caso de PageRank, con la única diferencia de que S depende de H en lugar de la matriz de conectividad. Y H no es otra cosa que la matriz de conectividad (o adjacencia) donde cada adjacencia está pesada por la relación *puntaje en contra de i vs j / puntaje en contra total de i*, (siendo los puntajes i y j el resultado del encuentro entre ellos[a]) en lugar de $1/n_j$.

Es trivial[b] ver que esta nueva matriz es cuadrada e irreducible (su grafo asociado es nuevamente un grafo completo, y por lo tanto fuertemente conexo **[Meyer2000]**), que es positiva ($\alpha \in [0, 1)$), y estocástica por columnas (por los vectores de ajuste u y v de la definición de M). Luego, dadas todas estas condiciones, podemos asegurar la existencia y unicidad del vector de Perron transpuesto π de la cadena de irreducible de Markov de la matriz M (el cual satisface $M\pi = \pi$ **[Meyer2000]**), siendo este el vector que nos da el ranking de equipos al igual que nos daba el ranking de sitios web en el caso de PageRank.

Así pues, se observa que el modelo GeM mantiene la mismas propiedades que el modelo propuesto por PageRank, con lo cual aseguramos la existencia y unicidad de su solución y puede ser resuelto con los mismos métodos con

a. No quisismos decir *goles*, pues el término será muy dependiende de la competencia.
b. Luego de haber ya explicado los conceptos en la sección 1.2.

los cuales se resuelve el modelo PageRank (uno de los cuales será desarrollado más adelante). Sin embargo, que el modelo sea consistente y resoluble no nos asegura que sus resultados sean útiles, y sobre este aspecto hablaremos en la sección 4.

2. DESARROLLO

EN la sección anterior se presento una descripción breve del problema a resolver, y se llegó a un modelo matricial del mismo sobre el cual la resolución del problema se reducía a calcular el autovector x asociado al autovalor λ de valor 1. A su vez, se demostró que como consecuencia del modelo y su matriz resultante, este x y λ existirán (y en el caso de x, que será único).

A continuación presentamos un método numérico iterativo el cual fue utilizado para resolver este problema. Explicaremos brevemente el método en sí mismo, sus posibles desventajas y porque el mismo funcionará en este contexto.

2.1. Método de la Potencia

El método de la potencia es una técnica iterativa usada para calcular el autovalor dominante de una matriz (es decir, el autovalor de mayor valor absoluto). A su vez este método nos permite calcular no sólo el autovalor dominante, sino que también un autovector asociado (que es de hecho, el uso que es relevante para este trabajo).

Para utilizar este método se asume que se tiene una matriz cuadrada de dimensión n que tiene n autovectores *li*[a] Y, evidentemente por lo ya comentado en el párrafo anterior, se asume que la matriz tiene un único autovalor dominante:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n| \geq 0$$

Estos requisitos son los necesarios para asegurar que el método de la potencia será satisfactorio. En el caso de no tener n autovectores li el método aún podría terminar satisfactoriamente, pero no se podría garantizar este resultado [**Burden2010**].

El método en cuestión, en su forma de pseudocódigo, se encuentra en el algoritmo 1:

Algoritmo 1: Método de la potencia - Pseudocódigo [**Burden2010**]

Entrada: Dimensión n; matriz M; vector x; tolerancia ϵ; máximo número de iteraciones N
Salida: Autovalor aproximado μ; autovector aproximado x (con $||x||_\infty = 1$)

$k \leftarrow 1$;
Sea p el menor entero tal que $1 \leq p \leq n$ y $|x_p| = ||x||_\infty$;
$x \leftarrow x/x_p$;
mientras $k \leq N$ **hacer**
  $y \leftarrow Mx$;
  Sea p el menor entero tal que $1 \leq p \leq n$ y $|y_p| = ||y||_\infty$;
  $\mu \leftarrow y_p$;
  si $y_p = 0$ **entonces**
    devolver M *tiene un autovalor* 0, *seleccione un nuevo vector* x *y cominece de nuevo*;
    /* El proceso finalizó insatisfactoriamente */
  fin
  $\delta \leftarrow ||x - (y/y_p)||_\infty$;
  $x \leftarrow y/y_p$;
  si $\delta < \epsilon$ **entonces**
    devolver μ, x;
    /* El método finalizó satisfactoriamente */
  fin
  $k \leftarrow k + 1$;
fin
devolver *Se alcanzó el máximo número de iteraciones*;
/* El proceso finalizó insatisfactoriamente */

Ahora bien, el método expuesto como se lo ve en el algoritmo 1 es genérico, y no toma en cuenta las características propias de nuestra matriz M de entrada.

a. Linealmente independientes

En primer lugar, para que este método sea útil a nuestro problema, deberíamos poder asegurar que nuestro autovalor $\lambda = 1$ es el **único** autovalor dominante de la matriz M. Esto efectivamente ocurre, como demostramos a continuación:

- M es estócastica por columnas, positiva y cuadrada (sección 1.2) $\implies$ M^T es estócastica por filas, positiva y cuadrada (trivial).
- $det(M) = det(M^T) \implies det(M - \lambda I) = det(M^T - \lambda I) \implies$ M y M^T tienen los mismos autovalores.
- $\nexists \lambda \in L(M^T) / |\lambda| > 1$ Para $L(M^T)$ el conjunto de autovalores de M^T. Demostración:

  Sea λ autovalor de M^T y v su autovector asociado, tal que $|\lambda| > 1$.

$$\iff M^T v = \lambda v \implies |M^T v| = |\lambda v| \overset{M^T > 0}{\implies} M^T|v| = |\lambda||v| \tag{13}$$

  Sea $m = M^T|v|$). Luego, podemos asegurar que cada componente del vector m será menor a v_{max}, para $v_{max} = max|v_i|, \forall i$

$$m_i = \sum_{j=1}^{n} M_{ij}^T |v_j| \overset{v_{max} \geq |v_i|}{\leq} \sum_{j=1}^{n} M_{ij}^T v_{max} = v_{max} \sum_{j=1}^{n} M_{ij}^T \overset{\text{Estócastica}}{=} v_{max} \tag{14}$$

$$\implies \forall i, \quad m_i \leq v_{max} \tag{15}$$

  Sin pérdida de generalidad, digamos que j es el índice tal que $|v_j| = v_{max}$. Luego:

$$m_j = |\lambda||v_j| = |\lambda| v_{max} \overset{|\lambda| > 1}{>} v_{max} \tag{16}$$

$$\implies m_j > v_{max} \tag{17}$$

  Pero entonces llegamos a un absurdo:

$$\left.\begin{array}{r} \text{De 15: } \forall i, \quad m_i \leq v_{max} \implies m_j \leq v_{max} \\ \text{De 17: } m_j > v_{max} \end{array}\right\} \implies \text{Absurdo.} \tag{18}$$

  Así pues, llegamos a un absurdo por haber asumido que $|\lambda| > 1$, ergo esta suposición ha de ser falsa y tenemos que $|\lambda| \leq 1$.

- Recién vimos que $\forall \lambda \in L(M^T), |\lambda| \leq 1$. Como M y M^T tienen los mismos autovalores, entonces $\forall \lambda \in L(M), \lambda \leq |1|$
- Como 1 es autovalor de M (sección 1.2) y sabemos que ningún autovalor de M puede tener magnitud mayor a 1 (ecuación 18) $\implies$ 1 es un autovalor dominante de M. Como corolario, tendríamos que $\rho(M) = 1$.
- Por último, para demostrar la unicidad de este autovalor dominante 1, alcanza con remarcar que la multiplicidad de dicho autovalor ya se ha mostrado que es 1 (sección 1.2). Pero sino, también podemos enunciar que M es positiva y cuadrada $\implies$ $\rho(M)$ es el único autovalor de M en el radio espectral [**Meyer2000**], con lo cual 1 es entonces **el único autovalor dominante** de M.

Así pues ya sabemos cual es el valor del autovalor dominante M: 1. Por lo tanto, el cálculo de este valor no nos hace falta. Más aún, como sabemos que M es estocástica y nuestro vector inicial x (u $x^{(0)}$) será un vector de probabilidad (es decir, estocástico), entonces podemos asegurar que $Mx^{(0)}$ será estocástico[a]. Así pues, muchos de los pasos del método expuesto en el algoritmo 1 que tienen como objetivo normalizar vectores, son innecesarios, pues $Mx^{(k)}$ será siempre estocástico (al menos en la teoría, ya que después por cuestiones de los cálculos númericos

a. Trivial de demostrar.

con aritmética finita en la computadora puede llegar a obligarnos a tener que normalizar los vectores luego de cada cálculo).

Justamente esta es una de las conclusiones de Bryan y Leise al decirnos como computar el autovector de PageRank (aunque en ningún momento mencionan cadenas de Markov, o método de la potencia), y además concluyen que el método convergirá para cualquier sea $x^{(0)}$ que sea un vector de probabilidad [**Bryan2006**]. En su teorema número 5, nos indican que nuestro autovector x puede ser calculado como:

$$x^{(k)} = \alpha S x^{(k-1)} + (1-\alpha)\frac{1}{n}ee^T = Mx^{(k-1)}$$
$$x = \lim_{k\to\infty} x^{(k)}$$

Claramente aquí cuando se toma un *límite* para calcular x, no es algo realmente computable. Justamente es por eso que el algoritmo 1 implementa un criterio de parada basado en la distancia del vector entre iteraciones, y a su vez tiene un número máximo de iteraciones para llegar a un resultado lo suficientemente aproximado.

Esta fórmula no es otra cosa que el método de la potencia, y justamente lo más interesante a extraer del artículo de Bryan y Leise (en lo que respecta al método de la potencia aplicado a nuestro problema) es que alcanza conque el vector inicial $x^{(0)}$ sea un vector de probabilidad. Luego, el método convergirá al x que buscamos (pues ya hemos demostrado que nuestra matriz M tiene un único autovector dominante).

De esta manera llegamos a un pseudocódigo del método de la potencia más orientado a nuestro problema, e incluso más claro quizás. El mismo se puede observar en el algoritmo 2.

Algoritmo 2: Método de la Potencia en el contexto de PageRank y su matriz M - Pseudocódigo [**Kamvar2003**]

Entrada: Matriz M; vector inicial $x^{(0)}$; número máximo de iteraciones N; tolerancia ϵ
Salida: Autovector aproximado x

```
1  v ← x^(0);
2  k ← 1;
3  repetir
4  |  x^(k) = M x^(k-1);
5  |  δ ← ||x^(k) − x^(k-1)||_1;
6  |  k ← k + 1;
7  hasta que δ < ε ∧ k ≤ N;
8  si k > N entonces
9  |  devolver Cantidad máxima de iteraciones alcanzada /* Finalización insatisfactoria */
10 fin
11 devolver x^(k) /* Finalización satisfactoria */
```

Como comentario, se puede notar que en el algoritmo 2 se utiliza para el criterio de corte la norma 1 (o distancia *Manhattan*). Esto es así pues Kamvar, Haveliwala et al. presentaron resultados empíricos que les permitieron demostrar que dicha medida era adecuada [**Kamvar2003**].

Así pues hemos llegado a un simple método númerico, el cual calcula un vector aproximado al autovector asociado al autovalor dominante de la matriz M del modelo explicado en la sección 1.2. A su vez se demostró que dicho autovalor dominante no es otro que 1, con lo cual el autovector $\tilde{x}$ calculado cumplirá con $M\tilde{x} \approx \tilde{x}$. Se probó que dicho método servirá (es decir, en la teoría su resultado convergirá **exactamente** al autovector buscado), pero al tratarse de un método iterativo no queda otra que conformarnos con una implementación que aproxime al resultado buscado, pues el no podemos iterar hasta el infinito[a]. Aún así, como lo que nos interesa en el contexto de este trabajo es obtener el orden que nos da el autovector, más que los valores de cada componente del mismo en cuestión, obtener una versión aproximada debería ser suficiente [b].

a. Es razonable esperar un resultado en tiempo finito.
b. De hecho obtener la versión exacta es imposible, ya que la computadora trabaja con aritmética finita.

3. IMPLEMENTACIÓN

A continuación nos explayaremos sobre los detalles de la implementación del método propuesto. En las secciones previas hemos dado una introducción al problema, su modelo y justificación de por qué el mismo sirve y a su vez hemos expuesto un método que nos permite hayar la solución.

En esta sección primero hablaremos sobre una implementación del algoritmo 2 (página 12), su justificación y correctitud. Luego pasaremos a resolver el problema de las estructuras de datos utilizadas para las matrices del problema (las cuales, por representar grafos completos, presentarán incovenientes al querer trabajar con *datasets* cada vez más grandes).

3.1. Implementación del Método de la Potencia

En la sección previa ya se han presentado dos pseudocódigos (algoritmo 1 y 2) correspondientes al método que utilizamos durante este trabajo. El principal problema es que ambas opciones (una siendo una versión simplificada de la otra, para nuestro contexto) es que computan $x^{(k)} \leftarrow Mx^{(k-1)}$. Recordemos la definición de nuestra matriz M entonces, de la ecuación 12:

$$A \in \mathbf{R}^{n\times n}/a_{ij} = \begin{cases} 0 & \text{si } j \text{ no tiene links a } i \\ 1/n_j & \text{si } j \text{ tiene links a } i \end{cases} \tag{19}$$

$$S = A + \underbrace{\frac{1}{n}e}_{\vec{v}}\ a^T \qquad \text{Donde: } a_i = \begin{cases} 1 & \text{si el nodo } i \text{ es un dangling node} \\ 0 & \text{caso contrario} \end{cases} \tag{20}$$

$$M = \alpha S + (1-\alpha)\ \underbrace{\overbrace{\frac{1}{n}e}^{\vec{v}}\ e^T}_{E} \tag{21}$$

El problema, entonces, es que al computar en cada iteración k del método de la potencia se efectúa la múltiplicación Mx^{k-1}, y M es una matriz positiva (como ya se a explicado, o como se puede deducir de las definiciones de las matrices recién expuestas). Entonces tenemos que:

- Cada iteración realizará $2n^2 - n$, es decir, $\mathcal{O}(n^2)$ operaciones matemáticas. Cada fila de M multiplca a $\vec{x}^{(k-1)}$, realizando n multiplicaciones, para luego sumar todos sus resultados ($n-1$ sumas) y obtener una coordenada de $\vec{x}^{(k)}$, y esto ocurre para n filas.
- Cuando comencemos a trabajar con muchas páginas web, el tamaño de la matriz crecerá mucho, y la complejidad espacial de la implementación de la matriz se volverá un problema de implementación (ni hablar cuando se trata de *rankear*[a] a todas las páginas de internet).

Kamvar et al. plantean que las aristas *artificiales* introducidas en la matriz A (que representa el grafo de conectividad pesado según el grado de salida de cada nodo) mediante $\vec{v}a^T$ y E no necesitan ser materializados en la matriz para realizar los cálculos (ya que de antemano sabemos como afectan a la matriz final M para cualquier operación que hagamos con ella) [**Kamvar2003**]. Por lo tanto, sugieren que el cómputo de $x^{(k)} \leftarrow Mx^{(k-1)}$ sea realizado con el siguiente algoritmo:

Algoritmo 3: Cómputo Eficiente de $x^{(k)}$[**Kamvar2003**]

Entrada: vector $\vec{x}^{(k-1)}$; matriz de conectividad pesada A; factor de navegación α
Salida: vector $\vec{x^{(k)}}$

1 $\vec{y} \leftarrow \alpha A\vec{x}^{(k-1)}$;
2 $w \leftarrow ||\vec{x}^{(k-1)}||_1 - ||\vec{y}||_1$;
3 $\vec{x}^{(k)} \leftarrow \vec{y} + w\vec{v}$;
4 **devolver** $\vec{x}^{(k)}$

Más adelante demostraremos que dicho algoritmo es correcto y efectivamente está computando $Mx^{(k-1)}$. Pero supongamos que asumimos esto como cierto, ¿qué es lo que se gana al implementar de esta manera $Mx^{(k-1)}$?

a. Palabra coloquial proveniente del inglés.

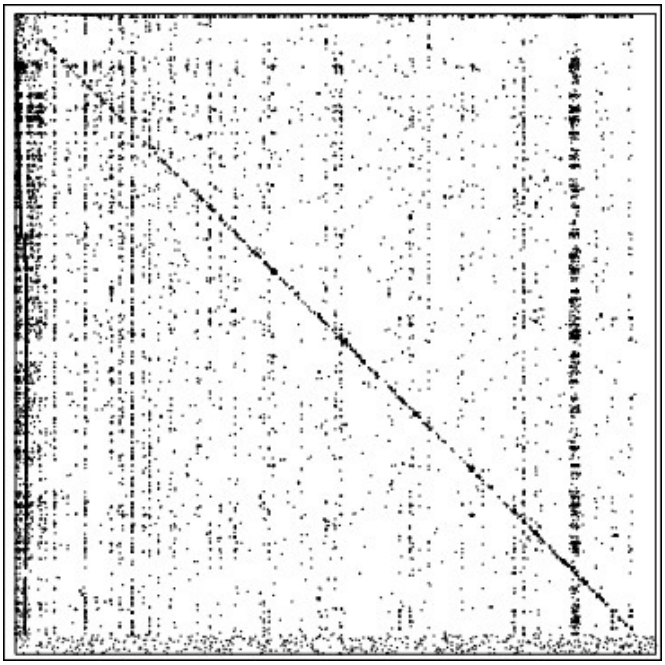


Figura 6. Ejemplo de Matriz Esparsa. Los valores no nulos están indicados en negro **[Langville2006]**

En primer lugar, observemos que nunca se utiliza la matriz M, sino que se usa A. Recordemos de la definición de A que la misma es una matriz de conectividad tal que $a_{ij} = {}^1/_{n_j}$ si hay un link de j a 1 y en caso contrario sera nulo. Lo interesante a retener, a partir de la definición de A y algo de conocimiento del dominio del problema, es que para casos muy grandes la matriz A comienza a ser cada vez más esparsa. Esto ocurre porque cada página tiene una cantidad muy acotada de links respecto de la cantidad de páginas web de internet.

En la figura 6 se puede observar un ejemplo real de una matriz A, donde todos los valores son nulos excepto por los pixeles en negro.

Así pues, llegamos a computar nuestro $Mx^{(k-1)}$ con una matriz esparsa. Esto lleva a solucionarnos los 2 problemas planteados de este cómputo. Por un lado las matrices esparsas pueden ser almacenadas utilizando muy poco espacio en memoria (mediante el uso de estructuras de datos que solo almacenan los valores no nulos más alguna otra información, las cuales veremos un poco más adelante). Por el otro, la multiplicación matriz-vector con una matriz esparsa requiere una complejidad mucho menor que $\mathcal{O}(n^2)$ operaciones matemáticas. De hecho, requiere $\mathcal{O}(nnz(A))$, para $nnz(A)$ la cantidad de valores no nulos de A. Valores estadísticos muestran que la cantidad promedio de links de una página web es alrededor de 10, lo que señalaría que $nnz(A) = 10n$ en lugar de los n^2 de la matriz M **[Langville2006]**. Entonces pasamos de tener una complejidad $\mathcal{O}(n^2)$ a $\mathcal{O}(n)$ (por supuesto, la implementación deberá permitir acceder y conocer la posición de los elementos no nulos de manera eficiente).

Claramente, entonces, tendríamos los dos problemas planteados resueltos en caso de ser equivalentes el cómputo de $x^{(k)} \leftarrow Mx^{(k-1)}$ y el algoritmo 3 propuesto. Luego sólo quedaría decidir qué estructura(s) de matrices esparsas utilizar.

3.2. Correctitud de Cómputo de $\mathbf{x^{(k)} \leftarrow Mx^{(k-1)}}$

Del algoritmo 3, reemplazando sus 3 pasos principales el uno en el otro, llegamos a que el $x^{(k)}$ devuelto será:

$$\vec{x}^{(k)} = \alpha A\vec{x}^{(k-1)} + (||\vec{x}^{(k-1)}||_1 - ||\alpha A\vec{x}^{(k-1)}||_1)\vec{v} \tag{22}$$

Dado que queremos demostrar que el algoritmo efectivamente computa $Mx^{(k-1)}$, lo que queremos demostrar es:

$$M\vec{x}^{(k-1)} = \alpha A\vec{x}^{(k-1)} + (||\vec{x}^{(k-1)}||_1 - ||\alpha A\vec{x}^{(k-1)}||_1)\vec{v} \tag{23}$$

Se puede observar en esto que queremos demostrar, ahora sólo tenemos $\vec{x}^{(k-1)}$, por lo tanto para facilitar la lectura nos referiremos al mismo como $\vec{x}$. Entonces:

$$\text{QVQ: } M\vec{x} = \alpha A\vec{x} + (||\vec{x}||_1 - ||\alpha A\vec{x}||_1)\vec{v} \tag{24}$$

De las ecuaciones 19, 20 y 21, podemos llegar a la siguiente definición de M:

$$\begin{aligned} M &= \alpha S + (1-\alpha)\vec{v}e^T \\ &= \alpha(A + \vec{v}a^T) + (1-\alpha)\vec{v}e^T \\ &= \alpha A + \alpha\vec{v}a^T + \vec{v}e^T - \alpha\vec{v}e^T \\ &= \alpha A + \vec{v}(e^T - \alpha(e^T - a^T)) \end{aligned} \tag{25}$$

Luego, multiplicamos por $\vec{x}$

$$M\vec{x} = \alpha A\vec{x} + \vec{v}(\underbrace{e^T\vec{x} - \alpha(e^T\vec{x} - a^T\vec{x})}_{\stackrel{?}{=}||\vec{x}||_1 - ||\alpha A\vec{x}||_1}) \tag{26}$$

Analicemos en primer lugar $e^T\vec{x}$ de esta última ecuación. Recordemos que $\vec{x}$ era un vector estócastico, entonces:

$$e^T\vec{x} = \sum_{i=1}^{n} x_i \overset{x_i \geq 0}{\downarrow} = \sum_{i=1}^{n} |x_i| = ||\vec{x}||_1 \tag{27}$$

Por otro lado, analicemos $||A\vec{x}||_1$

$$||A\vec{x}||_1 = \sum_{i=1}^{n} \left| \sum_{j=1}^{n} a_{ij}x_j \right| \overset{\substack{a_{ij} \geq 0 \\ x_j \geq 0}}{\downarrow} = \sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}||x_j| = \sum_{j=1}^{n} |x_j| \underbrace{\left(\sum_{i=1}^{n} |a_{ij}| \right)}_{= \begin{cases} 0 & \text{Si } n_j = 0 \\ 1 & \text{Si } n_j \neq 0 \end{cases}} \overset{x_i \geq 0}{\downarrow} = \sum_{\substack{j=1 \\ n_j \neq 0}}^{n} x_j \tag{28}$$

$\uparrow$ Columna j de A es estocástica si $n_j \neq 0$ — Ecuación 19

Y ahora analicemos el término restante de la ecuación 26:

$$e^T\vec{x} - a^T\vec{x} = \left(\sum_{i=1}^{n} x_i \right) - \left(\sum_{\substack{i=1 \\ n_i=0}}^{n} x_i \right) = \sum_{\substack{i=1 \\ n_i \neq 0}}^{n} x_i \tag{29}$$

Entonces, si observamos las ecuaciones 28 y 29, llegamos a que:

$$e^T\vec{x} - a^T\vec{x} = ||A\vec{x}||_1 \tag{30}$$

Por último, si tomamos las igualdades obtenidas en las ecuaciones 27 y 30 y las utilizamos en la ecuación 26, llegamos a que:

$$M\vec{x} = \alpha A\vec{x} + \vec{v}(||\vec{x}||_1 - \alpha||A\vec{x}||_1) \overset{\alpha \geq 0}{\downarrow} \implies M\vec{x} = \alpha A\vec{x} + (||\vec{x}||_1 - ||\alpha A\vec{x}||_1)\vec{v} \tag{31}$$

Que es, justamente, lo que queríamos demostrar (ecuación 24).

3.3. Estructuras de Datos para Matrices Esparsas[a]

Como se ha mencionado, la princpal "optimización" de la implementación[b] es la utilización de estructuras especiales para representar matrices esparsas. Esta elección, sin embargo, debe ser inteligente para permitir que el algoritmo 3 funcione con las complejidades ya enunciadas.

El enunciado de este trabajo menciona 3 posibles implementaciones de matrices esparsas (sección A, página 48), las cuales enunciamos a continuación junto con sus características pertintentes a nuestro contexto:

Dictionary of Keys (DoK) Es una estructura buena para construir la matriz incrementalmente con datos de entrada desordenados, pero no es eficiente para iterar los valores no nulos en orden, lo cual hace a esta estructura ineficiente para realizar operaciones algebraicas (como multiplcación matriz-vector). Suele usarse para construir la matriz, la que luego es transformada a un formato más eficiente para el procesamiento de los datos[**wiki˙dok**]. Es también llamada en otros ámbitos como **Hash Table Storage[alglib˙sparse**].

Compressed Row Storage (CRS) Es una estructura díficil de inicializar, pues a pesar de ser eficiente en cuanto al uso de memoria, los datos no pueden ser insertados en orden arbitrario[c]. Esto nos obliga a pasarle los datos en un orden específico para poder construir dicha estructura. Como ventaja, permite realizar operaciones algebraicas de manera eficiente, a pesar de requerir un direccionamiento indirecto a los elementos para acceder a los elementos una vez dada la posición del mismo[**wiki˙dok][alglib˙sparse**].

a. Término coloquial utilizada para referirnos a matrices poco densas, es decir matrices con muchos valores nulos.
b. Y más que optimización, necesidad, ya que para instancias grandes no nos alcanzaría con 16GB de memoria RAM para instanciar la matriz.
c. No es que no se pueda, pero se vuelve realmente difícil e ineficiente.

Compressed Column Storage (CCS) Es una estructura idéntica a CRS respecto a sus características. La diferencia radica en que en lugar de almacenar referencias a las filas, lo hace a las columnas. En otras palabras, podemos decir que dada una matriz A, CCS es el formato CRS de A^T[**netlib˙ccs**]. Al entrar más en detalles en los siguientes párrafos sobre la estructura de CRS, esto quedará más claro.

Observando estas características, claramente afrontamos un problema. Necesitamos una estructura para matrices esparsas que sea apta para operaciones algebraicas (como lo son CRS y CCS), pero en principio no podemos asegurar que la información de las instancias de entrada vengan manteniendo algún orden específico, con lo cual la construcción de dicha estructura no es viable. Podríamos usar DoK para construir sin problemas dicha instancia, pero esta estructura no es apta para las operaciones que luego necesitaremos realizar con ella.

La decisión entonces fue entonces la mencionada en casi todas las referencias: utilizamos un DoK para la instanciación de los datos, para luego transformar dicha estructura en un formato más adecuado para las operaciones algebraicas. Nuestra elección en este caso fue CRS, ya que interpretamos que la implementación de la operación multiplicación matriz-vector era más fácil con ésta que con CCS (como explicaremos dentro de unas secciones).

A continuación, pasamos a explicar los detalles de las estructuras elegidas y nuestra implementación de ellas.

Dictionary of Keys (DoK)

Los artículos investigados al respecto describen al DoK como una tabla de *hash*, la cual almacena los datos de entrada en tiempo constante (si asumimos que generar el hash es de tiempo constante, lo que suele ocurrir) pero de manera desordenada. Es decir, la estructura termina siendo un diccionario implementado sobre una tabla de hash, lo que nos da tiempo de inserción, acceso y modificación a los datos en tiempo constante.

En contrapartida, recorrer los elementos almacenados en orden (tanto por clave como por valor) no es una opción soportada. Ni tampoco saber, por ejemplo, para una fila dada que elementos no nulos se tienen almacenados de la misma (ni en qué columna). Esto último es lo que dificulta enormemente las operaciones algebraicas sobre esta estructura (consultar si hay un valor no nulo almacenado en una posición tiene costo constante, pero al multiplcar se requiere consultar columna por columna para una fila dada, obteniendo un costo final $\mathcal{O}(n)$ que no es mejor que lo que provee una matriz almacenada como vector de vectores[a]).

La utilidad de esta estructura, al elegirla, es la de poder realizar la inserción de datos de manera desordenada (u inserción en posiciones arbitrarias), para luego poder recorrer estos datos en forma ordenada, que es lo que necesitamos para poder pasar a una representación CRS. Pero esto último no parece ser una opción de una implementación clásica de DoK. Llegados a este punto, contemplamos las siguientes opciones:

- Una solución sería recorrer los elementos desordenados, utilizar un vector temporal para ordenarlos (donde cada elemento es una terna $< fila, columna, valor >$) y luego utilizar los datos ordenados para construir un CRS.
- Una alternativa sería realizar diccionario basado en un *AVL*[**wiki˙avl**] en lugar de una tabla de hash. Esto haría que los datos estén ordenados internamente (por el par $< fila, columna >$), aunque pagaríamos el costo de insertar los datos de manera ordenada (costo que en la solución previa, pagaríamos al ordenar el vector temporal).

La segunda opción planteada no es, estrictamente hablando, un DoK; ya que si bien tenemos un diccionario que nos asocia una llave o *key* con un valor, el mismo no tiene un costo constante para inserción y acceso a los datos. Sería un DoK si solo observamos la abstracción de los datos, y el hecho de que la inserción de datos desordenados (como su acceso) puede ser realizado sin mayores problemas[b].

Dado el uso que le daremos a esta estructura, observamos que el costo de la primer solución es $\mathcal{O}(nnz\ log(nnz))$ para nnz la cantidad total de valores no nulos[c]. En la alternativa, tendremos un costo de $\mathcal{O}(nnz\ log(nnz))$[d], es decir, el costo asintótico en cuanto accesos/escrituras a memoria es el mismo.

Así pues decidimos ir por la segunda alternativa, por el simple hecho de que cualquiera de las dos implementaciones se basarían en la *standard library* de *C++* (*unordered_map*[**stl˙unordered˙map**] y *map*[**stl˙map**]), pero usando *map* nos evitamos tener que crear un vector y ordenarlo para luego construir la matriz CRS.

a. Aunque se obtiene una mejora en cuanto a la complejidad espacial.
b. Ya que la abstracción de un diccionario implementado sobre AVL ya está realizada[**stl˙map**].
c. $\mathcal{O}(nnz)$ inserciones, $\mathcal{O}(nnz\ log(nnz))$ para ordenar el vector y $\mathcal{O}(nnz)$ para recorrerlo.
d. Crear la matriz tendá costo $\sum_{i=1}^{nnz} \mathcal{O}(log(i)) = \mathcal{O}(nnz\ log(nnz))$, y luego recorrerla en orden será $\mathcal{O}(nnz)$.

Resumiendo, nuestra implementación de (pseudo)DoK no es otra cosa que un $diccionario < enteros, diccionario < enteros, valor >>$, donde ambos diccionarios están implementados utilizando el contenedor *Map* de la *stl* de C++, el cual utiliza como estructura interna un árbol AVL y nos garantiza las complejidades enunciadas[**stl˙map**]. Los detalles más específicos sobre esta implementación se pueden encontrar en el apéndice B.

Compressed Row Storage (CRS)

Como ya hemos expresado, la implementación CRS tiene como características almacenar la información de la matriz esparsa de forma eficiente (sino no tendría mucho sentido) y permite implementar funciones aritméticas de manera igualmente eficiente.

La estructura de datos en si misma consta de 3 vectores: uno para los valores no nulos de la matriz (llamaremos a este vector val) en orden de aparición de los mismos al recorrer la matriz de izquierda a derecha y de arriba hacia abajo, otro vector (col_ind) que almacena los índices de las columnas de los elementos de val (es decir, si $val[k] = m_{ij}$, entonces $col\_ind[k] = j$) y por último row_ptr que almacena los índices sobre val que representan el primer valor no nulo de una fila (es decir, si $val[k] = m_{ij}$, entonces $row\_ptr[i] \leq k < row\_ptr[i+1]$). Por convención, se define a $row\_ptr[n+1] = nnz + 1$[**netlib˙crs**].

Como se puede observar, finalmente se almacenan $2nnz+n+1$ valores entre los 3 vectores, en lugar de almacenar n^2 si se hubiera usado la estructura intuitiva de vector de vectores.

Presentamos un ejemplo a continuación, para ilustrar la idea. Consideremos la siguiente matriz $A \in \mathbf{R}^{6\times6}$[**netlib˙crs**]:

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{bmatrix}$$

Y los vectores de la implementación CRS de esta matriz serían:

val	10	-2	3	9	3	7	8	7	3…9	13	4	2	-1
col_ind	1	5	1	2	6	2	3	4	1…5	6	2	5	6

row_ptr	1	3	6	9	13	17	20

Observando esto, queda claro por qué es complicado hacer una inserción arbitraría de elementos. Supongamos en el ejemplo que deseamos agregar un nuevo dato no nulo x en $a_{2,3}$. En primer lugar deberíamos incrementar el tamaño de val en uno, para luego insertar x en $val[5]$. Antes de hacer esto, debemos mover todos los valores a partir de $val[5]$ una posición dentro de val, es decir $val[i] \leftarrow val[i-1]$ para $i = 5..nnz$[a]. Lo mismo habrá que hacer en col_ind para poder insertar la columna de x (3) en $col\_ind[5]$. Ahora todos índices de row_ptr se encuentran desactualizados, con lo cual debemos incrementar en uno todos los índices a partir de la posición 3, pues las posiciones a las que hacían referencia en val fueron "corridas" una posición por el agregado de x. Por último, debemos actualizar el último valor almacenado en row_ptr, ya que nnz ha sido incrementado por la inserción de x.

En este ejemplo, con esto bastaría, pero si el nuevo elemento insertado se hubiera convertido en el nuevo primer elemento de la fila 2, deberíamos a su vez actualizar el índice de val que nos indica el primer valor no nulo de la fila 2: $row\_ptr[2] \leftarrow 5$. Todo este proceso tiene un costo total de $\mathcal{O}(2nnz + n)$, es decir, lineal en $max(nnz, n)$. Si comparamos este costo contra el costo de inserción de un simple vector de vectores ($\mathcal{O}(1)$), o contra nuestro DoK ($\mathcal{O}(log(nnz))$), o incluso el DoK clásico ($\mathcal{O}(1)$), veremos que claramente no es eficiente esta estructura para insertar nuevos valores. Aún así, es trivial ver que si los valores vienen ordenados (siguiendo el orden de izquierda a derecha y de arriba hacia abajo), la generación de estos vectores costará simplemente $\mathcal{O}(nnz)$, pues a medida que vamos iterando los valores ordenados, debemos simplemente ir agregando elementos al final de cada uno de los

a. Asumimos que nnz incluye al nuevo valor x.

vectores (en el caso de row_ptr, deberemos agregar un nuevo índice cuando un nuevo valor pertenece a una fila distinta que su valor anterior).

Cabe aclarar que este ejemplo no contempla la posibilidad de que haya una fila sin valores no nulos. Dado este caso, en nuestra implementación, decidimos colocar en la posición correspondiente de row_ptr un valor interpretable como ∞. En nuestro caso, ese valor es el valor máximo del tipo de datos del vector (*unsigned int*)[**stl˙limits**].

El último aspecto que nos queda por analizar es por qué esta representación de una matriz esparsa nos sirve para realizar operaciones aritméticas (por lo menos la multiplicación matriz-vector) de manera eficiente. Como ya hemos mencionado, si no usaramos una estructura que pueda saber en todo momento donde están los valores no nulos, al realizar la multiplicación de cada fila por el vector nos veremos obligados a realizar $2n - 1$ operaciones matemáticas (las n multiplicaciones y las $n - 1$ adiciones), de las cuales muchas serán multiplicar por 0[a] y siendo estas operaciones innecesarias[b]. Ahora bien, al utilizar la estructura de CRS, podemos iterar por los valores no nulos de una fila dada, y en cada caso saber exactamente cual es el índice del vector con el cual deben multiplicarse: el índice de la columna del elemento no nulo. Así pues, en cada multiplicación de vectores (fila de la matriz por vector), estaremos realizando la cantidad justa y necesaria de operaciones matemáticas, lo cual claramente hace más eficente a nuestro algoritmo del método de la potencia, el cual depende de realizar la operación $Mx^{(k-1)}$ en cada k iteración (algoritmo 3, página 13).

Así hemos introducido como es la estructura CRS y porque la misma sirve a los propósitos del actual trabajo: sus ventaja espaciales para almacenar matrices esparsas y para las operaciones aritméticas, sumados a su bajo costo de inicialización si los valores no nulos de la matriz puede ser recibidos en orden, cosa que conseguimos mediante el uso de una representación auxiliar de la matriz: DoK.

Compressed Row Storage (CCS)

Aprovechando, mencionamos que la implementación de CCS sigue exactamente la misma idea que CRS, salvo que invierte la idea de tener índices a columnas y conocer las posiciones de val inician una fila por tener índices a filas y conocer las posiciones de val que inician una columna. En la matriz A del ejemplo anterior tendríamos los siguientes vectores[**netlib˙ccs**]:

val	10	-2	3	9	3	7	8	7	3…9	13	4	2	-1
row_ind	1	2	4	2	3	5	6	3	4…5	6	2	5	6

col_ptr	1	4	8	10	13	17	20

En el contexto de este trabajo optamos por utilizar la representación CRS en lugar de ésta, ya que a la hora de realizar la operación aritmética de mayor peso para el método de la potencia era más importante poder iterar los valores no nulos de una fila más que los de una columna (ya que debemos hacer el producto interno de cada fila con un vector). Claramente esto no es un obstáculo insalvable dentro de esta estructura (basta con transponer la matriz y hacer los productos internos columna a columna), pero realizar esto nos pareció innecesario y poco intuitivo, más aún teniendo que implementar cualquier representación desde cero y pudiendo elegir CRS.

a. Cosa que incluso nos podría llevar a tener errores numéricos.

b. Se podría verificar que el valor de la matriz sea no nulo antes de efectuar la operación, pero dicha verificación no es más eficiente que realizar la multiplicación.

4. EXPERIMENTACIÓN

A continuación se detallan todos los experimentos realizados en este trabajo y sus resultados. Se detalla no sólo el experimento en si, sino que también se explican los resultados que se esperan comprobar y sus motivaciones.

4.1. PageRank

4.1.1. Comportamiento Esperado de PageRank

Objetivo Ejemplificar el comportamiendo esperado de PageRank. Proponemos, a su vez, que el orden obtenido por PageRank será el mismo para cualquier factor de navegación α[a], aunque las probabilidades del vector resultado (asociadas a cada nodo) cambiarán según este parámetro.

Proposición No creemos que haga falta aclarar mucho. Ya se presentó y explicó método a utilizar. Generaremos un grafo de entrada lo suficientemente pequeño que nos permita calcular a mano el resultado de PageRank y explicar el porque de su resultado en base al conocimiento del método.

Método de Experimentación Realizamos la búsqueda "site:uba.ar" en *Google*[**google**] y tomamos los primeros 10 resultados. Generamos una instancia de entrada utilizando las herramientas de la cátedra y estos 10 resultados. Por último, dada esta instancia de prueba, calculamos el *ranking* con el método PageRank para varios distintos factores de navegación α.

Resultados, análisis y discusión

En la figura 7 podemos observar el grafo de conectividad resultante de los 10 resultados obtenidos por la búsqueda realizada en Google. En el mismo se observan 4 dangling nodes conectados al grafo: *fvet.uba.ar, ffyb.uba.ar, derecho.uba.ar* y *videos.agro.uba.ar*, mientras que existen otros dos dangling nodes que no son alcanzados por ninguno de los nodos del grafo expuesto[b]: *orga2.exp.dc.uba.ar* y *iigg.sociales.uba.ar*[c].

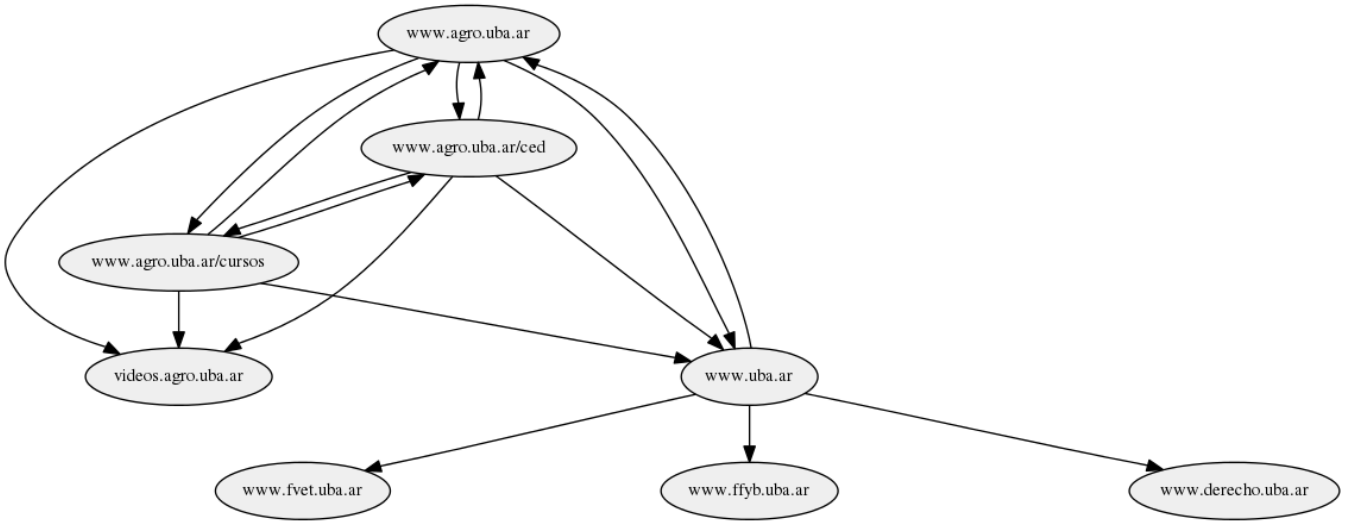


Figura 7. Grafo de conectividad de la instancia generada con los links obtenidos mediante la búsqueda en *Google*

Así pues, ya podemos comenzar a analizar un poco que es lo que esperamos encontrar del orden que nos dará PageRank. En primer lugar, los dangling nodes que están disconexos del grafo consideramos que deberían

a. Lo llamamos "factor de navegación" al parámetro α ya que el mismo determina el peso/importancia del grafo/matriz de navegación S (definida en la ecuación 5, página 5). Se puede observar en la definición de M en la página 21 que justamente α define en que porporción estarán incluídos en S y la matriz de teletransportaicón uniforme. Llamamos entonces a α factor de navegación, ya que de alguna manera nos indica directamente sin tener que pensar en $(1-\alpha)$ en que proporción participa S en M, y decimos que S es una matriz de navegación ya que es la matriz que contiene los datos de la matriz de conectividad con el inconveniente de los dangling nodes resuelto.

b. Y por lo tanto, no los hemos graficado.

c. Estos dos sitios que originalmente aparecían en los primeros 10 resultados de Google al momento de realizar este experimento, ya no lo hacen más.

tener el puntaje más bajo del grafo, ya que los mismos no tienen backlinks y por lo tanto se puede pensar que no reciben ningún "voto". De los restantes dangling nodes, 3 tienen una arista entrante proveniente del mismo nodo, con lo cual esperamos que estos tengan el mismo puntaje (ya que al recibir una arista cada uno del mismo nodo, reciben el mismo puntaje de este, pues cada nodo divide por igual su "poder de voto" entre todas sus ejes salientes): *fvet.uba.ar, ffyb.uba.ar, derecho.uba.ar*.

De los restantes nodos, podemos observar que tenemos 3 nodos con grado de entrada 3 (*agro.uba.ar, videos.agro.uba.ar, uba.ar*), y otros 2 nodos con grado de entrada 2 (*agro.uba.ar/ced, agro.uba.ar/cursos*). Y si observamos el grado de salida de los nodos (descartando los dangling nodes), tenemos que es igual a 4. Es decir, todos los nodos dividen su "poder de voto" por 4, pero 3 de ellos reciben 3 "votos", y los restantes 2 reciben 2 "votos". Así pues, para seguir analizando ya debemos observar de donde salen las aristas, ya que de provenir de un nodo con grado de entrada 3, tendran más peso que de salir de un nodo de grado de entrada 2.

agro.uba.ar Recibe los votos de: *uba.ar* (grado de entrada 3), *agro.uba.ar/ced* (grado de entrada 2), *agro.uba.ar/cursos* (grado de entrada 2).

videos.agro.uba.ar Recibe los votos de: *agro.uba.ar* (grado de entrada 3), *agro.uba.ar/cursos* y *agro.uba.ar/ced* (ambos con grado de entrada 2).

uba.ar Recibe los votos de: *agro.uba.ar* (grado de entrada 3), *agro.uba.ar/cursos* y *agro.uba.ar/ced* (ambos con grado de entrada 2).

agro.uba.ar/cursos Recibe los votos de: *agro.uba.ar* (grado de entrada 3), *agro.uba.ar/ced* (grado de entrada 2).

agro.uba.ar/ced Recibe los votos de: *agro.uba.ar/cursos*(grado entrada 2), *agro.uba.ar* (grado de entrada 3).

Observamos de este resumen que las primeras 3 páginas reciben 3 votos (uno de grado 3 y dos de grado 2). Las restantes 2 páginas reciben 2 votos (uno de grado 3 y el restante de grado 2). Con lo cual esperamos encontrar que estos primeros 3 sitios aparezcan primero en el orden ya que reciben un voto más, pero ninguno de los 3 deberá tener más puntaje que el otro. Seguido de estos, debemos tener las 2 páginas con 2 votos, nuevamente se repite que ninguno de estos dos sitios deberá tener más votos que el otro.

Así pues *a priori*, tendríamos un orden obtenido más basado en los grados de entrada de los nodos, ya que en ningún momento observamos que el "peso" de los votos altere el orden que nos daría *In-Deg*. Sobre esto desarrollaremos más en el siguiente experimento. Por lo pronto, observemos el cuadro 1 para analizar el orden obtenido.

Cuadro 1
Órdenes obtenidos y sus porcentajes para los resultados obtenidos de la búsqueda "site:uba.ar" en *Google*

Caso particular $\alpha = 0$		Casos $0 < \alpha < 1$									
Orden/Página	0	Orden/Página	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
derecho.uba.ar	0.1	agro.uba.ar	0.103548	0.107198	0.110953	0.114823	0.118812	0.122929	0.127182	0.131579	0.13613
orga2.exp.dc.uba.ar	0.1	uba.ar	0.103548	0.107198	0.110953	0.114823	0.118812	0.122929	0.127182	0.131579	0.13613
agro.uba.ar	0.1	videos.agro.uba.ar	0.103548	0.107198	0.110953	0.114823	0.118812	0.122929	0.127182	0.131579	0.13613
ffyb.uba.ar	0.1	agro.uba.ar/cursos	0.101023	0.102093	0.103212	0.104384	0.105611	0.106895	0.10824	0.109649	0.111127
uba.ar	0.1	agro.uba.ar/ced	0.101023	0.102093	0.103212	0.104384	0.105611	0.106895	0.10824	0.109649	0.111127
fvet.uba.ar	0.1	derecho.uba.ar	0.0984973	0.0969883	0.0954716	0.0939457	0.0924093	0.0908605	0.0892978	0.0877193	0.0861231
videos.agro.uba.ar	0.1	ffyb.uba.ar	0.0984973	0.0969883	0.0954716	0.0939457	0.0924093	0.0908605	0.0892978	0.0877193	0.0861231
iigg.sociales.uba.ar	0.1	fvet.uba.ar	0.0984973	0.0969883	0.0954716	0.0939457	0.0924093	0.0908605	0.0892978	0.0877193	0.0861231
agro.uba.ar/cursos	0.1	orga2.exp.dc.uba.ar	0.0959086	0.0916284	0.0871502	0.0824635	0.0775579	0.0724212	0.0670411	0.0614038	0.0554939
agro.uba.ar/ced	0.1	iigg.sociales.uba.ar	0.0959086	0.0916284	0.0871502	0.0824635	0.0775579	0.0724212	0.0670411	0.0614038	0.0554939
Probabilidad Total	1	Probabilidad Total	0.9999991	1.0000017	0.9999982	1.0000011	1.0000017	1.0000009	1.0000016	1.0000005	1.0000011

Lo primero que debemos señalar es que para obtuvimos siempre el mismo orden (con distintos valores de convergencia/probabilidad de permanencia) para $0 < \alpha < 1$; y únicamente obtuvimos un orden distinto para $\alpha = 0$.

En el caso $\alpha = 0$, esto ocurre pues la matriz M pasa a ser una matriz tal que $m_{ij} = {}^1/_n$ para todo i, j. Recordemos como era la definición de M de la ecuación 4.1.1:

$$M = \alpha S + (1 - \alpha)\frac{1}{n}ee^T \qquad \alpha \in \mathbf{R} \wedge \alpha \in [0, 1)$$

Luego, para $\alpha = 0$, tenemos que:

$$M = 0S + \frac{1}{n}ee^T$$
$$M = \frac{1}{n}ee^T$$

Por lo tanto, para este caso particular, tenemos una matriz que nos representa un grafo completo y completamente equiprobable. Es decir, volviendo a la idea del navegante aleatorio de la sección 1, que no importa donde se encuentre el navegante, el mismo se "moverá" a cualquier página web del grafo con la misma probabilidad. Así pues, tenemos que en realidad el navegante debería pasar la misma cantidad de tiempo en todas las páginas, lo que explica que el orden devuelto para este caso nos dé una probabilidad de $0{,}1$ para todas las páginas web/nodos. Claramente aquí el orden en que PageRank devuelve los sitios se debe a una cuestión del orden de entrada en cual fueron ingresados los sitios, ya que la coordenada 1 del vector resultante se corresponde con el sitio 1, la coordenada 2 con el sitio 2, etc. Si todos los nodos tienen la misma probabilidad, el vector nunca es ordenado según las probabilidades y lo que estamos observando es seguramente el orden en el cual fueron ingresados los nodos.

Para el resto de los casos, donde el orden obtenido no cambia, debemos decir que es un resultado con sentido. La elección del parámetro α afecta a los valores de la matriz M (ecuación 4.1.1, página 21), pero no afecta a la conectividad representada por la matriz. Y esto último es lo que mayor peso tiene en el modelo PageRank. Claramente puede ocurrir, para valores de α **muy pequeños**, que el orden sea distinto (tendiendo a equiprobable), pero a medida que este parámetro aumente, cada vez será menor la injerencia de la teletransportación y mayor será la de la conectividad del grafo/instancia inicial. En este argumento hay un factor clave que no se está teniendo en cuenta: el tamaño de la instancia. La injerencia de la teletransportación será cada vez mayor a medida que tengamos instancias más grandes, pues tendremos cada vez más subgrafos disconexos que serán convertidos a fuertemente conexos mediante el agregado de los ejes artificiales. No olvidemos que el concepto de teletransportación fue introducido artificialmente para salvar el escollo de la convergencia del cálculo del autovector. Es esperable entonces que a medida que estos ejes artificiales toman menor injerencia (recordar la definición de M) ya sea por un α muy grande y/o un tamaño de instancia chico[a] más se "acentúe" el orden basado en la premisa de PageRank: mayor puntaje a aquellos que tienen la mejor relación *cantidad de votos - calidad de votos* (observar que para $\alpha = 0{,}9$ la diferencia entre los primeros y los segundos aumentó, respecto de la diferencia para $\alpha = 0{,}1$, y lo mismo ocurre con los últimos). Aunque esto ocurre en ambas direcciones, a mayor tamaño de instancia, seguramente obtendremos distintos ordenes con α cada vez mayores.

Otra observación que podemos hacer pasa por la evolución de los porcentajes para los distintos α en el intervalo $0{,}1 \ldots 0{,}9$. Vemos que a medida que aumenta α, la mitad del ranking superior comienza a incrementar su probabilidad en contraparte con la mitad inferior, que se decrementa.

Si bien observamos que todos los nodos que tiene menos probabilidad a mayor α son dangling nodes, también podemos señalar que hay un dangling node entre los nodos que aumentan su probabilidad: *videos.agro.uba.ar*. Entonces, no podemos generalizar y decir que este comportamiento ocurre dependiendo de si un nodo tiene grado de salida 0 o no.

Analicemos entonces que significa aumentar α. Recordemos la ecuación de M presentada en la sección 1:

$$M = \alpha S + (1 - \alpha)\frac{1}{n}ee^T \qquad \alpha \in \mathbf{R} \wedge \alpha \in [0, 1)$$

Recordemos también que S no es otra cosa que la matriz de conectividad del grafo, habiendo ya resuelto el inconveniente de los dangling nodes. Así pues, vemos que el factor α maneja la proporción en la que S y la matriz de distribución uniforme ($(1/n)ee^T$) componen a M, siendo una inversamente proporcional a la otra. Ergo, al aumentar α aumenta el peso que tiene S (la cual podemos llamar *la matriz de navegación*, ya que se basa en el grafo de conectividad más los dangling nodes con capacidad de teletransportación) en M. Intuitivamente hablando, esto debería darle mayor importancia/probabilidad a aquellos nodos que ya se encontraban en la matriz de conectividad,

a. Recordemos que las instancias de este tipo suele crecer en nodos mucho más que en aristas.

ya que en esta los "pesos" de los ejes salientes tienen un peso proporcional a α respecto de la capacidad de teletransportación que le agrega a M la matriz de distribución uniforme.

Es razonable, entonces, pensar que a mayor α cobre más peso en el ranking aquellos nodos que dependen de S, y en esta matriz tenemos bien marcado el esquema de "votos" de PageRank: los nodos reciben su puntaje de acuerdo a la cantidad y "calidad" de los nodos que los apuntan. En este contexto, tiene sentido que la mitad superior del ranking aumente su probabilidad conforme aumenta α, ya que son los nodos a los que inicialmente S les otorga un mayor puntaje (recordar la definición de S en la página 5).

Para finalizar, debemos hacer notar al lector (sino lo ha hecho ya) que la suma total de las probabilidades para casi todos los casos difiere de 1. Esto no es otra cosa que un nuevo ejemplo de los errores númericos de la aritmética finita, de los cuales ninguna computadora puede escapar (a lo sumo puede mitigar sus efectos). Claramente, al correr el método de la potencia para obtener el ranking PageRank, el error númerio podría incluso afectar a la convergencia real (pues teóricamente ya sabemos que debería converger). Una de las medidas para mitigar esto es la de trabajar siempre con vectores normalizados desde el código en cada iteración del método, a pesar de saber que desde la teoría estos ya deberían estarlo[a].

A lo largo de este experimento analizamos un ejemplo pequeño para ilustrar el comportamiento del PageRank. El ejemplo en sí mismo quizás no fue el óptimo, ya que llegamos a la conclusión de que para la instanci a de prueba PageRank no se comportaría diferente que In-Deg (aunque esto no es tan grave ya que en el siguiente experimento ahondaremos en esto). Sin embargo, fue un ejemplo bueno que nos permitió exponer como funciona PageRank, cuando se comporta como *In-Deg* y, quizás lo más importante de este experimento, unos primeros análisis de como el factor de navegación α afecta al orden y las probabilidades del autovector resultante.

a. Medida que no tomamos, ya que nos dimos cuenta luego de haber corrido los experimentos.

4.1.2. Órdenes: Google vs PageRank vs In-Deg

Objetivo Analizar el orden obtenido respecto de otros órdenes disponibles. ¿Es igual? ¿Hay coincidencias? ¿Cuántas? ¿Tienen sentido?

Proposición Cualitativamente hablando, ¿cómo es el orden obtenido por PageRank? ¿Bueno? ¿Malo?. Obviamente que estas categorizaciones son intrínsecas a la realidad: sólo nosotros podemos decir que al realizar una búsqueda web, los resultados vinieron en un orden correcto o deseable (es decir, lo que se buscaba en las primeras posiciones). Utilizar nuestro criterio personal para hablar de la calidad del orden obtenido no sería muy correcto, ya que cualquier otra persona con criterios distintos podría disentir y ninguno de los criterios sería *a priori* más correcto que el otro[a]. Pero lo que si podemos hacer es comparar el resultado obtenido con otros resultados disponibles, de los cuales proponemos In-Deg (que se basa en el grafo de conectividad) y los resultados de un *search engine*: Google.

Método de Experimentación Utilizamos la misma instancia que en el experimento anterior. Sobre esta instancia sólo nos falta calcular el orden In-Deg (el cual no es otra cosa que ordenar a los nodos en orden descendiente según su grado de entrada, es decir según la cantidad de ejes que los apuntan). Luego utilizamos el orden provisto por Google en la búsqueda inicial más los órdenes obtenidos en el experimento previo.

Resultados, análisis y discusión

Cuadro 2
Órdenes comparativos entre los resultados de Google, PageRank e In-Deg

Google	PageRank	In-Deg
www.derecho.uba.ar	www.agro.uba.ar	www.agro.uba.ar
orga2.exp.dc.uba.ar	www.uba.ar	www.uba.ar
www.agro.uba.ar	videos.agro.uba.ar	videos.agro.uba.ar
www.ffyb.uba.ar	www.agro.uba.ar/cursos	www.agro.uba.ar/cursos
www.uba.ar	www.agro.uba.ar/ced	www.agro.uba.ar/ced
www.fvet.uba.ar	www.derecho.uba.ar	www.derecho.com.ar
videos.agro.uba.ar	www.ffyb.uba.ar	www.ffyb.uba.ar
iigg.sociales.uba.ar	www.fvet.uba.ar	www.fvet.uba.ar
www.agro.uba.ar/cursos	orga2.exp.dc.uba.ar	orga2.exp.dc.uba.ar
www.agro.uba.ar/ced	iigg.sociales.uba.ar	iigg.sociales.uba.ar

Los resultados obtenidos en 2 dieron resultados idénticos en cuanto a In-Deg vs PageRank, no así la búsqueda en google, que debe utilizar otras heurísticas que no consideramos en este trabajo. Asimismo, el orden de las búsquedas en google para un mismo término van cambiando a lo largo del tiempo[b] .

Respecto a In-Deg y PageRank, sus órdenes resultaron idénticos. Los motivos por lo cual esto ocurre fueron desarrollados en el experimento anterior, pero intuyendo que esto no tiene que ser siempre así (sino claramente PageRank no tendría sentido), realizamos un nuevo experimento. Creamos una nueva instancia de prueba basada en los resultados de buscar en wikipedia[**wikipedia**] distintos términos relacionados con los temas vistos en la materia. El grafo de conectividad resultante se puede observar en la figura 8.

a. Se podría ver muestralmente que opina la gente de distintos ámbitos, pero esto escapa al objeto de estudio de este trabajo.
b. True Story.

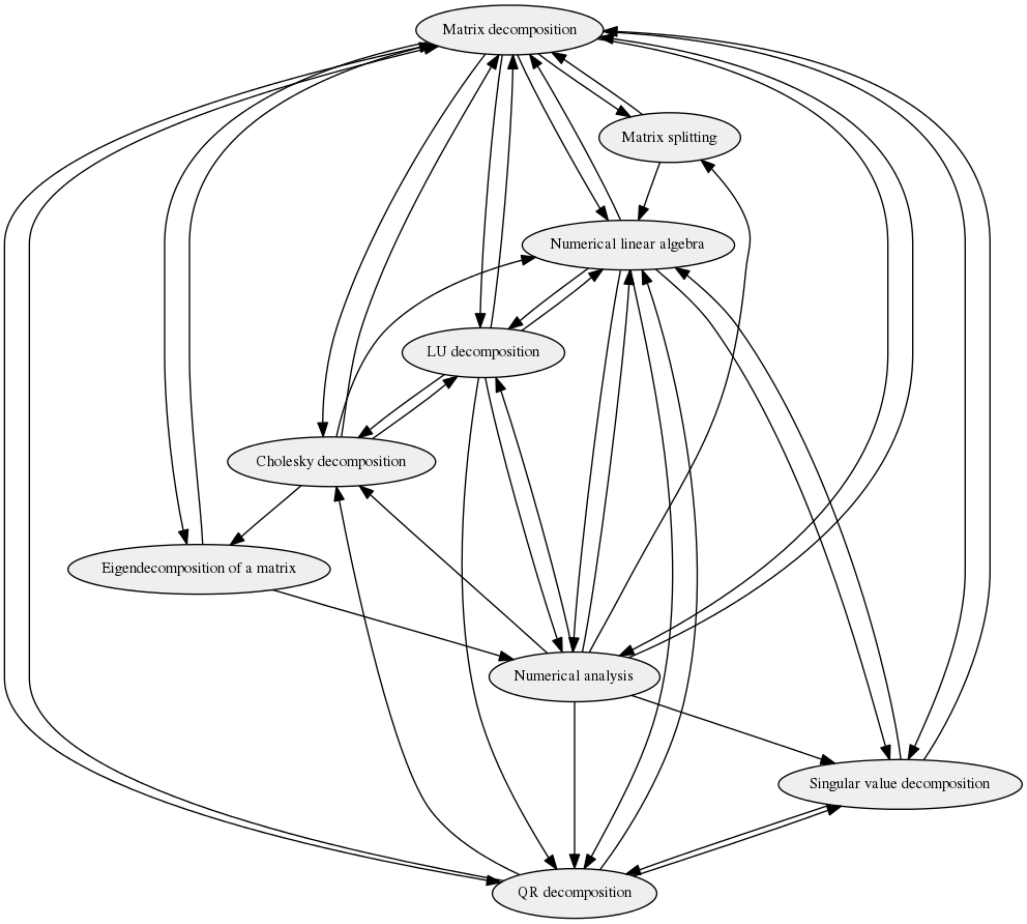


Figura 8. Grafo de conectividad de páginas de Wikipedia relacionadas con métodos numéricos

Sobre estas páginas, calculamos los órdenes de PageRank e In-Deg, cuya comparativa se encuentra detallada en el cuadro 3. Nuevamente, In-Deg y PageRank dieron resultados muy similares, pero no idénticos. Más aún, en In-Deg quedaron varios nodos empatados, teniendo la misma cantidad de ejes entrantes. Pero esto en PageRank no ocurrió, quedando definido un órden total.

Cuadro 3
Órdenes comparativos entre PageRank e In-Deg para Wikipedia

PageRank		In-Deg	
Puntaje	Nodo	Puntaje	Nodo
0.196	Matrix_decomposition	8	Matrix_decomposition
0.165	Numerical_linear_algebra	7	Numerical_linear_algebra
0.125	QR_decomposition	5	QR_decomposition
0.106	Numerical_analysis	4	Numerical_analysis
0.105	Singular_value_decomposition	4	LU_decomposition
0.098	LU_decomposition	4	Cholesky_decomposition
0.093	Cholesky_decomposition	4	Singular_value_decomposition
0.057	Eigendecomposition_of_a_matrix	2	Eigendecomposition_of_a_matrix
0.050	Matrix_splitting	2	Matrix_splitting

La diferencia entre ámbos órdenes está en la ubicación de *Singular value decomposition*. Mientras que en PageRank se encuentra en la posición 5, In-Deg lo lista en la 7ma ubicación. En el caso de In-Deg, la determinación de la

ubicación es determinística, dependiendo del grado de entrada del nodo que lo representa. Pero en nuestro ejemplo, vemos que existe un empate con otros 3 nodos (como ya se ha explicado), con lo cual aquí el orden también depende de la estabilidad y/o criterio de desempate que implemente In-Deg. En nuestro caso, la implementación es estable, con lo cual se respeta el orden inicial (o numeración) de los nodos. Por el otro lado, observando el grafo podemos entender porque PageRank lo ubica en una posición más alta que In-Deg: El nodo *Numerical Linear Algebra*, uno de los de mayor puntaje (y grado de entrada) tiene un link al nodo en cuestión y a *LU decomposition*, pero no al resto de los valores que empatan en In-Deg. Por lo visto en el experimento 4.1.1, sabemos que esto tiene un efecto de subir el puntaje notoriamente en los nodos "linkeados". Luego, utilizando el mismo razonamiento, observamos que *Single value decomposition* queda por encima de *LU decomposition* por el voto/link de *QR decomposition*.

Semánticamente, podemos observar que **en general**[a] quedan primeros en el ranking términos más *generales* o *abarcativos*; y a medida que avanza el ranking se encuentran términos mas particulares. Claramente, esta jerarquía de *generalidad* es árbitraria para los autores de este trabajo[b], aunque estimamos que habrá muy pocas posibilidades de disenso al respecto para los temas representados por los nodos del ejemplo.

Para evidenciar aún más la diferencia entre los algoritmos de In-Deg y PageRank decidimos alterar explícitamente el grafo de conectividad de este último ejemplo, agregando aristas desde los 5 nodos peor puntuados hacia uno de los últimos en el ranking In-Deg (*Eigendecomposition of a matrix*). Esto deberia aumentar drásticamente el rankeo del ultimo elemento en In-Deg ya que aumentamos su grado de entrada en 5, pero no tanto asi en Pagerank donde la "calidad" de los votantes tiene una mayor injerencia. Los resultados de este último experimento pueden verse en el cuadro 4.

Cuadro 4
Órdenes comparativos entre PageRank e In-Deg para Wikipedia con grafo alterado explícitamente

PageRank		In-Deg	
Puntaje	Nodo	Puntaje	Nodo
0.189	Matrix_decomposition	8	Matrix_decomposition
0.137	Numerical_linear_algebra	7	Numerical_linear_algebra
0.123	Numerical_analysis	7	Eigendecomposition_of_a_matrix
0.114	Eigendecomposition_of_a_matrix	5	QR_decomposition
0.108	QR_decomposition	4	Numerical_analysis
0.097	Singular_value_decomposition	4	LU_decomposition
0.089	LU_decomposition	4	Cholesky_decomposition
0.087	Cholesky_decomposition	4	Singular_value_decomposition
0.051	Matrix_splitting	2	Matrix_splitting

Puede observarse que respecto a In-Deg el elemento con ejes entrantes artificiales paso a ser el tercero por su nuevo grado de entrada aumentado. En Pagerank, el elemento subió de ranking pero quedó por debajo de Numerical_analysis, lo cual en principio es raro, ya que este item quedó con puntaje 4 en In-Deg. Si miramos el grafo, veremos que, efectivamente, el hecho de que *Numerical analysis* sea apuntado por *Matrix decomposition* y *Numerical linear algebra*, que son los términos mas importantes, le da mas potencia al elemento en cuestión que al elemento *Eigendecomposition of a matrix*, apuntado por los últimos 5 de la lista.

A lo largo de este experimento pudimos evidenciar las diferencias que existen entre dos algoritmos de elaboración de rankings distintos: In-Deg y PageRank. Esto, que no lo habíamos podido exponer en el experimento previo, nos demostró el peso que PageRank le otorga a los links provenientes de páginas web con mejores puntajes, diferenciación que In-Deg no realiza. Más aún, nos encontramos conque In-Deg parece ser tener más posibilidades de empate en su forma de "rankear" que PageRank, volviéndose dependiente del criterio de desempate que se utilice y convirtiéndose en otra faceta/problema a resolver, situación que puede ser despreciada en el caso de PageRank (las probabilidades de empate ya son chicas para pocos nodos, y las mismas son cada vez menores a mayor cantidad). Como dato de color, observamos que los resultados devueltos por *Google* son bastante distintos de los obtenidos con PageRank, con lo cual queda claro que a pesar de haber sido este método un hito en la historia

a. *QR* queda por encima de *Numerical Analysis*, creemos que esto se debe a la morfología del grafo de referencias entre artículos en Wikipedia.
b. ¿Suma puntos hablar en tercera persona de nosotros mismos? Very Scientific!

del motor de búsqueda, el mismo ya a evolucionado muchísimo en pocos años, lo que demuestra la importancia del problema estudiado.

4.1.3. Convergencia de PageRank

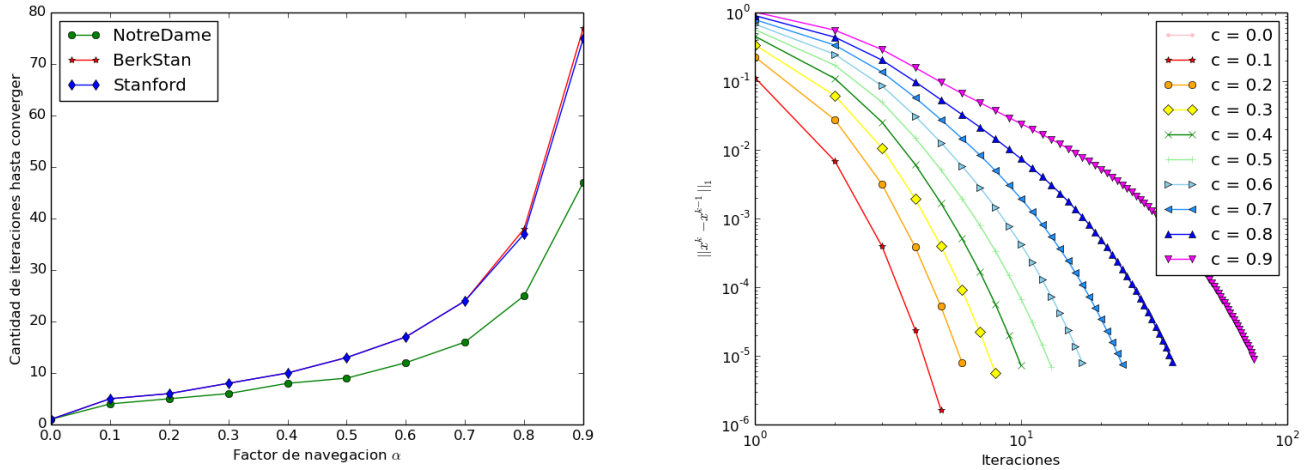
Objetivo Estudiar como se comporta la convergencia en función de el factor de navegación α.

Proposición PageRank propone una solución donde se observa a un conjunto de páginas web como un grafo dirigido. Luego pasa este modelo a uno matemático, utilizando una matriz para computar una solución. Esta matriz, por construcción, termina representando a un grafo completo (el grafo original no necesariamente lo era), y los valores del mismo dependen principalmente de 2 variables: ϵ y α (algoritmo 2, página 12 y ecuación 4.1.1, página 21). Si se observa el algoritmo, se verá que claramente modificar el ϵ tendrá como resultado hacer que cada corrida converja en más o menos iteraciones, ya que el criterio de parada depende de él. Así pues, no vemos como algo rico experimentar con este valor. En cambio, α es una variable que modifica en gran medida a nuestra matriz M, con lo cual es dificil saber cual será su injerencia en la convergencia (en principio). El objetivo, entonces, es ver como α afecta al método matemático iterativo de la potencia en cuanto a su convergencia.

Método de Experimentación Tomaremos 3 instancias de grafos de conectividad de páginas web de tamaño mediano-grande, obtenidas en http://snap.stanford.edu/data/#web. Luego, correremos el algoritmo de PageRank para $\alpha = 0{,}0; 0{,}1; 0{,}2; \ldots; 0{,}9$ y ϵ fijo en $0{,}00001$.

Resultados, análisis y discusión

En el gráfico 9a se exponen la cantidad de iteraciones necesarias hasta converger a medida que el parámetro α crece. Para las 3 instancias el comportamiento observado es el mismo: a medida que el parámetro α aumenta, la cantidad de iteraciones necesarias para converger crece exponencialmente. Puede observarse que aunque con valores numéricos diferentes, las curvas pertenecen a la misma familia de funciones, y por lo tanto tienen un comportamiento idéntico (empíricamente hablando) en cuanto a la cantidad de iteraciones en función de α.



Iteraciones hasta Converger en función de α (a) Escala Lineal

Velocidad de convergencia en función de α (b) Escala logarítmica

Figura 9. Análisis de Convergencia en función de α

Como se explicó en la sección 1, el factor α hace variar la proporción entre S y la matriz equiprobable a la hora de definir a M. Es decir, en el rango de posibles valores de α, la única manera en la que se afecta a M es en los valores de cada uno de sus elementos. Pero para ningún α se puede pasar a tener un elemento nulo en M (justamente se quería tener la representación de un digrafo completo). Así pues, la pregunta pasa a ser: ¿Por qué a mayor α se necesitan más iteraciones para converger?

Se nos ocurre que, volviendo al contexto de una cadena de Markov y el navegante aleatorio, un α pequeño nos genera una matriz de transición "más" equiprobable (recordar la definición de M en la ecuación 4.1.1, página 21) y dado que el método de la potencia comienza inicialmente con el vector equiprobable, le toma pocas iteraciones converger. A medida que α aumenta, el grafo generado denota más la navegación estricta por los links entre los

sitios, reduciendo la probabilidad de teletransportación. Es decir, a mayor α, el grafo se "parece más" al grafo de conectividad original no adulterado (con los dangling nodes ya resueltos), en el sentido de que S predomina mucho más en M que $(1/n)ee^T$. Esta matriz S no necesariamente representa un grafo fuertemente conexo, una de las condiciones necesarias para asegurar convergencia del método de la potencia en el contexto de este problema, con lo cual el método de la potencia inicia sus iteraciones con una, si se quiere, **seguridad de convergencia más débil**.

Respecto a la velocidad de convergencia, exponemos los resultados de una sola instancia ya que para las 3 instancias los resultados son similares. Como puede observarse en el gráfico 9b, a medida que el factor de navegacion α crece, la velocidad de convergencia disminuye, obteniéndose curvas cada vez más pronunciadas.

La "velocidad de convergencia" la definimos como la distancia *Manhattan* entre los autovectores $\vec{x^{(k)}}$ y $\vec{x^{(k-1)}}$ calculados en dos iteraciones consecutivas. Mientras mayor sea la distancia, más rápido decimos que se converge[a], pues se está acercándose cada vez **más rápido** al umbral de corte del algoritmo (basado en ϵ o una cantidad fija máxima de iteraciones[b]).

Nuestra explicación para este comportamiento es exactamente la misma que se expuso para el caso de las iteraciones hasta converger en función de α. Sin ir más lejos, estamos hablando del mismo fenómeno, salvo que en el caso de la figura 9b hacemos hincapié en **como** se acerca el método al vector solución, mientras que en la figura 9a observamos más en detalle la **cantidad de iteraciones** que se necesitan. Sin embargo, ambos gráficos nos muestran lo mismo: *a qué velocidad*, o *cuánto tiempo/iteraciones* se requieren para converger en función de α.

Ahora bien, hemos visto como α afecta a la convergencia. Si esto fuera lo único, claramente siempre elegiríamos el valor que nos permita converger más rápido. Ocurre que, como vimos en el experimento 4.1.1, el α afecta al orden obtenido también, es decir, a la calidad del resultado. Así pues, la elección de este parámetro será la búsqueda del equilibrio *efectividad* y *eficiencia*, u entre calidad del resultado y tiémpo de cómputo (intuitivamente, más iteraciones implican mayor tiempo de cómputo). La bibliografía consultada nos indica que en su momento, Google fijó inicialmente este parámetro en $0{,}85$, considerando a este un *trade-off* lo suficientemente bueno[**Bryan2006**][**Langville2006**].

Así pues concluimos nuestra experimentación sobre la convergencia del método de la potencia para PageRank. Llegamos a un resultado inmediato que nos dice que a mayor α tendremos una velocidad de convergencia menor y, ergo, una cantidad de iteraciones necesaria mayor para obtener un resultado lo suficientemente apróximado. También evaluamos brevemente la contracara de tener un tiempo de convergencia chico: el resultado obtenido no necesariamente es el mismo ya que predomina más la equiprobabilidad de los ejes de teletransportación artificalmente añadidos a la matríz M. También obtuvimos como resultado que este comportamiento es, presumiblemente (haría un falta un análisis más minucioso para poder afirmarlo), el mismo para cualquier instancia de entrada, más allá de que la cantidad de iteraciones absoluta entre dos instancias pueda variar para el mismo α, el comportamiento en función de α parecería ser el mismo.

a. Asumiendo que $\vec{x^{(k)}} < \vec{x^{(k-1)}}$, caso contrario nos estaríamos alejando del criterio de corte del método, es decir, de converger.

b. Esto se implementa, ya que si bien la teoría nos indica que el método convergirá, los errores númericos de la aritmética finita puede hacer que esto no ocurra.

4.1.4. Análisis de Tiempo de Cómputo

Objetivo Analizar la complejidad temporal del método.

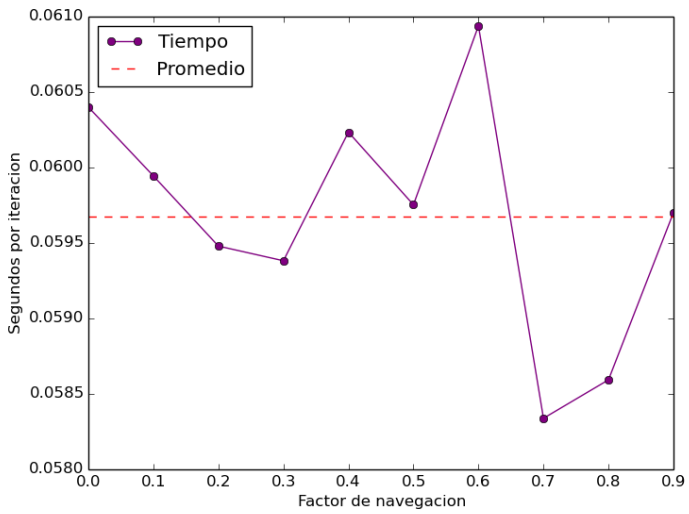
Hipótesis Proponemos que el tiempo de cómputo por iteración para una instancia y ϵ fijos será siempre el mismo para todo α. También proponemos que el tiempo de cómputo por interación para ϵ fijo será el mismo para **toda instancia** (sin importar tamaño, cantidad de ejes, etc).

Proposición De los experimentos previos hemos llegado a la conclusión de que a mayor α, mayor cantidad de iteraciones serán necesarias para converger, lo cual implica inmediatamente un mayor tiempo de cómputo requerido. La pregunta ideal a responder sería "cuánto tiempo", pero también concluimos previamente que la cantidad de iteraciones es dependiente (entre otros factores) de la instancia de entrada. Con lo cual, responder a esta pregunta en el contexto de este trabajo es imposible sin una cota teórica para la cantidad de iteraciones (que ni siquiera sabemos si existe). En cambio, lo que sí podemos analizar es si el tiempo de cómputo **por iteración** es el mismo para toda instancia, sin importar el ϵ[a] o α. Analizamos entonces si el tiempo por iteración varía según la densidad de la instancia inicial, y a su vez si varía para distintos valores de α.

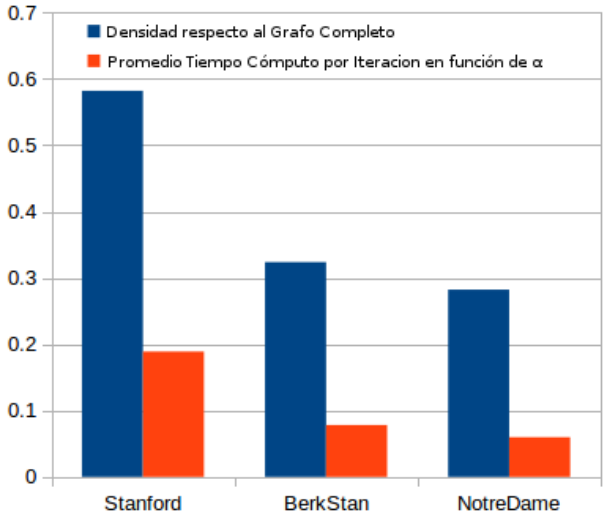
Método de Experimentación Tomamos 3 instancias de tamaño mediano-grande, con una diferencia relativa de densidad entre ellas significativa. Particularmente, tomamos las mismas del experimento anterior. Luego resolvemos cada instancia con PageRank 10 veces para $\alpha = 0{,}0; 0{,}1; 0{,}2; \ldots; 0{,}9$[b]. Tomamos entonces para cada instancia y α fijos, el promedio de los tiempos de cómputo. Por último, calculamos el tiempo de cómputo por iteración dividiendo este promedio por la cantidad de iteraciones totales que necesitó el método para converger.

Resultados, análisis y discusión

Consideramos 2 enfoques para extraer conclusiones, el primero analiza el tiempo consumido por iteración por el método de la potencia para diferentes valores de α y el segundo se centra en el tiempo por iteración respecto de la densidad de la instancia/grafo de entrada[c].



(a) Tiempo por Iteración en función de α.



(b) Tiempo por Iteración promedio vs Densidad del Grafo

Figura 10. Análisis de Tiempo de Cómputo

Para el primer enfoque, observemos los resultados obtenidos en la figura 10a. En el mismo observamos dos curvas. La primera, *Tiempo*, que representa el tiempo por iteración promedio para cada α (calculado como el tiempo promedio hasta converger de las 10 ejecuciones para un α fijo, dividido por la cantidad de iteraciones que necesitó);

a. Más allá de que para nuestros experimentos lo estemos dejando fijo.
b. Totalizando un total de 300 corridas del método (10 corridas para 10 valores de α para 3 instancias distintas).
c. Al referirnos a *densidad* de un grafo, nos referimos a la cantidad de ejes que tiene. A mayor cantidad de ejes, más denso es.

y la segunda *Promedio* que no es otra cosa que el promedio de los tiempos por iteración calculados para la curva *Tiempo*. Los resultados expuestos en esta figura son similares para las 3 instancias de prueba que se tomaron, con lo cual se decidió exponer una sola.

Puede observarse en esta misma figura que el tiempo por iteración calculado en función de α oscila en valores muy pequeños alrededor del promedio. Para ilustrar esto, presentamos en el cuadro 5 las métricas empíricas del experimento.

Cuadro 5
Métricas del tiempo por iteracion respecto de α

Métrica	Segs
Promedio	0.059676
Desvío Estándar	0.000788
Mínimo	0.058338
Máximo	0.060938
Diferencia %[a]	4.266231

Estas variaciones seguramente se deben al *scheduler* del sistema operativo y a fenómenos de bajo nivel de las corridas de los experimentos. De hecho, ya vimos en el experimento 4.1.3 que a medida que aumentamos el α, más iteraciones serán necesarias para converger. Obviamente esto implica un mayor tiempo de cómputo, ergo, la ejecución del experimento se vuelve aún más sensible pues al estar más tiempo ejecutándose más probabilidades hay de que el Sistema Operativo decida darle el CPU a otro proceso de mayor prioridad que surja (o que al estar tanto tiempo ejecutándose, su prioridad vaya bajando).

Finalmente concluimos que que el tiempo por iteración es constante, ya que en las métricas nos confirman, al observar los valores muy pequeños del desvío estándard y diferencia porcentual, que estas diferencias para distintos α sean muy probablemente errores de medición (recordemos que además estamos trabajando con instancias de entrada de tamaño mediano-grande).

Habiendo llegado a la conclusión que la hipótesis sobre el tiempo de cómputo por iteración es constante, debemos analizar por qué. Repasando la sección 3.3, podemos decir que es razonable el comportamiento presupuesto y observado, ya que el valor de α no modifica la *densidad* de la matriz original de conectividad A. Recordemos entonces que nuestra implementación se basa en el algoritmo 3, página 13, que multiplica usando justamente esta matriz A y aprovechando que la misma es esparsa. Al no modificar esta característica la selección del parámetro α, podemos afirmar que la cantidad de $flops$ necesarios para el producto matriz-vector se mantendrá constante en función de α (de hecho, α es un parámetro que luego multiplica al vector resultante $A\vec{x}^{(k-1)}$ en nuestro algoritmo, con lo cual no afecta en lo absoluto al producto matriz-vector que involucra a A).

El segundo enfoque centra su atención en el tiempo por iteracion respecto a la densidad del grafo asociado a la instancia de entrada. Al contrario que lo esperado, la densidad (cantidad de ejes) del grafo **sí** altera el tiempo de cómputo por iteración. Si observamos la figura 10b, observaremos que a mayor densidad, mayor tiempo por iteración se necesita.

En esta figura se compara el tiempo promedio consumido[b] contra una medida de densidad del grafo[c], y se ve, para las 3 instancias evaluadas, que el crecimiento del tiempo de cómputo parecería ser líneal en función de la densidad del grafo. Dado que sólo experimentamos con estas 3 instancias, sería muy apresurado hacer dicha afirmación, pero sí podemos decir que tenemos sospechas de que eso ocurra, dejando este aspecto para un futuro posible experimento[d].

Volviendo al hecho de que a mayor densidad, mayor tiémpo de cómputo, llegamos a la conclusión de que esto se debe al mismo motivo que en el primer enfoque, salvo que esta vez justifica el hecho de que **sí** se requiera más tiempo de cómputo. Ocurre que al ser la instancia de entrada más densa, menos esparsa será A (más ejes implica

a. Diferencia %: 100*(max-min)/max.

b. Se utiliza el valor promedio del tiempo para todos los factores de navegación.

c. Se toma como métrica un cociente entre la cantidad de aristas del gráfo y la cantidad de aristas de una *clique* de su misma cantidad de nodos; multiplicado por una constante para igualar las magnitudes con los valores de los tiempos de cómputo.

d. Vale la pena aclarar, en este caso, que podríamos considerar a nuestras 3 instancias como poco densas, más alla de la diferencia notoria de densidad entre ellas.

mayor cantidad de valores no nulos en la matriz), y por lo tanto el producto $A\vec{x}$ de la implementación del método efectuará más $flops$.

Finalizando, concluimos que una de nuestras hipótesis era correcta, y la otra no. Peculiar es que las justificaciones que le encontramos a ambos comportamientos (la relación tiempo de cómputo por iteración en función de α y de la densidad de la instancia) es la misma: mientras que α no afecta a la densidad de la matriz de conectividad pesada A, cosa que si lo hace la cantidad de ejes del grafo. Es decir, terminamos entendiendo que el principal aspecto a tener en cuenta a la hora de ver como se verá afectado el tiempo de computo (para nuestra implementación basada en el algoritmo propuesto por Kamvar et al.[**Kamvar2003**]) será la densidad/esparcidad[a] de la matriz inicial del modelo.

a. Coloquialismo.

4.2. GeM

4.2.1. PageRank/GeM vs Orden Total Conocido

Objetivo Analizar la convergencia del modelo GeM asumiendo que existe ranking ideal y correcto al cual converger.

Hipótesis PageRank, utilizando el modelo GeM, converge finalmente a un "Orden Real" o "Correcto" de los competidores deportivos, si este existe. Además, no necesitará de un grafo completo (los resultados de todos contra todos) para converger al mismo.

Proposición Como se comentó previamente en la experimentación sobre páginas web, establecer cuál es el "mejor orden" u "orden correcto" es completamente arbitrario. No hay una vara sobre la cual medir. En los deportes esto es aún más evidente, ya que depende de los resultados deportivos, ¿y quién es capaz de afirmar que la probabilidad de que Platense -el mejor equipo del mundo e insipirador del título del enunciado de este trabajo- le gane al Barcelona es 0? Así pues, en el caso de los deportes tampoco tenemos un orden total, conocido y determinístico para verificar que el resultado de PageRank es el correcto. Pero si existiese este orden, si fuese determinístico, ¿PageRank convergiría al mismo?

Método de Experimentación Generamos dos instancias ideales y completamente abstractas de los resultados de un torneo de fútbol con 10 y 50 equipos respectivamente, que juegan todos contra todos una única vez (45 partidos para la primera instancia, 1225 para la segunda). Las instancias son construidas de manera tal que $equipo_i$ le gana a $equipo_j$ si y sólo si $i < j$. Es decir, el ranking correcto es la numeración de los equipos de forma ascendente, y cada $equipo_i$ ocupa el puesto i.

Aprovechamos el hecho de que en los deportes hay una componente temporal y generamos los partidos por fecha (es decir, grupos de partidos que ocurren todos al mismo tiempo, o al menos así será para la perspectiva del algoritmo, que recibirá todos los partidos de una fecha juntos). En cada fecha se enfrentan, de a dos, equipos que no se hayan enfrentado todavía y que no jueguen otro partido esa misma fecha. Dentro de esas restricciones, los enfrentamientos de cada fecha se eligen al azar, pero con una semilla fija (5) para obtener reproducibilidad en los experimentos. **Notar que la cantidad de fechas necesarias para que se jueguen todos los partidos no está determinada únicamente por la cantidad de equipos sino que depende también de como se elijan los enfrentamientos de cada fecha. Esto se debe a que confeccionar un generador de instancias aleatorias que respete las restricciones y genere un fixture de $n-1$ equipos es complejo y escapa a los objetivos de este trabajo. Así pues, nuestra instancia de 10 equipos consta de 11 fechas y la instancia de 50 equipos consta de 53 fechas**.

El resultado de todos los partidos es siempre 1 a 0. No hace falta considerar empates dado que siempre hay un ganador.

Ejecutamos GeM tantas veces como fechas haya, pasándole en cada instancia una fecha más. Es decir, en la ejecución 1 le pasamos los resultados de la fecha 1, en la 2 los resultados de la fecha 1 y 2, y así sucesivamente. Para cada resultado de GeM, comparamos el ranking obtenido con el correcto, para alguna medida de distancia entre rankings, que definiremos más adelante.

Hace falta considerar un detalle importante, que es qué decisión tomar ante empates del "puntaje" asignado por GeM. Si desempatáramos por número de equipo de manera ascendente caeríamos en el molesto caso de que ya desde antes de empezar el torneo la salida de GeM coincidiría con el orden ideal. Lo mismo vale para cualquier subconjunto de competidores empatados en un momento dado: su orden relativo coincidiría con el ideal, aunque por casualidad y no por virtud del algoritmo. Resolvimos entonces "romper" el caso y que el desempate se realice de manera descendente, asegurándonos así de no evaluar como correctos resultados en donde hay muchos empates.

Repetimos el experimento variando los valores de α en factores de $0{,}2$, para estudiar la convergencia en cada caso. De confirmarse nuestra hipótesis la diferencia/distancia del ranking respecto del orden total ideal (que sabemos que existe por construcción) debería llegar a ser 0, eventualmente "antes" de que se hayan jugado todas las fechas.

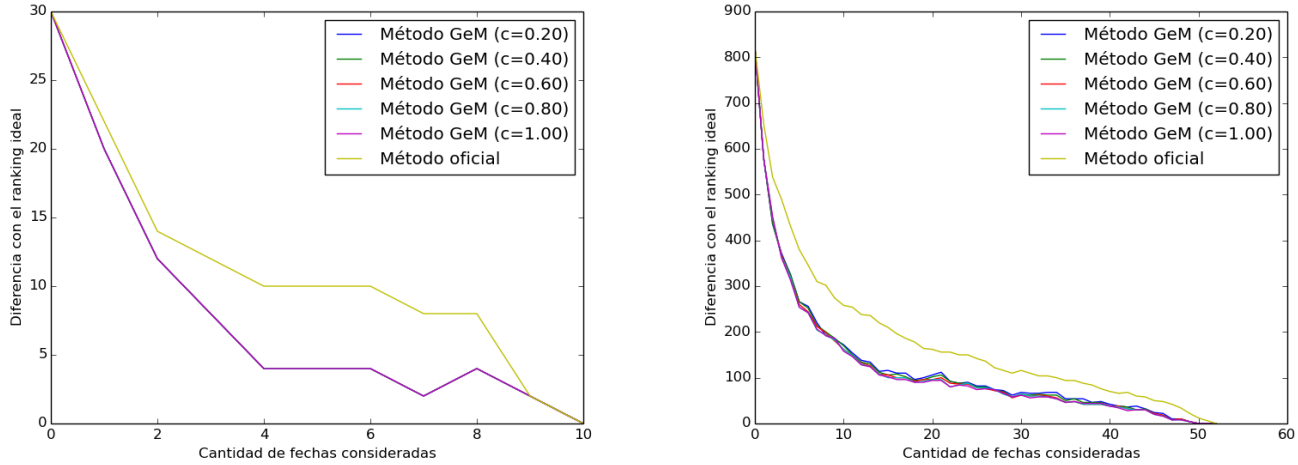
Resultados, análisis y discusión

En primer lugar, como hemos adelantado, debemos decidir cómo determinar la "distancia" entre dos rankings. Es decir, determinar alguna manera de comparar dos rankings dados, y poder tener una magnitud de cuan parecidos son. Para ello consideramos algunas opciones dados dos rankings A y B, de entre las cuales mencionamos:

1. La sumatoria de la distancia, para cada competidor, entre su posición en el ranking A y la posición en el ranking B.

2. La cantidad de competidores que aparecen, en A, en una posición distinta a la que aparecen en B.
3. La sumatoria, para cada competidor, de la cantidad de equipos que tiene por encima en A y que tiene por debajo en B.

Luego de evaluar estas opciones (y algunas más no tan concisas), decidimos trabajar de aquí en más con la distancia basada en la suma de las distancias de todos los competidores (definición de distancia número 1). Consideramos que dicha forma de tomar la distancia entre dos rankings, de alguna manera nos está señalando cuántas permutaciones de elementos/equipos contiguos hay entre un ranking y el otro, cosa que no queda tan claro con las demás opciones. Y esa forma de medir la distancia es la que consideramos que se ajusta a lo que queremos observar del dominio del problema: cuántos equipos están ubicados distinto y cuán lejos.



(a) Torneo de 10 equipos

(b) Torneo de 50 equipos

Figura 11. Distancia al orden total ideal/correcto ($c = \alpha$)

La hipótesis fue confirmada. GeM converge al Orden Real para todos los valores no nulos de α y no precisa todas las fechas para hacerlo, como se puede observar en la figura 11. Sin embargo, necesita una cantidad significativa: para todos los valores no nulos de α fueron necesarias 10 de 11 fechas para el caso de 10 equipos y 50 de 53 fechas para el caso de 50 equipos.

El caso $\alpha = 0$ no funciona porque la matriz termina descartando los resultados de los equipos y usando simplemente la misma probabilidad para cualquier equipo. En adelante dejaremos de lado este caso.

La variación α en el caso de 10 equipos no representó diferencias en el orden devuelto en cada fecha. es por eso que en la figura 11a no se ven más que dos curvas (una de las cuales corresponde al siguiente experimento): estan todas superpuestas. En el caso de 50 equipos sí hubo leves diferencias en el orden devuelto en cada fecha. Observamos que a mayor α el algoritmo en general converge "mejor" al Orden Real (es decir, dada la misma cantidad de fechas, se acerca más), aunque siempre precisa 50 fechas para converger al orden real (figura 11b). La diferencia de todos modos es pequeña y probablemente se deba a que, en nuestro modelo ideal, las victorias son totalmente transitivas: la probabilidad de que C le gane a A dado que A le ganó a B y B le ganó a C es siempre 0. Como valores pequeños de α tienden a agregar una probabilidad de que C efectivamente le gane a A, la convergencia mejora levemente para valores altos de α (donde esa probabilidad artificial es cada vez menor). Sin embargo, variando la semilla usada para ordenar los enfrentamientos encontramos casos en donde un valor mayor de α empeoraba la convergencia en algún punto (ver figura 12a), por lo que no podemos generalizar esta conclusión.

Así pues, motivados por estos resultados, realizamos la siguiente experimentación.

PageRank vs Ranking FIFA en un caso de Orden Total Conocido

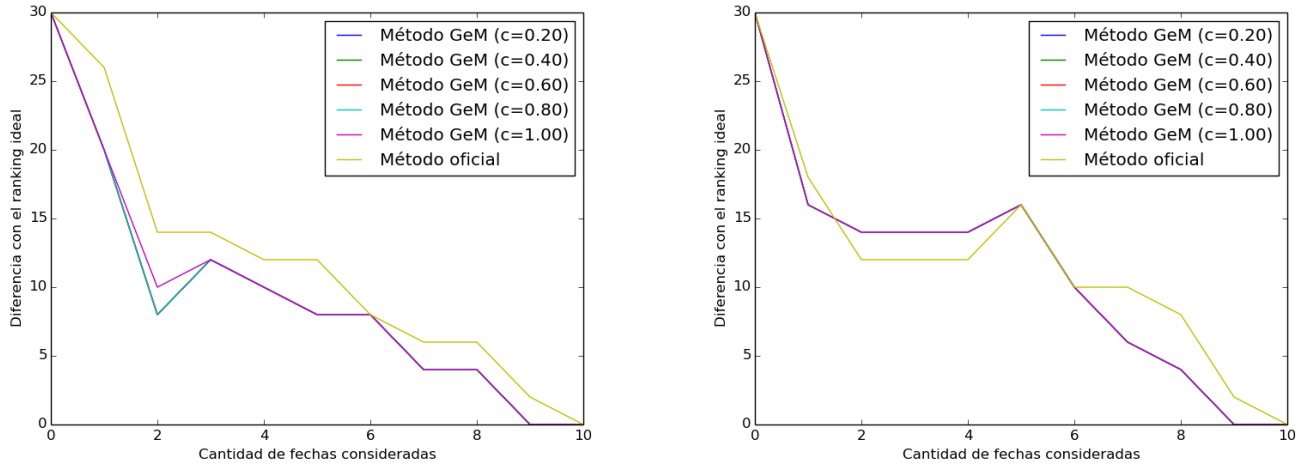
Objetivo Comparar GeM contra el raking estándar del fútbol en un caso ideal.

Hipótesis PageRank, utilizando el modelo GeM, converge más rápido al "Orden Real", si lo hay, que el sistema oficial del fútbol asociado establecido por la FIFA[**fifa**].

Proposición Asumiendo que se confirmó la hipótesis del experimento anterior, nos interesa analizar si, asumiendo que existe un orden ideal/correcto, GeM se comporta peor, igual o mejor que la forma estándar de ordenar a los equipos (por puntos, 3/1/0 puntos para victoria/empate/derrota) donde "comportarse" mejor significa que con la misma información disponible (por ejemplo, la mitad de los partidos jugados) se acerca más al orden ideal.

Método de Experimentación Consideramos las mismas instancias de prueba que el experimento anterior. Comparamos la distancia entre GeM y el orden ideal contra la distancia entre el ordenamiento estándar y el orden ideal, usando la misma definición de distancia de rankings que para el experimento anterior y el mismo criterio para ordenar equipos empatados en puntaje. De confirmarse nuestra hipótesis, deberíamos ver que GeM se acerca "más rápido", es decir, con menos fechas, al orden ideal.

Resultados, análisis y discusión



(a) Caso particular malo para α grande

(b) Caso particular malo para GeM

Figura 12. Casos Patológicos (10 equipos, $c = \alpha$)

Comparamos GeM con el orden oficial, establecido por asignación de puntos (3-1-0) y confirmamos también nuestra hipótesis de que en el caso promedio, GeM se comporta mejor que el orden oficial de asignación de puntajes, acercándose más al ideal para cualquier cantidad de fechas.

Sin embargo también encontramos semillas para las cuales esto no era real en algún punto (ver figura 12b) por lo que tampoco podemos generalizar esta conclusión.

Por último, dado que los dos casos "patológicos" fueron encontrados con la instancia de pocos equipos (10) y no logramos encontrar una semilla que los genere en el caso de muchos (50), podríamos argumentar que GeM se comporta "mejor" cuando la cantidad de equipos es grande. Queda fuera de los alcances de este trabajo confirmar más sistemáticamente esa hipótesis, pero es un buen trabajo futuro.

En este experimento se trabajó principalmente con instancias artificiales "de juguete". Las mismas lejos están de representar la realidad, pero nos sirvieron para poder asegurar que el modelo GeM "descubriría" el ranking justo (o real, como diría Platon) si este existiese. Aún así, observamos que para llegar a este ranking el modelo requerirá de mucha información de entrada, lo cual le quita quizás utilidad como predictor de resultados de torneos[a]. También observamos que en la progesión fecha a fecha, el parámetro α pareciera tener de muy poca a nula influencia sobre los rankings obtenidos.

a. Sin ánimo de menospreciar a GeM, si uno tiene los datos de 50 sobre 53 fechas, predecir con buena probabilidad de acierto el orden final no es necesariamente una tarea extremadamente difícil.

Por último, al comparar la ”velocidad de aproximación al ranking ideal” con el sistema de puntaje real, pudimos ver que si bien considerabamos no tan bueno a GeM para aproximarse con poca información al resultado final, se comporta mejor que el ranking oficial (para los casos ideales que fueron considerados), con lo cual este modelo quizás podría ser un primer paso hacia un predictor de resultados de torneos de fútbol profesional[a].

a. Y la dominación mundial.

4.2.2. GeM vs Ranking Oficial

Objetivo Comparar GeM con los rankings estándar en el mundo real.

Proposición En la vida real la emoción subyacente de los deportes radica, en parte, en la posibilidad de que cualquiera le gane a cualquiera. ¿Para qué practicar u observar/seguir un deporte si este no es el caso? Un ranking trata de dar un orden que determine que participante es mejor que otro entre los que forman parte de una competición. Pero la confección de este ranking puede tener que variar de acuerdo a la forma de la competición: no en todos los eventos todos los equipos juegan contra todos, no siempre todos juegan la misma cantidad de partidos ni la misma cantidad de veces entre sí. Así pues, dar un ranking se vuelve más complicado. Nuestra idea es tomar casos del mundo real y observar los resultados de GeM y ver cómo se comporta respecto de estas asimetrías inherentes al tipo de competición; mientras que lo comparamos contra el ranking oficial utilizado en cada caso.

Método de Experimentación Evaluaremos 3 instancias de deportes/ligas:

1. El campeonato de Primera División del fútbol argentino 2015, tomando hasta la fecha 23.
2. La Copa Mundial de la FIFA de 2014, realizada en Brasil.
3. La Copa Mundial de la FIFA de 1954, realizada en Suiza.

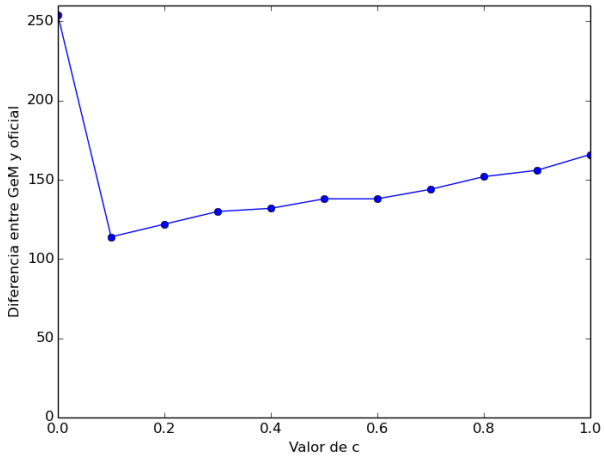
Tomamos el campeonato argentino como un ejemplo del formato de liga, y las Copas Mundiales de Fútbol como un ejemplo del caso en que no juegan todos contra todos ni todos juegan la misma cantidad de veces. Tomamos el caso de 2014 como representante del formato "actual" de 32 equipos con una única fase de grupos y una fase eliminatoria. Y tomamos el caso de 1954 por presentar una situación interesante de analizar: en la fase de grupos Hungría le ganó 8 a 3 al que luego saldría campeón, Alemania Federal.
Para los casos de los Mundiales, tomamos como "oficial" el ranking final publicado por la FIFA. Para el caso del campeonato de primera división, la ordenación estándar por puntaje 3-1-0.
En los tres casos ejecutamos el algoritmo GeM variando los valores de α y comparamos contra el ranking oficial de la instancia. En los tres casos decidimos ignorar los empates. Para las definiciones por tanda de penales, tomamos como resultado del partido el resultado final de los mismos.

Resultados, análisis y discusión

1. Lo primero que notamos es que para valores grandes de α la diferencia con el ranking oficial aumenta, alcanzando un mínimo en $\alpha = 0{,}1$. Consideramos que esto tiene que ver por darle demasiada importancia a la "transitividad de victorias" cuando el fútbol en general no funciona de esa manera. Por ejemplo, para $\alpha = 1$ los primeros cinco puestos son:

GeM		Oficial: 3-1-0	
Equipo	Puntaje	Equipo	Puntaje
Boca Juniors	0.0934262	San Lorenzo	50
River Plate	0.0828491	Boca Juniors	49
San Martín (SJ)	0.0674819	Racing Club	46
Aldosivi	0.0648027	Rosario Central	45
San Lorenzo	0.0596129	River Plate	44

(a) Primeros Puestos



(b) Diferencia en función de α

Figura 13. GeM vs Primera División Fútbol Argentino para $\alpha = 1$

La posición de San Martín de San Juan (13º según el puntaje oficial) en el ranking GeM se debe, en parte, a

que le ganó a San Lorenzo en la fecha 3, mientras que Aldosivi (24º según el puntaje oficial) logró lo mismo en la fecha 10 y le gano a San Martín de San Juan en la fecha 22. Removiendo esos partidos sus posiciones en GeM bajan significativamente.

De todos modos podemos observar que, para el valor de $\alpha = 0{,}1$, el ranking devuelto por GeM se parece un poco más al oficial, dándonos los 6 primeros puestos que se pueden observar en el cuadro 14a.

GeM		Oficial: 3-1-0	
Equipo	Puntaje	Equipo	Puntaje
River Plate	0.0383429	San Lorenzo	50
Boca Juniors	0.038337	Boca Juniors	49
San Lorenzo	0.0372973	Racing Club	46
Racing Club	0.0363868	Rosario Central	45
San Martín (SJ)	0.0352285	River Plate	44
Rosario Central	0.0347587	Independiente	38

(a) Primeros Puestos

GeM		Oficial: 3-1-0	
Equipo	Puntaje	Equipo	Puntaje
Olimpo	0.0316438	Godoy Cruz	22
Huracán	0.0315532	Huracán	21
Godoy Cruz	0.0315278	Atlético de Rafaela	20
Atlético de Rafaela	0.0309498	Arsenal	17
Colón	0.0308694	Nueva Chicago	14
Nueva Chicago	0.0307189	Crucero del Norte	14

(b) Últimos Puestos

Figura 14. GeM vs Primera División Fútbol Argentino para $\alpha = 0{,}1$

De estos 6 primeros puestos de GeM, 5 efectivamente corresponden a esos primeros 6 lugares según el oficial (el que falta es Independiente, que GeM ubica 15º) y uno de ellos está en la misma posición en ambos rankings.

Otra coincidencia notable son los últimos puestos, los cuales se pueden observar en el cuadro 14b. Esta tabla tiene 3 coincidencias débiles (mismos equipos en distinta posición) y una coincidencia exacta.

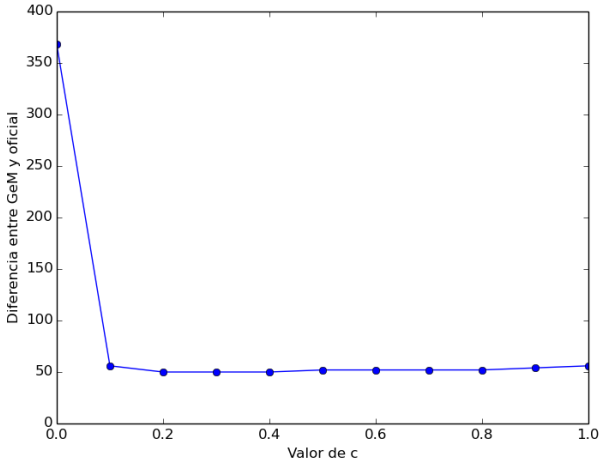
2. En este caso, la diferencia para valores no nulos de α es escasa, y la diferencia con el oficial en general es mucho mejor que para el caso del campeonato argentino. Consideramos esto relacionado al hecho de que hay pocos partidos en total, y a que la organización del torneo también se basa fuertemente en la transitividad de victorias (al menos en la fase final): en un Mundial, si A le ganó a B y B a C, se asume que A es mejor que C.

   Para todos los valores no nulos de α GeM ubicó correctamente como ganador a Alemania. Para $\alpha = 0{,}1$ considero que el subcampeón fue Países Bajos, lo cual sorprende dado que Argentina le ganó ”4 a 2” (el resultado de los penales). Evidentemente un valor tan bajo de α le da baja importancia a esto y pondera más las diferencias de goles de Países Bajos contra sus rivales (4 vs España, 2 vs Chile), mucho mejores que las de Argentina (que ganó todos sus otros partidos por un gol de diferencia). Para todos los demás valores de α, GeM identificó correctamente a Argentina como subcampeón.

   Las menores diferencias generales se obtuvieron para $\alpha = 0{,}2$, $\alpha = 0{,}3$ y $\alpha = 0{,}4$ ”empatadas” en una distancia de 50 con el ranking oficial de la FIFA. En estos casos sorprende la precisión de los resultados. Por ejemplo, observemos los primeros 8 lugares en la siguiente tabla, que coinciden exactamente con el ranking provisto por la FIFA:

Equipo	Puntaje
Alemania	0.0986858
Argentina	0.0764719
Países Bajos	0.0650904
Brasil	0.0480157
Colombia	0.0419815
Bélgica	0.0405001
Francia	0.0396461
Costa Rica	0.0344357

(a) Primeros puestos $\alpha = 0{,}4$



(b) Diferencia en Función de α

Figura 15. GeM vs Copa del Mundo 2014

Al consultarle su opinión al respecto de si fue o no penal, GeM guardó un respetuoso silencio.

3. Nuevamente observamos una buena aproximación entre GeM y el ranking oficial. Quizás los más sorprendente es que para todos los valores no nulos de α, GeM supo clasificar a Alemania Federal como el ganador del torneo a pesar de haber perdido por una diferencia de 5 goles en la fase de grupos con el subcampeón Hungría. La explicación que le encontramos a esto se basa en que la final se disputó precisamente entre esos dos equipos, y Alemania Federal se consagró ganador por 3 a 2. Esto produce que el grafo de conectividad tenga un ciclo entre esas dos selecciones, lo cual hace que parte del "puntaje"de Hungría vuelva a Alemania Federal. Todo eso sumado a la excelente campaña de este último en el campeonato (4 a 1 vs Turquía, 7 a 2 nuevamente vs Turquía, 2 a 0 vs Yugoslavia y 6 a 1 vs Austria, que venía de ganar varios partidos también por gran diferencia) lo posiciona efectivamente como ganador según GeM.

   También para todos los valores de α GeM acierta en ubicar a Hungría como subcampeón.

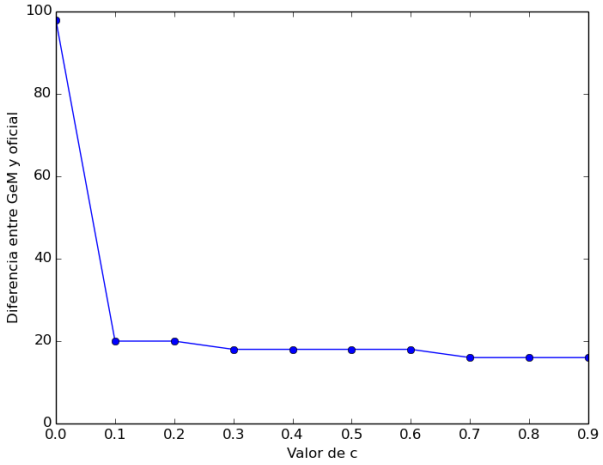
   La menor diferencia con el ranking oficial se obtiene con $\alpha = 0{,}8$ y $\alpha = 0{,}9$. Es preciso notar que no pudimos evaluar el caso $\alpha = 1$ por no converger para esta instancia, a diferencia de las dos competiciones anteriores[a].

   En estos dos casos, GeM acertó en los 5 primeros puestos de la competición, siendo estos:

a. Sabemos por la sección 1 que la convergencia está asegurada solo para $0 \leq \alpha < 1$, pero de todos modos venimos "ensayando" con $\alpha = 1$ sabiendo que dicho valor no tiene por qué hacer converger al método de la potencia para nuestro M

Equipo	Puntaje
Alemania Federal	0.421402
Hungría	0.409884
Austria	0.0299615
Uruguay	0.0252637
Suiza	0.0169375

(a) Primeros Puestos $\alpha = 0{,}4$



(b) Diferencia en Función de α

Figura 16. GeM vs Copa del Mundo 1954

Lo más jugoso de toda esta experimentación radica en que comparamos el modelo GeM de rankings contra todos rankings oficiales que están vigentes. El primer resultado interesante es observar que GeM está mucho más cerca para del ranking utilizado en las copas del mundo que del torneo actual del fútbol argentino, y esto se ve para cualquier $\alpha \neq 0$ (el caso con $\alpha = 0$, como ya se comentado en experimentos previos, concibe un modelo donde todos los equipos le pueden ganar a todos con la misma probabilidad, sin importar los resultados previos, haciendo a este parámetro poco interesante para analizar por su obvia ignorancia de la realidad[a]). Aún así, a pesar de esta distancia, vemos que en el caso del torneo de fútbol argentino, su ranking en los extremos de la "tabla" (particularmente en el extremo inferior) suele ubicar a los mismos equipos que el ranking oficial. Esto nos indica que a pesar de tener rankings muy distintos, en términos generales ambos rankings diferencian de forma similar a los equipos "buenos" y "malos".

Concluyendo, vemos que GeM puede o no parecerse a otros rankings oficiales, pero la determinación de si es mejor o peor dependerá de qué opinen quienes lo utilicen. Lo que si podemos afirmar es que GeM tiende a aproximarse más a los rankings oficiales (al menos los vistos) en los extremos del mismo. Es decir, a pesar de estar a una distancia no despreciable del ranking oficial, los equipos señalados como los más fuertes o mejores por ambos rankings (o los peores) tienen varias coincidencias. Decimos entonces que, desde el punto de vista de clasificar a un equipo como "de los buenos" o "de los malos" de la competencia, cualquiera de los rankings daría resultados similares, pero estos se diferencian al entrar en el detalle de qué equipo es mejor que otro.

a. ¿O acaso las chances de que Racing le gane a Crucero del Norte son las mismas de que pierda?

4.2.3. Estabilidad de GeM

Objetivo Observar el ranking de PageRank/GeM puede sufrir variaciones importantes de una fecha a la otra, al recibir un set nuevo de información.

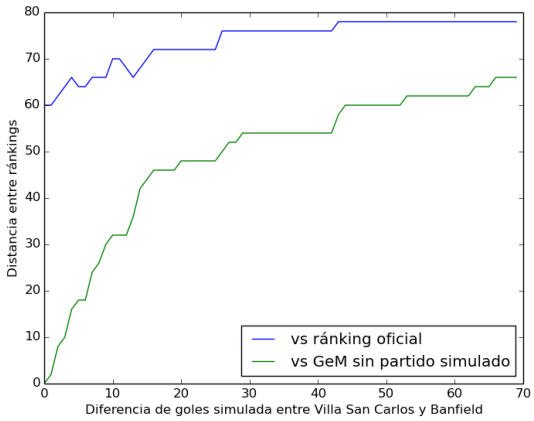
Hipótesis GeM es más inestable que la puntuación oficial del fútbol.

Proposición Nos interesa considerar casos extremos para evaluar la estabilidad del ranking que devuelve GeM. El sistema de puntaje actual del fútbol tiene la característica de ser bastante "estable": en una única fecha un equipo puede ganar a lo sumo 3 puntos, lo cual (salvo en los comienzos de un torneo o casos de empate múltiple difíciles de encontrar en la realidad) no lo hace avanzar más de 4 o 5 posiciones.
Nos interesa analizar si esta propiedad se conserva en GeM. Para eso, imaginemos un torneo de fútbol desbalanceado, es decir, un torneo en que al finalizar, el primer equipo tiene mucha diferencia con el último[a]. Si en una última fecha "inventada", agregada artificialmente, el último le ganase al primero, el método de puntuación estándar difícilmente altere demasiado el ranking. Queremos observar si esto ocurre con GeM.

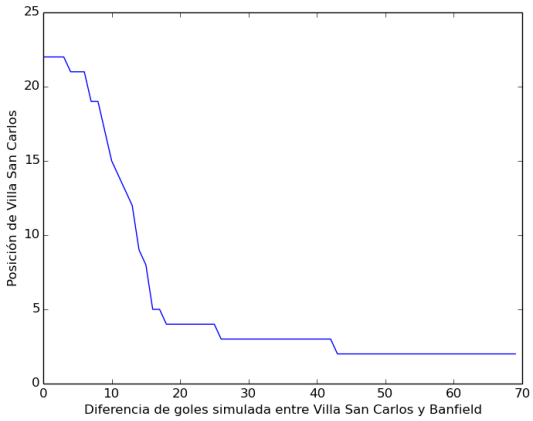
Método de Experimentación Tomamos el Campeonato de Primera B Nacional 2013/14, en el cual Banfield (1º) terminó con 78 puntos mientras que Villa San Carlos (22º y último) terminó con 24. Generamos una fecha artificial extra en la que Villa San Carlos le gana a Banfield. El ranking oficial no cambiaría, dado que con 27 puntos Villa San Carlos seguiría último. De confirmarse nuestra hipótesis, esperaríamos ver un cambio en la posición que GeM le asigna a Villa San Carlos. En este caso, variando la cantidad de goles, estudiamos cuánto se alteraría el resultado si la victoria fuese más abultada. El valor de α usado fue de 0.85.

Resultados, análisis y discusión

Se confirmó la hipótesis. Efectivamente GeM altera el ranking por una simple victoria por 1 a 0, aunque Villa San Carlos sigue último. Esto se debe a que el puntaje GeM de Banfield cae y el de VSC sube, alterando a los demás equipos que ganaron o perdieron contra alguno de ellos.



(a) Distancia GeM Adulterado vs Oficial/GeM



(b) Posición de Villa San Carlos en GeM Adulterado

Figura 17. Consecuencias de introducir un partido artificial

Alcanza con una victoria de 4 a 0 para que Villa San Carlos deje el último lugar. Y con una victoria de 43 a 0 pasa a estar en 2do lugar. Si esto parece irreal, considerar el récord del fútbol profesional de goles en un mismo partido: *AS Adema 149 - 0 SO l'Emyrne*, el 31 de octubre de 2002 por la *THB Champions League de Madagascar*[b].

En la figura 17b se puede observar la posición de Villa San Carlos en función de la cantidad de goles del partido artificial. En la figura 17a se puede observar la diferencia entre rankings producida por el partido simulado en función de la cantidad de goles del partido. Se observa claramente la "inestabilidad" de la que hablábamos: por

a. Consideramos esto desbalanceado. Si no es este el caso del lector, simplemente considerar una instancia que cumpla con esa condición.

b. En honor a la verdad, este resultado fue alcanzado por goles en contra por protesta contra el arbitraje. El resultado "honesto" más abultado fue *Arbroath FC 36 - 0 Bon Accord FC*, el 12 de septiembre de 1885 por la Copa Escocesa 1885-86 y, más recientemente, el conocido *Australia 31 - 0 Samoa Americana*, el 11 de abril de 2001, por la clasificación al Mundial de Fútbol Corea-Japón 2002

los resultados de un simple partido GeM puede alterar el orden en un factor grande. Consideramos que esta no es una propiedad deseable del sistema, al menos no para el fútbol: hay muchos factores que influyen en un resultado, independientemente de la calidad de los competidores (azar, arbitraje, clima, estado del campo, etc.). Un equipo "malo" no debería dejar de serlo solo por haber haber tenido suerte (o incluso por haber hecho un único buen partido) contra un equipo "bueno", o porque su rival haya hecho un partido malo.

El próximo experimento apunta a dejar aún más en evidencia esta situación.

4.2.4. Caso Particular GeM

Objetivo Analizar cuan "justo" es GeM, para un caso particular en el cual no haya dudas sobre lo que es justo y lo que no[a].

Proposición Nos interesa analizar cuán "justo" es GeM para cierta definición de justicia. Consideremos el caso de un torneo en que el equipo A le gana a todos los equipos salvo a B, y B pierde todos los partidos salvo el que le gana a A. Bajo nuestra definición de "justicia" o "equidad", o un aspecto de ella, A debería estar seguro por encima de B y B no debería estar por encima de muchos equipos (ya que perdió contra todos). Observamos que en el caso del fútbol y su ránking 3-1-0 (o el esquema antiguo, 2-1-0) efectivamente B estaría en la última posición y A estaría en la primera (eventualmente compartiendo dichas posiciones con algún otro equipo). Entendemos entonces que este caso particular el ranking 3-1-0 es "justo" en este aspecto. Pero intuimos que esto no será lo que ocurra con GeM, ya que en el grafo de la instancia, A tiene un único eje saliente (hacia B) y 18 entrantes, con lo cual su arista debería hacer subir mucho a B en el ranking.

Hipótesis PageRank/GeM no es "justo" en cuanto al aspecto mencionado.

Método de Experimentación Generamos una instancia de 20 equipos todos contra todos, donde existen A y B como se describieron. Entre los demás equipos hacemos que el ganador sea aleatorio (con semilla = 5). Todos los partidos terminan 1 a 0. Ejecutamos GeM y observamos el ranking final para diferentes valores de α (el factor de navegación).

Resultados, análisis y discusión

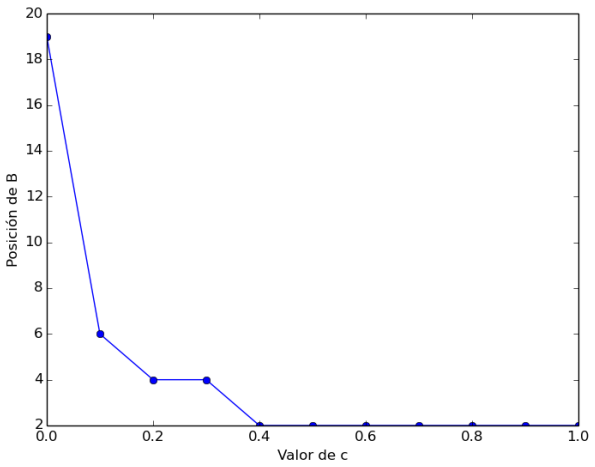


Figura 18. Posición del equipo B en el ranking en función del factor α ($c = \alpha$)

Lo primero que observamos es que B no ascendió al primer lugar para ningún valor de α, lo cual era esperable dados los resultados del experimento 4.2.3. Sin embargo, y como se puede ver en la figura 18, la posición del equipo B sí mejora significativamente ya para valores pequeños de α: un $\alpha = 0{,}1$ deja a B 6to en el ranking, y a partir de $\alpha = 0{,}4$ pasa a estar 2do. Consideramos entonces que se confirmó nuestra hipótesis en el sentido de que GeM no es "justo" en un caso del estilo. Como ya postulamos en el experimento 4.2.3, no parece "justo" que un equipo prácticamente invicto que tuvo un "mal día" le permita a otro equipo salir 2do en una competencia en la que perdió con cuanto rival se cruzó.

Y no sólo eso. Supongamos ahora que el equipo A entra en una mala racha y pierde muchos partidos consecutivos. Esto impactará negativamente (para los primeros partidos que pierda A luego de ser derrotado por B) en la ubicación en el ranking de B. Es decir, B se verá propulsado hacia las posiciones superiores del ranking, pero luego comenzará a perder posiciones no por "mérito" própio, sino por errores ajenos.

Podemos concluir entonces que los valores más "justos" de α, para evitar esta inestabilidad, serían los más bajos, dado que le dan menos importancia a estas situaciones extrañas pero perfectamente posibles en cualquier deporte.

Lo que observamos es que GeM a la hora de otorgar un puntaje se "fija" constantemente cuál es el nuevo puntaje de aquellos equipos que hacen variar el ranking de un equipo X, lo cual en un caso estático no pareciera estar mal, pero si lo pensamos en el tiempo avanzando fecha a fecha, podría ser injusto a los ojos de alguno[b]. Ejemplifiquemos esto último: supongamos que Racing, en zona de descenso, le gana al puntero del campeonato y a partir de la inestabilidad encontrada en GeM, pasa a estar fuera de la zona de descenso. Pero ahora depende de que el puntero siga ganando si no quiere volver a dicha zona. La mayoría de los rankings oficiales de los deportes evitan esto, ya

a. O que haya muy poca probabilidad de que haya dudas al respecto.
b. Ni hablar de las posibilidades de fraude. "Vos me ganaste, y si ahora yo salgo a perder te perjudico". *Imagine the possibilites*.

que equiparan los puntos que se reciben por ganarle a cualquier equipo (no hay diferencia en cuanto a ganarle al puntero que a un equipo de media tabla).

Para finalizar, diremos que si bien se encontró esta inestabilidad en el modelo GeM, si esto es perjudicial o no dependerá del contexto de uso. La utilización de este esquema para determinar el orden de los competidores en algún evento deportivo podría dar lugar a situaciónes mejores o peores. Como siempre, con esto y con casi todas las cosas, su aplicación, beneficios y desventajes dependerán del contexto de uso.

Experimentos a Futuro

Como experimentos futuros relacionados con estos temas, que por cuestiones de tiempo y alcance de este trabajo no fueron realizados, quedaría evaluar la *sensibilidad* de PageRank y su convergencia al orden más que al (auto)vector de distribución estacionario.

En primer caso, la sensibilidad para PageRank, trata de exponer si cambios pequeños en un grafo dado (algunos ejes menos, algunos ejes más) modifican radicalmente el orden obtenido. Y también experimentar sobre como α afecta a esta sensibilidad.

En cuanto al caso de la convergencia al orden más que al vector resultante, lo que queremos explicar es que en realidad lo que se busca obtener con PageRank es un ranking, más allá de los puntajes de cada página web. Quizás se podría observar si para toda instancia y α fijo, se puede estimar estadistícamente que luego de k iteraciones (que esperamos que sea un número, o incluso una función que tome ciertos parametros y devuelva un número, mucho menor que la cantidad de interaciones necesarias por el método para converger) ya se llegó al mismo orden al que se llegará al converger (aunque con distintos puntajes). Si esto se pudiera realizar, podríamos tener una primera heurística para mejorar el tiempo de convergencia, aunque claramente estaríamos trabajando ya con orden áun más aproximado que el que devuelve el método de la potencia implementado sobre aritmética finita.

5. CONCLUSIONES

A lo largo de este trabajo pudimos vislumbrar las complejidades propias del problema de ordenar una colección en principio desordenada de elementos. Tomamos el ejemplo de las páginas web por un lado y el de las competiciones deportivas por otro, enfocándonos particularmente en el fútbol. Usamos el algoritmo de PageRank y su adaptación GeM para resolver ambos problemas respectivamente, y estudiamos su comportamiento al variar diferentes parámetros de entrada, particularmente el factor de teletransportación α.

Respecto al contexto de páginas Web, como conclusión general acerca de los experimentos de performance pudimos observar que la variación del factor de navegacion α no altera el tiempo consumido por iteración del método de cálculo del ranking (método de la potencia). Otras condiciones sobre el grafo de entrada (ie. densidad) sí producen una variación en el tiempo de cómputo por iteración.
Sobre los experimentos acerca de la convergencia, pudimos observar que a medida que el factor de navegación crece -para un vector inicial equiprobable- la cantidad de iteraciones requeridas hasta cumplir la condición de corte del algoritmo se incrementan, o equivalentemente, la velocidad de convergencia decrece. Estos experimentos nos dieron la pauta de cuales condiciones o parámetros afectan la performance del método implementado y de qué manera.

Respecto a los experimentos cualitativos acerca del orden, corroboramos que para cualquier factor de navegación α dentro de un rango válido el ranking obtenido por PageRank no varía a pesar de cambiar el autovector resultante. Para poner a prueba las bondades cualitativas de PageRank contrastamos el ranking para estas instancias mencionadas anteriormente contra otros algoritmos de rankeo, entre ellos In-Deg y el ranking de Google mismo al dia de hoy. Pudimos comprobar que PageRank genera rankings más "interesantes" o útiles que In-Deg y que, por ejemplo, los términos más importantes de una lista de temas quedan en los primeros puestos del ranking.

Como conclusión general para el caso de las competencias deportivas pudimos concluir que, si bien el algoritmo se comporta "mejor" que la punutación oficial para casos de comportamiento "ideal" donde todos los resultados están determinados de entrada y las victorias son transitivas (una liga de lo más aburrida), presenta comportamientos que podemos considerar "injustos" o "indeseables" para cierta definición de dichas palabras. Por ejemplo, la posibilidad de que un único encuentro altere fuertemente el orden final, y mejore significativamente la posición de un equipo que en nuestra opinión no "merecía" el lugar que GeM le asignaba. O también la posibilidad de que una mala racha de un equipo afecte negativamente la posición de todos los que le ganaron a él, dando lugar a especulaciones de todo tipo. Para estos casos, encontramos que valores pequeños de α se acercan más a nuestra noción de "justicia", dado que representan una mayor probabilidad de que "cualquiera le gane a cualquiera" lo cual es cierto en todos los deportes. Y ocurre al revés con los casos ideales: el algoritmo (en promedio) se comporta mejor para valores grandes de α cuando las victorias sí son transitivas.

A pesar de estos problemas, encontramos que, para casos estables y sin alterar, el algoritmo GeM puede acercarse muy bien a los órdenes establecidos por los ránkings oficiales, sobretodo para los primeros y últimos lugares. Inclusive logra esto en ciertos casos en donde ha habido resultados "extraños", como que un subcampeón le haya ganado al campeón en una fecha anterior a la final. E independientemente de la exactitud de posiciones, GeM parecería hacer un buen trabajo a la hora de distinguir, mirándolo como conjuntos, a los "buenos" de los "malos", obteniendo coincidencias fuertes (módulo el orden específico establecido por cada uno) con los sistemas de puntuación oficial.

En última instancia, queda a criterio del lector decidir si los órdenes devueltos por GeM son "mejores" o "peores" que los resultados oficiales, un terreno en el que decidimos conscientemente no meternos por estar repleto de subjetividades[a], limitándonos a estudiar casos precisos donde estuviese más o menos claro qué orden era "correcto". En cualquier caso, siempre sobrevive la *mayor objeción a la aplicación de GeM en la vida real*: prácticamente nadie entendería por qué su equipo está donde está.

a. Si, según GeM, Argentina le ganaba a Alemania en el Mundial lo habríamos calificado como el mejor método del mundo

APÉNDICE A
ENUNCIADO DEL TRABAJO PRÁCTICO

Métodos Numéricos

Segundo Cuatrimestre 2015

Trabajo Práctico 2

Departamento de Computación

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ohhh solo tiran π-edras...

Contexto y motivación

A partir de la evolución de Internet durante la década de 1990, el desarrollo de motores de búsqueda se ha convertido en uno de los aspectos centrales para su efectiva utilización. Hoy en día, sitios como Yahoo, Google y Bing ofrecen distintas alternativas para realizar búsquedas complejas dentro de un red que contiene miles de millones de páginas web.

En sus comienzos, una de las características que distinguió a Google respecto de los motores de búsqueda de la época fue la calidad de los resultados obtenidos, mostrando al usuario páginas relevantes a la búsqueda realizada. El esquema general de los orígenes de este motor de búsqueda es brevemente explicado en Brin y Page [**Brin1998**], donde se mencionan aspectos técnicos que van desde la etapa de obtención de información de las páginas disponibles en la red, su almacenamiento e indexado y su posterior procesamiento, buscando ordenar cada página de acuerdo a su importancia relativa dentro de la red. El algoritmo utilizado para esta última etapa es denominado PageRank y es uno (no el único) de los criterios utilizados para ponderar la importancia de los resultados de una búsqueda. En este trabajo nos concentraremos en el estudio y desarrollo del algoritmo PageRank.

Por otro lado, las competencias deportivas, en todas sus variantes y disciplinas, requieren casi inevitablemente la comparación entre competidores mediante la confección de *Tablas de Posiciones* y *Rankings* en base a resultados obtenidos en un período de tiempo determinado. Estos ordenamientos de equipos están generalmente (aunque no siempre) basados en reglas relativamente claras y simples, como proporción de victorias sobre partidos jugados o el clásico sistema de puntajes por partidos ganados, empatados y perdidos. Sin embargo, estos métodos simples y conocidos por todos muchas veces no logran capturar la complejidad de la competencia y la comparación. Esto es particularmente evidente en ligas donde, por ejemplo, todos los equipos no juegan la misma cantidad de veces entre sí.

A modo de ejemplo, la NBA y NFL representan dos ligas con fixtures de temporadas regulares con estas características. Recientemente, el Torneo de Primera División de AFA se suma a este tipo de competencias, ya que la incorporación de la *Fecha de Clásicos* parece ser una interesante idea comercial, pero no tanto desde el punto de vista deportivo ya que cada equipo juega contra su *clásico* más veces que el resto. Como contraparte, éstos rankings son utilizados muchas veces como criterio de decisión, como por ejemplo para determinar la participación en alguna competencia de nivel internacional, con lo cual la confección de los mismos constituye un elemento sensible, afectando intereses deportivos y económicos de gran relevancia.

El problema, Parte I: PageRank y páginas web

El algoritmo PageRank se basa en la construcción del siguiente modelo. Supongamos que tenemos una red con n páginas web $Web = \{1, \ldots, n\}$ donde el objetivo es asignar a cada una de ellas un puntaje que determine la importancia relativa de la misma respecto de las demás. Para modelar las relaciones entre ellas, definimos la *matriz de conectividad* $W \in \{0,1\}^{n \times n}$ de forma tal que $w_{ij} = 1$ si la página j tiene un link a la página i, y $w_{ij} = 0$ en caso contrario. Además, ignoramos los *autolinks*, es decir, links de una página a sí misma, definiendo $w_{ii} = 0$. Tomando esta matriz, definimos el grado de la página j, n_j, como la cantidad de links salientes hacia otras páginas de la red, donde $n_j = \sum_{i=1}^{n} w_{ij}$. Además, notamos con x_j al puntaje asignado a la página $j \in Web$, que es lo que buscamos calcular.

La importancia de una página puede ser modelada de diferentes formas. Un link de la página $u \in Web$ a la página $v \in Web$ puede ser visto como que v es una página importante. Sin embargo, no queremos que una página obtenga mayor importancia simplemente porque es apuntada desde muchas páginas. Una forma de limitar esto es ponderar los links utilizando la importancia de la página de origen. En otras palabras, pocos links de páginas

importantes pueden valer más que muchos links de páginas poco importantes. En particular, consideramos que la importancia de la página v obtenida mediante el link de la página u es proporcional a la importancia de la página u e inversamente proporcional al grado de u. Si la página u contiene n_u links, uno de los cuales apunta a la página v, entonces el aporte de ese link a la página v será x_u/n_u. Luego, sea $L_k \subseteq Web$ el conjunto de páginas que tienen un link a la página k. Para cada página pedimos que

$$x_k = \sum_{j \in L_k} \frac{x_j}{n_j}, \qquad k = 1, \ldots, n. \tag{32}$$

Definimos $P \in \mathbb{R}^{n \times n}$ tal que $p_{ij} = 1/n_j$ si $w_{ij} = 1$, y $p_{ij} = 0$ en caso contrario. Luego, el modelo planteado en (32) es equivalente a encontrar un $x \in \mathbb{R}^n$ tal que $Px = x$, es decir, encontrar (suponiendo que existe) un autovector asociado al autovalor 1 de una matriz cuadrada, tal que $x_i \geq 0$ y $\sum_{i=1}^{n} x_i = 1$. En Bryan y Leise [**Bryan2006**] y Kamvar et al. [**Kamvar2003**] se analizan ciertas condiciones que debe cumplir la red de páginas para garantizar la existencia de este autovector.

Una interpretación equivalente para el problema es considerar al *navegante aleatorio*. Éste empieza en una página cualquiera del conjunto, y luego en cada página j que visita sigue navegando a través de sus links, eligiendo el mismo con probabilidad $1/n_j$. Una situación particular se da cuando la página no tiene links salientes. En ese caso, consideramos que el navegante aleatorio pasa a cualquiera de las página de la red con probabilidad $1/n$. Para representar esta situación, definimos $v \in \mathbb{R}^{n \times n}$, con $v_i = 1/n$ y $d \in \{0, 1\}^n$ donde $d_i = 1$ si $n_i = 0$, y $d_i = 0$ en caso contrario. La nueva matriz de transición es

$$\begin{aligned} D &= vd^t \\ P_1 &= P + D. \end{aligned}$$

Además, consideraremos el caso de que el navegante aleatorio, dado que se encuentra en la página j, decida visitar una página cualquiera del conjunto, independientemente de si esta se encuentra o no referenciada por j (fenómeno conocido como *teletransportación*). Para ello, consideramos que esta decisión se toma con una probabilidad $c \geq 0$, y podemos incluirlo al modelo de la siguiente forma:

$$\begin{aligned} E &= v\bar{1}^t \\ P_2 &= cP_1 + (1 - c)E, \end{aligned}$$

donde $\bar{1} \in \mathbb{R}^n$ es un vector tal que todas sus componentes valen 1. La matriz resultante P_2 corresponde a un enriquecimiento del modelo formulado en (32). Probabilísticamente, la componente x_j del vector solución (normalizado) del sistema $P_2 x = x$ representa la proporción del tiempo que, en el largo plazo, el navegante aleatorio pasa en la página $j \in Web$. Denotaremos con π al vector solución de la ecuación $P_2 x = x$, que es comúnmente denominado *estado estacionario*.

En particular, P_2 corresponde a una matriz *estocástica por columnas* que cumple las hipótesis planteadas en Bryan y Leise [**Bryan2006**] y Kamvar et al. [**Kamvar2003**], tal que P_2 tiene un autovector asociado al autovalor 1, los demás autovalores de la matriz cumplen $1 = \lambda_1 > |\lambda_2| \geq \cdots \geq |\lambda_n|$ y, además, la dimensión del autoespacio asociado al autovalor λ_1 es 1. Luego, π puede ser calculada de forma estándar utilizando el método de la potencia.

Una vez calculado el ranking, se retorna al usuario las t páginas con mayor puntaje.

El problema, Parte II: PageRank y ligas deportivas

Existen en la literatura distintos enfoques para abordar el problema de determinar el *ranking* de equipos de una competencia en base a los resultados de un conjunto de partidos. En Govan et al. [**Govan2008**] se hace una breve reseña de dos ellos, y los autores proponen un nuevo método basado en el algoritmo PageRank que denominan GeM[a]. Conceptualmente, el método GeM representa la temporada como un red (grafo) donde las páginas web representan a los equipos, y existe un link (que tiene un valor, llamado peso, asociado) entre dos equipos que los relaciona modelando los resultados de los posibles enfrentamientos entre ellos. En base a este modelo, Govan et al. [**Govan2008**] proponen calcular el ranking de la misma forma que en el caso de las páginas web.

En su versión básica, que es la que consideraremos en el presente trabajo, el método GeM (ver, e.g., [**Govan2008**]) es el siguiente[b]:

a. Aunque no se especifica, asumimos que el nombre se debe a las iniciales de los autores.

b. Notar que en artículo, Govan et al. [**Govan2008**] lo definen sobre la traspuesta. La definición y las cuentas son equivalentes, simplemente se modifica para mantener la consistencia a lo largo del enunciado.

1. La temporada se representa mediante un grafo donde cada equipo representa un nodo y existe un link de i a j si el equipo i perdió al menos una vez con el equipo j.
2. Se define la matriz $A^t \in \mathbb{R}^{n \times n}$

$$A^t_{ji} = \begin{cases} w_{ji} & \text{si el equipo } i \text{ perdió con el equipo } j, \\ 0 & \text{en caso contrario,} \end{cases}$$

   donde w_{ji} es la diferencia absoluta en el marcador. En caso de que i pierda más de una vez con j, w_{ji} representa la suma acumulada de diferencias. Notar que A^t es una generalización de la matriz de conectividad W definida en la sección anterior.
3. Definir la matriz $H^t_{ji} \in \mathbb{R}^{n \times n}$ como

$$H^t_{ji} = \begin{cases} A^t_{ji} / \sum_{k=1}^{n} A^t_{ki} & \text{si hay un link } i \text{ a } j, \\ 0 & \text{en caso contrario.} \end{cases}$$

4. Tomar $P = H^t$, y aplicar el método PageRank como fue definido previamente, siendo π la solución a la ecuación $P_2 x = x$. Notar que los páginas sin links salientes, en este contexto se corresponden con aquellos equipos que se encuentran invictos.
5. Utilizar los puntajes obtenidos en π para ordenar los equipos.

En función del contexto planteado previamente, el método GeM define una estructura que relaciona equipos dependiendo de los resultados parciales y obtener un ranking utilizando solamente esta información.

Enunciado

El objetivo del trabajo es experimentar en el contexto planteado utilizando el algoritmo PageRank con las variantes propuestas. A su vez, se busca comparar los resultados obtenidos cualitativa y cuantitativamente con los algoritmos tradicionales utilizados en cada uno de los contextos planteados. Los métodos a implementar (como mínimo) en ambos contexto planteados por el trabajo son los siguientes:

1. *Búsqueda de páginas web:* PageRank e IN-DEG, éste último consiste en definir el ranking de las páginas utilizando solamente la cantidad de ejes entrantes a cada una de ellas, ordenándolos en forma decreciente.
2. *Rankings en competencias deportivas:* GeM y al menos un método estándar propuesto por el grupo (ordenar por victorias/derrotas, puntaje por ganado/empatado/perdido, etc.) en función del deporte(s) considerado(s).

El contexto considerado en 1., en la búsqueda de páginas web, representa un desafío no sólo desde el modelado, si no también desde el punto de vista computacional considerando la dimensión de la información y los datos a procesar. Luego, dentro de nuestras posibilidades, consideramos un entorno que simule el contexto real de aplicación donde se abordan instancias de gran escala (es decir, n, el número total de páginas, es grande). Para el desarrollo de PageRank, se pide entonces considerar el trabajo de Bryan y Leise [**Bryan2006**] donde se explica la intución y algunos detalles técnicos respecto a PageRank. Además, en Kamvar et al. [**Kamvar2003**] se propone una mejora del mismo. Si bien esta mejora queda fuera de los alcances del trabajo, en la Sección 1 se presenta una buena formulación del algoritmo. En base a su definición, P_2 no es una matriz esparsa. Sin embargo, en Kamvar et al. [**Kamvar2003**] se propone una forma alternativa para computar $x^{(k+1)} = P_2 x^{(k)}$. Este resultado debe ser utilizado para mejorar el almacenamiento de los datos.

En la práctica, el grafo que representa la red de páginas suele ser esparso, es decir, una página posee relativamente pocos links de salida comparada con el número total de páginas. A su vez, dado que n tiende a ser un número muy grande, es importante tener en cuenta este hecho a la hora de definir las estructuras de datos a utilizar. Luego, desde el punto de vista de implementación se pide utilizar alguna de las siguientes estructuras de datos para la representación de las matrices esparsas: *Dictionary of Keys* (dok), *Compressed Sparse Row* (CSR) o *Compressed Sparse Column* (CSC) Se deberá incluir una justificación respecto a la elección que consdiere el contexto de aplicación. Además, para PageRank se debe implementar el método de la potencia para calcular el autovector principal. Esta implementación debe ser realizada íntegramente en C++.

En función de la experimentación, se deberá realizar un estudio particular para cada algoritmo (tanto en términos de comportamiento del mismo, como una evaluación de los resultados obtenidos) y luego se procederá a comparar cualitativamente los rankings generados. La experimentación deberá incluir como mínimo los siguientes experimentos:

1. Estudiar la convergencia de PageRank, analizando la evolución de la norma Manhattan (norma L_1) entre dos iteraciones sucesivas. Comparar los resultados obtenidos para al menos dos instancias de tamaño mediano-grande, variando el valor de c.
2. Estudiar el tiempo de cómputo requerido por PageRank.
3. Para cada algoritmo, proponer ejemplos de tamaño pequeño que ilustren el comportamiento esperado (puede ser utilizando las herramientas provistas por la cátedra o bien generadas por el grupo).

Puntos opcionales:

1. Demostrar que los pasos del Algoritmo 1 propuesto en Kamvar et al. [**Kamvar2003**] son correctos y computan P_2x.
2. Establecer una relación con la proporción entre $\lambda_1 = 1$ y $|\lambda_2|$ para la convergencia de PageRank.

El segundo contexto de aplicación no presenta mayores desafíos desde la perspectiva computacional, ya que en el peor de los casos una liga no suele tener mas que unas pocas decenas de equipos. Más aún, es de esperar que en general la matriz que se obtiene no sea esparsa, ya que probablemente un equipo juegue contra un número significativo de contrincantes. Sin embargo, la popularidad y sensibilidad del problema planteado requieren de un estudio detallado y pormenorizado de la calidad de los resultados obtenidos. El objetivo en este segundo caso de estudio es puramente experimental.

En función de la implementación, aún cuando no represente la mejor opción, es posible reutilizar y adaptar el desarrollo realizado para páginas web. También es posible realizar una nueva implementación desde cero, simplificando la operatoria y las estructuras, en C++, MATLAB o PYTHON.

La experimentación debe ser realizada con cuidado, analizando (y, eventualmente, modificando) el modelo de GeM:

1. Considerar al menos un conjunto de datos reales, con los resultados de cada fecha para alguna liga de alguún deporte.
2. Notar que el método GeM asume que no se producen empates entre los equipos (o que si se producen, son poco frecuentes). En caso de considerar un deporte donde el empate se da con cierta frecuencia no despreciable (por ejemplo, fútbol), es fundamental aclarar como se refleja esto en el modelo y analizar su eventual impacto.
3. Realizar experimentos variando el parámetro c, indicando como impacta en los resultados. Analizar la evolución del ranking de los equipos a través del tiempo, evaluando también la evolución de los rankings e identificar características/hechos particulares que puedan ser determinantes para el modelo, si es que existe alguno.
4. Comparar los resultados obtenidos con los reales de la liga utilizando el sistema estándar para la misma.

Puntos opcionales:

1. Proponer (al menos) dos formas alternativas de modelar el empate entre equipos en GeM.

Parámetros y formato de archivos

El programa deberá tomar por línea de comandos dos parámetros. El primero de ellos contendrá la información del experimento, incluyendo el método a ejecutar (`alg`, 0 para PageRank, 1 para el método alternativo), la probabilidad de teletransportación c, el tipo de instancia (0 páginas web, 1 deportes), el *path* al archivo/directorio conteniendo la definición de la red (que debe ser relativa al ejecutable, o el path absoluto al archivo) y el valor de tolerancia utilizado en el criterio de parada del método de la potencia.

El siguiente ejemplo muestra un caso donde se pide ejecutar PageRank, con una probabilidad de teletransportación de 0.85, sobre la red descripta en `test1.txt` (que se encuentra en el directorio `tests/`), correspondiente a una instancia de ranking aplicado a deportes y con una tolerancia de corte de $0{,}0001$.

```
0 0.85 1 tests/red-1.txt 0.0001
```

Para la definición del grafo que representa la red, se consideran dos bases de datos de instancias con sus correspondientes formatos. La primera de ellas es el conjunto provisto en SNAP [**SNAP**] (el tipo de instancia es 0), con redes de tamaño grande obtenidos a partir de datos reales. Además, se consideran las instancias que se forman a partir de resultados de partidos entre equipos, para algún deporte elegido por el grupo.

En el caso de la base de SNAP, los archivos contiene primero cuatro líneas con información sobre la instancia (entre ellas, n y la cantidad total de links, m) y luego m líneas con los pares i, j indicando que i apunta a j. A modo de ejemplo, a continuación se muestra el archivo de entrada correspondiente a la red propuesta en Bryan y Leise [**Bryan2006**]:

```
# Directed graph (each unordered pair of nodes is saved once):
# Example shown in Bryan and Leise.
# Nodes: 4 Edges: 8
# FromNodeId    ToNodeId
1   2
1   3
1   4
2   3
2   4
3   1
4   1
4   3
```

Para el caso de rankings en ligas deportivas, el archivo contiene primero una línea con información sobre la cantidad de equipos (n), y la cantidad de partidos totales a considerar (k). Luego, siguen k lńeas donde cada una de ellas representa un partido y contiene la siguiente información: número de fecha (es un dato opcional al problema, pero que puede ayudar a la hora de experimentar), equipo i, goles equipo i, equipo j, goles equipo j. A continuación se muestra el archivo de entrada con la información del ejemplo utilizado en Govan et al. [**Govan2008**]:

```
6 10
1 1 16 4 13
1 2 38 5 17
1 2 28 6 23
1 3 34 1 21
1 3 23 4 10
1 4 31 1 6
1 5 33 6 25
1 5 38 4 23
1 6 27 2 6
1 6 20 5 12
```

Es importante destacar que, en este último caso, los equipos son identificados mediante un número. Opcionalmente podrá considerarse un archivo que contenga, para cada equipo, cuál es el código con el que se lo identifica.

Una vez ejecutado el algoritmo, el programa deberá generar un archivo de salida que contenga una línea por cada página (n líneas en total), acompañada del puntaje obtenido por el algoritmo PageRank/IN-DEG/método alternativo.

Para generar instancias de páginas web, es posible utilizar el código Python provisto por la cátedra. La utilización del mismo se encuentra descripta en el archivo README. Es importante mencionar que, para que el mismo funcione, es necesario tener acceso a Internet. En caso de encontrar un bug en el mismo, por favor contactar a los docentes de la materia a través de la lista. Desde ya, el código puede ser modificado por los respectivos grupos agregando todas aquellas funcionalidades que consideren necesarias.

Para instancias correspondientes a resultados entre equipos, la cátedra provee un conjunto de archivos con los resultados del Torneo de Primera División del Fútbol Argentino hasta la Fecha 23. Es importante aclarar que los dos partidos suspendidos, River - Defensa y Justicia y Racing - Godoy Cruz han sido arbitrariamente completados con un resultado inventado, para simplificar la instancia. En función de datos reales, una alternativa es considerar el repositorio DataHub [**datahub**], que contiene información estadística y resultados para distintas ligas y deportes de todo el mundo.

Fechas de entrega

- *Formato Electrónico:* Martes 6 de Octubre de 2015, hasta las 23:59 hs, enviando el trabajo (informe + código) a la dirección `metnum.lab@gmail.com`. El subject del email debe comenzar con el texto `[TP2]` seguido de la lista de apellidos de los integrantes del grupo.
- *Formato físico:* Miércoles 7 de Octubre de 2015, a las 18 hs. en la clase práctica.

Importante: El horario es estricto. Los correos recibidos después de la hora indicada serán considerados re-entrega.

APÉNDICE B
CÓDIGO FUENTE RELEVANTE

Interfaz Matriz CSR

```
class CSR
{

        /************************ GETTERS ************************/
        uint rows() const;
        uint cols() const;
        void get_row(uint fila, std::vector<T>& elementos, std::vector<uint>& columnas) const;

        /**
        * Operador ()
        * Permite acceder y asignar al elemento i, j de la matriz usando la notacion A(i, j)
        */
        T operator()(uint fila, uint columna) const;

        /************************ METODOS ************************/

        void prod_Ax(const std::vector<T>& x,
                     std::vector<T>& y/*result*/,
                     double parametro_c) const;

        void power_method(const std::vector<T>&, double _initial_vector,
                          power_method_stop_criteria_t,
                          std::vector<T>& _output) const;

        /************************ OUTPUT ************************/
        /**
        * Salida
        * Imprime la matriz completa en un stream de salida.
        */
        void print_sparse(std::ostream& os) const;

        /************************ CONSTRUCTORES ************************/
        CSR(DoK<T>& dok);
};
```

Interfaz Matriz DOK

```
class DoK
{
        /******************* OPERATORS y METODOS ******************/

        sparse_vector<T>& operator[] (uint pos);
        const sparse_vector<T>& operator[] (uint pos) const;

        uint cols();
        uint rows();

        /************************ OUTPUT ************************/
        /**
        * Salida
        * Imprime la matriz completa en un stream de salida.
        */
        void print_sparse(std::ostream& os) ;

        /********************* CONSTRUCTORES ********************/
        DoK() : matrix() {};
        DoK(uint r) : matrix(r),_rows(r) {};
        DoK(uint r,uint c) : matrix(r),_rows(r),_cols(c) {};
};
```

Implementacion Método de la potencia

```
// Nota: este metodo esta recortado para ser mas legible,
// para ver su implementacion completa, ir al codigo fuente
void CSR<T>::power_method(const std::vector<T>& _initial_vector,
                          double parametro_c,
                          power_method_stop_criteria_t criterio_parada,
                          std::vector<T>& _output_vector) const
{
    std::vector<T> eigenvec_candidate(_initial_vector);
```

```
    std::vector<T> new_eigenvec_candidate(eigenvec_candidate.size());
    double diff = 0.0;
    double epsilon_diff_corte = criterio_parada.valor.delta_diff;
    do {

        prod_Ax(eigenvec_candidate, new_eigenvec_candidate, parametro_c); //Ax

        diff = norma1(new_eigenvec_candidate-eigenvec_candidate,false);

        double norma_autovec = norma1(new_eigenvec_candidate, true);
        eigenvec_candidate = new_eigenvec_candidate/norma_autovec; // Reemplazo para proxima iteracion
             normalizado
        //eigenvec_candidate = new_eigenvec_candidate; // Reemplazo para proxima iteracion

    } while(diff >= epsilon_diff_corte);

    //Escribo la salida en el parametro de salida
    _output_vector = new_eigenvec_candidate;
};
```

Implementacion producto matriz por vector

```
// Impl. algoritmo 1 de golub Ax
// Nota: utilizamos operaciones sobre vectores sobrecargadas,
// ie. Producto vector por escalar, etc
void CSR<T>::prod_Ax(const std::vector<T>& x,
                     std::vector<T>& y/*resultado*/,
                     double parametro_c) const
{
    // Itero sobre las filas de la matriz
    for (uint idx_fila = 0; idx_fila < _numfilas; idx_fila++)
    {
        std::vector<T> fila_actual_elementos;
        std::vector<uint> fila_actual_columnas_llenas;
        get_row(idx_fila, fila_actual_elementos, fila_actual_columnas_llenas);

        // Hago el producto interno <fila_i, y> * x
        y[idx_fila] = 0;
        if(!fila_actual_elementos.empty())
        {
            for(uint idx_elem = 0;
                idx_elem < fila_actual_elementos.size();
                ++idx_elem)
            {
                y[idx_fila] +=
                        (fila_actual_elementos[idx_elem] *
                                x[fila_actual_columnas_llenas[idx_elem]]);
            };

            y[idx_fila] *= parametro_c;
        };
    };

    // Tenemos y = c* P^t * x

    double w = norma1(x, true) - norma1(y, true);//Podemos asumir que x e y
                                                 //tienen todas componentes >=0
                                                 //y nos ahorramos el abs en la
                                                 //norma !
    w/=(double)_numcolumnas;

    y+=w;
};
```