DETECCIÓN DE *LINK SPAM* USANDO *CLUSTERING* ESPECTRAL SOBRE CADENAS DE MARKOV

JOSÉ GÓMER DAVID GONZÁLEZ HERNÁNDEZ

Tesis presentada como requisito parcial para obtener el título de
MAGISTER EN INGENIERÍA
INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
Director:
GERMÁN HERNÁNDEZ, Ph. D.

UNIVERSIDAD NACIONAL DE COLOMBIA
FACULTAD DE INGENIERÍA
BOGOTÁ D. C.
2007

Resumen

Tabla de contenido

Lista de figuras

Lista de tablas

Capítulo 1

Introducción

Cuando se realiza una consulta en un motor de búsqueda, éste debe determinar cuáles son las páginas relevantes al tema de interés del usuario y listarlas en un orden de importancia de mayor a menor; de esta manera, los primeros resultados que se presentan deben coincidir con los documentos pertinentes de más de alta calidad. La medida de importancia utilizada debe coincidir en lo posible con exactitud a la idea subjetiva de importancia que tiene la gente, lo cual es una tarea complicada.

El problema del cómputo de la importancia tiene una analogía directa en la literatura académica, en la que es de interés identificar el conjunto de trabajos más influyentes en un área específica del conocimiento. En este contexto se ha aplicado exitosamente el análisis de las citaciones [30]: un documento se considera importante si recibe muchas citaciones (de personas diferentes a su autor), o inclusive si tiene algunas citaciones y estas son provenientes de documentos previamente identificados como relevantes (ya que las fuentes más valederas nunca citarían un documento de baja calidad).

Llevando estos principios al ámbito de la *web*, en la que los documentos corresponderían a páginas y las citas a enlaces entre ellas, el cálculo de importancia requiere hacer un análisis de conectividad entre los documentos. Las técnicas usadas para hacer este tipo de análisis y extraer los valores de importancia son conocidos en términos de los motores de búsqueda como "algoritmos de análisis de estructura de vínculos".

Los algoritmos de análisis de vínculos tienen un rol crucial en la visibilidad que tiene un negocio en Internet: aparecer listado entre los primeros puestos de un motor de búsqueda de alto uso en consultas realizadas frecuentemente por los usuarios incrementa ostensiblemente el número de visitas que un sitio dado recibe, lo cual constituye una ventaja competitiva. En sus intentos por obtener altos niveles de importancia, un creador de un sitio *web* puede lograr manipular o engañar un algoritmo de análisis de estructura de vínculos a su favor. Esta situación comúnmente se conoce como *link spam*. Dicha práctica tiene tres consecuencias:

- La competencia está siendo dejada atrás de forma injusta.
- La calidad de los motores de búsqueda disminuye en el sentido que los mejores contenidos relacionados con el tema de interés no están siendo incluídos en el lugar que deberían, perjudicando así a los usuarios.
- El motor de búsqueda desperdicia recursos de cómputo (espacio de almacenamiento y tiempo de proceso) ya que para aplicar técnicas como lo son encontrar los documentos pertinentes a un tópico particular o computar los valores de importancia mismos, se están incluyendo documentos irrelevantes que no aportan valor alguno.

En este trabajo se presenta un algoritmo que permite detectar el *link spam* que afecta a PageRank [36], el algoritmo de análisis de vínculos utilizado por el motor de búsqueda Google [2]. Aquí se muestra que el *spam* al que PageRank es vulnerable se puede caracterizar identificando las regiones

de baja conductancia de una cadena de Markov reversible definida a partir del digrafo inducido por la *web* (un grafo en el que las páginas son representadas como nodos y los hipervínculos como arcos dirigidos). Dichas regiones son conjuntos de estados de la cadena de Markov en los que un camino aleatorio tiene alta probabilidad de estancarse.

El hecho fundamental que permite el desarrollo del algoritmo es que se muestra que las regiones de baja conductancia pueden ser halladas a través del algoritmo de *clustering* espectral [33] aplicado sobre un grafo no dirigido que se define a partir de la cadena de Markov. Hasta donde se conoce, ésta es la primera propuesta que aplica algún tipo de análisis espectral con el objetivo de estudiar el fenómeno de *link spam*, constituyendo posiblemente un nuevo camino de investigación al respecto.

El algoritmo propuesto fue implementado y probado con porciones de la *web* de diferentes tamaños (hasta un millón de páginas) con el fin de evaluar el papel práctico que juega la conductancia en el problema de la detección automática de *link spam* y analizar la aplicabilidad del algoritmo a un nivel más realista, esto es, a un volumen de datos comparable con el que maneja un motor de búsqueda contemporáneo.

Este documento se estructura de la siguiente manera: El capítulo uno introduce conceptos básicos acerca de motores de búsqueda, entre ellos, el de análisis de estructura de vínculos y se detallan algunos de los algoritmos relevantes. El capítulo dos explica cómo se pueden engañar los algoritmos vistos y los estudios que se han publicado acerca de cómo combatir o detectar el fenómeno[1]. El capítulo tres trata el algoritmo de *clustering* espectral en general (aplicado a un grafo no dirigido arbitrario). En el capítulo cuatro se introduce el concepto de conductancia y se establece la analogía entre el algoritmo espectral y el de detección de regiones de baja conductancia; así mismo se discuten detalles del diseño de dicho algoritmo y se analiza su complejidad asintótica. En el capítulo quinto se dan detalles acerca de la implementación realizada, los conjuntos de datos utilizados en la experimentación, los resultados obtenidos y su respectivo análisis. Por último, el capítulo sexto presenta las conclusiones del trabajo.

[1] Dicha exploración bibliográfica contiene trabajos que fueron publicados hasta mediados del año 2006 únicamente.

Capítulo 2

Análisis de la estructura de vínculos de la *web*

2.1. Lista de *rankings*

Un motor de búsqueda se compone básicamente de cuatro partes: un colector de documentos *web*, un indexador, una lista de *rankings* y un manejador de consultas.

El colector de documentos[1] recupera páginas *web* y otro tipo de archivos que se encuentran en la *web* y las aloja en sistemas de almacenamiento físico. Este colector es un agente automatizado que explora el digrafo de la *web* usando por ejemplo, un recorrido en anchura para visitar los diferentes nodos que la componen.

El indexador se encarga de construir un índice invertido a través de un análisis de contenido sobre cada documento. Esto significa que el indexador debe extraer características textuales de los archivos que ha almacenado el colector. Un ejemplo típico de las características que se extraen de una página *web* son el contenido del *tag title*, del *tag keywords*, las palabras más frecuentes en el documento, las palabras cuyo tamaño de fuente sea mayor al del promedio de todo el documento, etc.

La lista de *rankings* determina un orden sobre las páginas colectadas de acuerdo a una medida global de popularidad[2] independiente de tópico, es decir, es una lista con un puntaje para cada página que representa cuán importante es con respecto a toda la *web*.

El manejador de consultas se encarga de responder las preguntas que los usuarios finales hacen a través de la interfaz del motor de búsqueda. Cuando el usuario envia su consulta, éste realiza un chequeo sobre el índice para determinar el conjunto de documentos que son relevantes al tópico de interés del usuario. Una vez este conjunto es construído, los items son presentados al usuario de acuerdo al orden definido por la lista de *rankings* ya precomputada: el primero con el puntaje general más alto, el siguiente con el segundo puntaje mayor, etc.

En general la lista de *rankings* se computa de manera periódica (cada vez que el colector hace una exploración de la *web*), sin embargo, algunos motores de búsqueda construyen la lista después de haber determinado el conjunto de páginas relevantes, en cuyo caso, sólo se computan puntajes para las páginas dentro de este conjunto.

El modo en que se construye ésta lista está relacionado con la importancia relativa entre páginas, sin embargo, este concepto es subjetivo y depende de los intereses, el conocimiento y las actitudes de cada usuario de modo que capturar de manera automatizada el interés que las personas prestan a cada uno de los documentos de la *web* parece ser una tarea muy compleja. De esta tarea se encargan los algoritmos de análisis de vínculos.

[1] También llamado "*crawler*", "*spider*" o "*robot*".
[2] Usualmente llamada importancia en la literatura.

2.2. Algoritmos de *ranking*

Los algoritmos de análisis de estructura de vínculos, también llamados algoritmos de *ranking* se usan para construir la lista de puntajes de popularidad. Su nombre se debe al hecho de que explotan la existencia de los hipervínculos en la *web* para inferir los *rankings* requeridos. Cada hipervínculo de la *web* es entendido como una relación entre un par de documentos: el autor del primer documento considera que el segundo (hacia donde el vínculo apunta) es merecido de ser citado por su "importancia".

A continuación se tratarán dos de los más conocidos algoritmos de *ranking*, entre ellos PageRank, el algoritmo más exitoso comercialmente, el cual es usado por el motor de búsqueda Google.

2.2.1. PageRank

Tomando inspiración en la literatura académica, se puede considerar que un documento es importante o popular si es frecuentemente citado en otros trabajos, esto es, a mayor número de citaciones obtenidas, más popular será. Esta idea también es aplicable en el contexto de la *web*: la gente apunta a páginas que ellos consideran ser importantes. En este sentido, la computación del valor de importancia de una página (que en adelante será llamado el PageRank de una página), requeriría de un simple conteo: el número total de vínculos "entrantes" para cada documento. Esta noción da el mismo valor a cada uno de los vínculos recibidos por cada documento, sin embargo, como en el caso de las citas académicas, algunas referencias deben tener mayor peso por ser más influentes.

Como respuesta a este cuestionamiento, los autores del algoritmo PageRank [36] sugieren una definición recursiva de importancia, la cual propone que la importancia de una página debe estar influenciada por la importancia que tienen las páginas que apuntan hacia ella, esto es, una página debería tener un PageRank alto si las páginas que le apuntan también poseen un PageRank elevado. A partir de esta noción se deriva la siguiente expresión:

$$\pi_i = \sum_{(j,i)\in E} \frac{\pi_j}{out(j)} \tag{2.1}$$

donde el digrafo inducido por la *web* es $G = (V, E)$, $i, j \in V$, π_i es el PageRank del nodo i y $out(j)$ es el grado (exterior) del nodo j, esto es, el número de hipervínculos que posee de la página j[3].

Obsérvese que cada nodo j contribuye a π_i en proporción (inversa) a su grado exterior. En otras palabras, el PageRank de i es la suma de las contribuciones de las páginas que le apuntan, y la contribución de cada una de ellas es su PageRank dividido por su grado exterior (ver figura 2.1).

La idea tras esta fórmula puede ser estudiada usando el modelo del "navegador aleatorio de la *web*" [12]; un personaje hipotético que estando en una página cualquiera elige entre sus vínculos de manera aleatoria y uniforme una nueva página para visitar y repite este proceso cada vez que visita una página. Después de mucho tiempo, la probabilidad con la cual el navegador visita la página i en su caminata aleatoria es justamente el PageRank de i (π_i).

Si la caminata aleatoria es estudiada como una cadena de Markov, es posible definir una matriz de probabilidades de transición de un paso $P = (p_{ij})$ de la siguiente forma:

$$\begin{aligned} p_{ij} &= Pr\{\text{caminante se mueve a } j \text{ dado que esta en } i \text{ actualmente}\} \\ &= \begin{cases} \frac{1}{out(i)} & \text{si } ij \in E \\ 0 & \text{de lo contrario} \end{cases} \end{aligned}$$

Es fácil ver que la matriz P es estocástica siempre y cuando $out(i) \neq 0$:

$$\sum_j p_{ij} = \sum_{j|\ ij\in E} \frac{1}{out(i)} = \frac{1}{out(i)} \sum_{j|\ ij\in E} 1 = 1$$

[3] También llamado el *out-degree* de j.

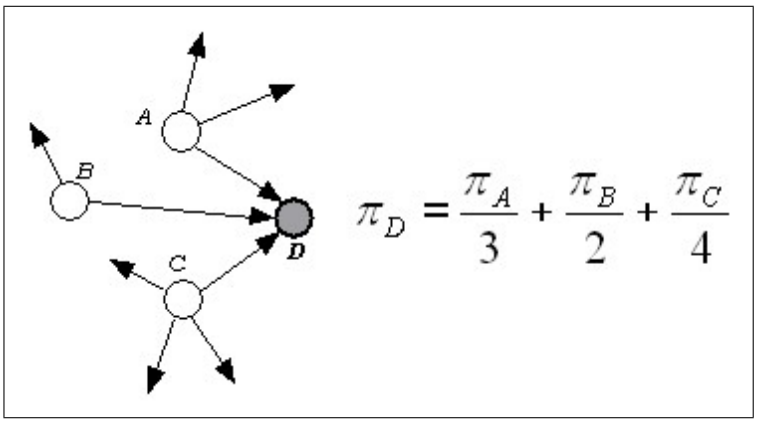

Figura 2.1: Contribución al PageRank de las páginas que apuntan

Cuando $out(i) = 0$, se dice que el nodo i es "colgante" (*dangling node*), es decir, no tiene aristas saliendo de él. Suponiendo por ahora que no existen nodos colgantes, los PageRank de las páginas son las componentes del vector de distribución estacionaria de dicha cadena. Para que dicha distribución π tenga sentido, es necesario que sea independiente del estado inicial (página en la que se inicia la caminata aleatoria), por tanto (y para asegurar la existencia de π), se requiere que la cadena de Markov sea ergódica.

Es bastante probable que G esté compuesto por varias componentes aisladas, de manera que el navegador aleatorio puede quedar atrapado por siempre en un subconjunto de V, es decir que la cadena puede tener varios subconjuntos irreducibles. Para forzar ergodicidad en la cadena se recurre al siguiente artificio:

- Cuando $out(i) = 0$, agregar un vínculo desde i hacia cada uno de los nodos del grafo, de modo que ahora $p_{ij} = 1/N$, es decir, $out(i) = N$ con $|V| = N$.
- Hacer que el navegador aleatorio se "aburra" con probabilidad $1 - d$[4] en cada página de modo que en vez de escoger entre los vínculos de la página actual, salte hacia cualquiera de las N páginas disponibles del grafo. Este tipo de salto se denomina "salto aleatorio" y a $1 - d$ se le conoce como "factor de salto".

Lo anterior garantiza que cualquier página pueda ser visitada con probabilidad positiva por el navegador aleatorio[5]. Es fácil ver que el grafo queda fuertemente conectado de modo que la nueva matriz de transiciones $Q = (q_{ij})$ es de la forma:

$$q_{ij} = \frac{1-d}{N} + dp_{ij} \tag{2.2}$$

La figura 2.2 muestra un ejemplo en el que se ha aplicado el proceso anterior sobre un pequeño grafo.

La distribución de la cadena ergódica es un vector columna π tal que

$$\pi^T = \pi^T Q$$

Existen diversas formas de computar π (un amplio panorama al respecto se puede encontrar en [9]). La técnica usada en este trabajo se ilustra en la sección 5.4.1.

[4] $d \in (0, 1)$.

[5] El salto aleatorio crea "ligeros" vínculos virtuales desde cada página hacia todas las demás.

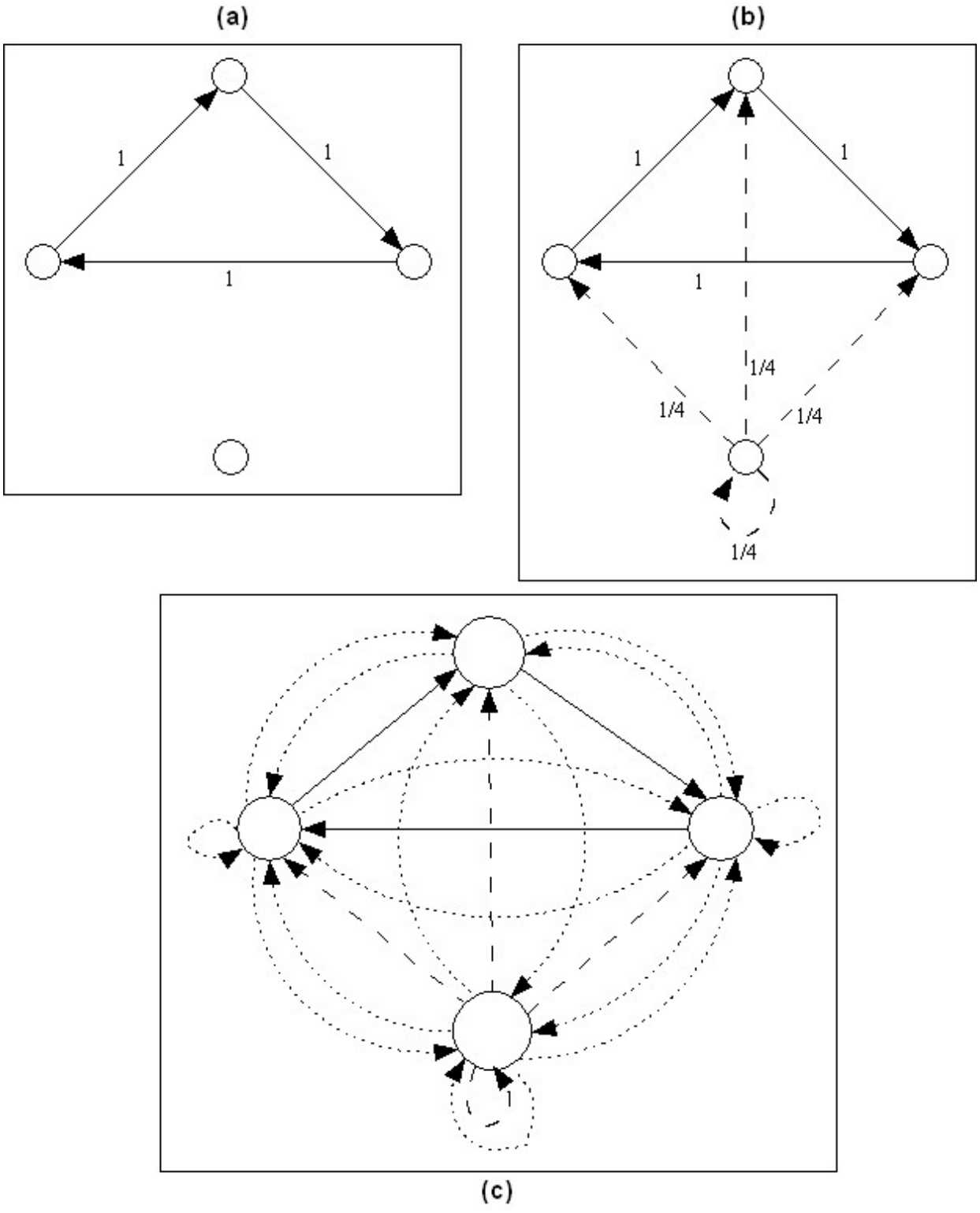

Figura 2.2: Salto aleatorio aplicado a un grafo *web* de ejemplo

(a) Grafo original. Las etiquetas de los arcos indican el valor de la probabilidad de pasar de un nodo a otro. (b) El nodo colgante se ha eliminado. Las líneas discontinuas muestran las nuevas aristas creadas. (c) El salto aleatorio agrega aristas de cada nodo hacia todos los nodos (líneas punteadas). El valor de las líneas sólidas es ahora d, el de las líneas discontinuas es $d/4$ y el de las punteadas $(1-d)/4$.

2.2.2. HITS

HITS es el acrónimo de "*Hypertext Induced Topic Selection*". Este algoritmo sugiere que la importancia de la página depende de la consulta o tema que se está buscando. Es entonces que a partir de un conjunto de documentos relevantes determina dos puntajes para cada uno: un puntaje llamado *authority* y otro llamado *hub*. De los documentos con un puntaje *authority* alto se espera que tengan contenido relevante, mientras que de aquellos con un puntaje *hub* alto, se espera que contengan vínculos hacia documentos de contenido relevante.

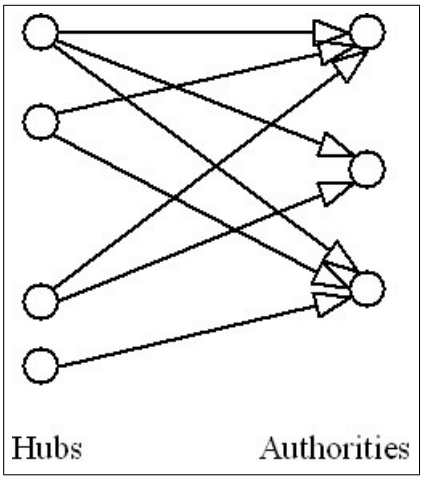

Figura 2.3: *Hubs* y *authorities* en HITS

Según lo anterior un documento que apunta a documentos considerados authorities es un buen *hub*, y por otro lado, un documento apuntado por varios hubs es buen *authority*.

Jon Kleinberg, el autor del algoritmo (ver [30]) propuso en su época (1997) usar un motor de búsqueda basado en texto (como AltaVista [1]) para obtener un conjunto "raíz" que consista de un pequeño grupo de páginas relevantes a la búsqueda del usuario. Luego este conjunto se aumenta usando sus vecinos: el conjunto de documentos que apuntan o son apuntados por nodos en el conjunto raíz. Este nuevo conjunto es llamado conjunto "base".

Sea N el número de páginas en el conjunto base, se conforma la matriz de adyacencias M de modo que $M_{ij} = 1$ si hay uno o más hipervínculos desde i a j, o de lo contrario $M_{ij} = 0$. Para un dado documento i en el conjunto, a_i y h_i representan los puntajes *authority* y *hub* respectivamente, que se calculan del siguiente forma:

$$a_i = \sum_j M_{ji} h_j, \qquad h_i = \sum_j M_{ij} a_j \tag{2.3}$$

Usando notación matricial lo anterior equivale a:

$$a = M^T h, \qquad h = Ma$$

donde los vectores a y h son tales que $a = [a_1, a_2, \ldots, a_N]^T$ y $h = [h_1, h_2, \ldots, h_N]^T$. Desacoplando las expresiones se llega a:

$$a = M^T M a, \qquad h = M M^T h$$

Es claro que tanto a como h son los vectores propios principales de las matrices $M^T M$ y $M M^T$ respectivamente.

problema en HITS por la difficultad de obtener vínculos hacia una página propia. Hoy en día, este es un problema de investigación a considerar pero no hay publicaciones que parezcan haber estudiado el fenómeno aún y ofrecido una solución.

3.1.2. PageRank

Es claro que entre más vínculos entrantes (*inlinks*) una página tiene, es más probable que sea visitada por el navegador aleatorio, lo cual redunda en un valor de PageRank alto. Es por esto que los *spammers* consiguen la mayor cantidad de *inlinks* posibles hacia sus páginas y preferiblemente desde sitios de alta reputación (como lo sugiere la definición recursiva de importancia, ec. 2.1). Así mismo evitan apuntar hacia otros sitios con el objetivo de no contribuir al PageRank de nadie más. Este último hecho implica que una vez el caminante aleatorio ingresa a las páginas de un *spammer* debe permanecer una buena cantidad de tiempo antes de que el salto aleatorio pueda ocurrir.

Es obvio que la manera más simple de hacer *link spam* es por ejemplo, crear muchas páginas en el dominio de Internet que se administra, cada una apuntando exclusivamente hacia un pequeño grupo de ellas con el fin de elevar su PageRank. Afortunadamente las implementaciones reales de PageRank ignoran o dan menor pesos a los vínculos internos. Alternativas más viables son:

- Crear un gran volumen de vínculos desde sitios que permitan hacerlo y que no tienen moderación.
- Pagar a administradores de otros sitios *web* para que creen vínculos desde sus páginas.
- Entrar en acuerdo con administradores de otros sitios *web* para obtener algunos vínculos y ceder algunos otros siempre y cuando se obtenga alguna ganancia en PageRank. Las alianzas de *link spam* son ampliamente estudiadas en [21].

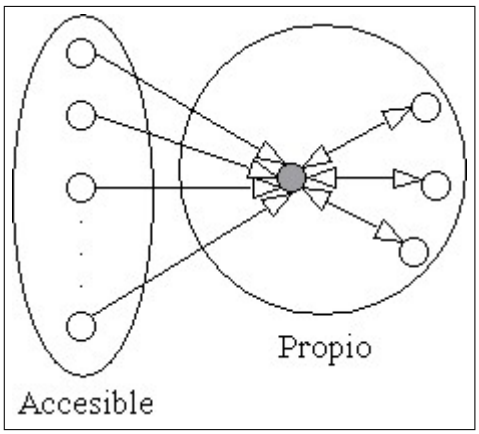

Figura 3.1: "*Single-target link farm*"

La página objetivo está en gris. Los nodos de esta estructura de *link spam* están en dos regiones: la "accessible", es decir, páginas desde las cuales se pueden crear vínculos hacia el objetivo y las "propias" que es la zona administrable por el *spammer* y por tanto desde allí puede crear tantas páginas que vinculen al objetivo como se desee. La página objetivo puede también apuntar a una o más páginas entre la región de lo propio.

Un caso típico de *link spam* se muestra en la figura 3.1, donde una página "objetivo" es apuntada por muchas otras páginas con el objetivo de alcanzar un PageRank más alto del que se merece. Estas páginas llamadas "*boosting pages*" o amplificadoras puede ser propiedad del *spammer* o simplemente páginas desde las cuales se pueden hacer vínculos. A esta estructura se le conoce como "*single-target*

link farm" y resulta ser la más exitosa (la justificación formal de la optimalidad de esta estructura se puede consultar en [21]).

Existen formas más complejas de *spam* en las que por ejemplo grupos de *spammers* se asocian e interconectan sus estructuras ya establecidas (ver figura 3.2). Aún más, los *spammers* se desvían un tanto de las formas estándar (por ejemplo creando algunos vínculos hacia fuera de las alianzas) con el objetivo de ser menos notorios ante posibles mecanismos automáticos de detección de *link spam*.

En [6] se estudian diferentes topologías de colusión y se derivan conclusiones importantes como:

1. Para una página sola, siempre hay algo que ganar si entra en colusión con otros
2. La ganancia por colusión es más baja para páginas que ya están altamente puntuadas

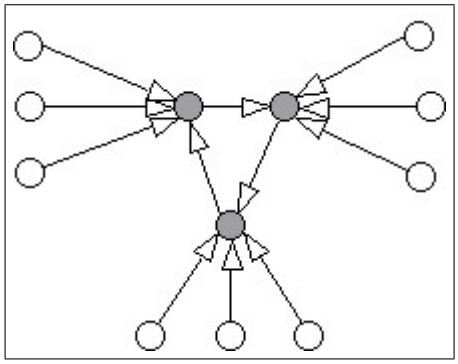

Figura 3.2: Alianza de *spammers*

Alianza de tres *link* farms. Las páginas objetivo (en gris) están interconectadas formando un anillo.

Finalmente, el trabajo de Adali et al. [3], revela cómo el "ataque directo" a una página (*single-target farm*) se puede relajar a un ataque indirecto (sin apuntar de manera directa a la página objetivo) con el fin de que el ataque no sea tan notorio.

Otro elemento importante que se debe considerar es el de alcanzabilidad. Es claro que la página o páginas objetivo son alcanzables por el colector de documentos gracias a las páginas amplificadoras, sin embargo, es posible que se reconozca la "existencia" de una página de manera indirecta usando dominios de Internet separados para cada página. Esto es posible porque los motores de búsqueda usualmente recorren la *web* partiendo desde todos los dominios existentes de modo que una página sin vínculos entrantes puede ser también reconocida.

Cabe aclarar que los vínculos que se hacen desde páginas ya existentes son más beneficiosos porque con seguridad poseen un PageRank mayor que aquellas que están en un dominio que nadie ha apuntado. También este tipo de vínculos tienen un bajo costo (usualmente cero) en comparación con el costo que se paga por un dominio.

3.2. Otras formas de *web spam*

A parte del *link spam* hay más formas de intentar manipular los motores de búsqueda. A continuación se explican estas técnicas:

3.2.1. *Term spam*

Es el método más flagrante y consiste en llenar las páginas *web* con palabras de modo que el indexador del motor de búsqueda infiera equivocadamente que dichos documentos están asociados a un tema que no tiene que ver con el tópico real de estos. Hay varias formas de lograr esto:

- Hacer figurar entre las palabras clave de la página *web* (en el *tag* meta) un conjunto de palabras relacionadas con el tema con el cual se quiera engañar el analizador de contenido.
- Usar un título falso para la página.
- Usar nombres de archivos y directorios de modo que la URL sea una composición de términos con los cuales hacer *spam*.
- Incluir términos en el cuerpo del documento (usualmente al final de la página) para no molestar al usuario con el fin de que el indexador suponga que la página se relaciona con un tema adicional.
- Incluir términos en el cuerpo del documento de tal modo que estén ocultos, por ejemplo usando un color de fuente igual al color de fondo, o usando un tamaño de fuente diminuto, etc., con el mismo propósito anterior.

Este tipo de *spam* también se puede usar para engañar al motor de búsqueda con respecto a los tópicos de una página distinta: Como el texto de los vínculos se supone como un buen indicador de los contenidos de la página a la que apunta el vínculo, esta es una manera de lograr que una página quede erroneamente asociada a un término.

Las técnicas de *term spamming* son tratadas con más detalle en [25].

3.2.2. *Cloaking*

Consiste en mandar contenido distinto al motor de búsqueda cuando se está explorando la *web*, que el que se manda a los usuarios regulares que visitan el sitio. Si un *spammer* puede reconocer que cierta petición la está haciendo el colector de documentos y no un usuario, puede servir un documento con la intención de que éste quede indexado mas no el real. Este tipo de prácticas causan confusión en los usuarios pues obtienen páginas que no tienen nada que ver con sus búsquedas, y sin embargo, las páginas pasan por ser contenido inocente.

A la fecha de escritura de este documento solo existe una publicación acerca del tema ([40]).

3.2.3. *Redirection*

Esta técnica es usada para mandar a los usuarios de modo automático hacia otra URL distinta a la que se solicitó. Esta es una manera de esconder el *spam* de los usuarios y aún quedar indexados en el motor de búsqueda. La redirección se puede efectuar de múltiples maneras: usar el *tag* `meta refresh` de HTML, cambiar el objeto `location` en *Javascript*, usar el método `showDocument` en caso de *applets*, y en general usando *scripting* de componentes que se cargan en la página a partir de *plugins*.

3.3. Combatiendo el *web spam*

Las formas de *spam* mencionadas tienen un alto impacto en la *web*[1]. Algunas estimaciones indican que al menos 8 % de todas las páginas indexadas corresponde a *spam* [19]. Como ejemplo, en una colección *web* obtenida por Eiron et al. encontraron que entre las 20 URLs que ocupan los primeros puestos entre 100 millones de páginas, 11 (que correspondían a pornografía) obtuvieron su *ranking* como producto de técnicas de *link spam* [18]. En otro de sus experimentos se encontró por ejemplo un proveedor de contenido pornográfico que mapeaba aproximadamente 1.4 millones de nombres de *host* distintos. Esto se puede realizar configurando un servidor DNS que pueda mapear un gran conjunto de nombres hacia una misma dirección IP.

Un caso similar descubierto por Fetterly et al. fue el del dominio highriskmortgage.com, en el cual cualquier *host* dentro del dominio resolvía la dirección 65.83.94.42 [19]. En definitiva, el uso de caracteres de comodín en los servidores DNS es una gran oportunidad para los *spammers* dado que

[1] Expertos consideran que el web spam es uno de los más difíciles retos que los motores de búsqueda enfrentan [28]

la mayoría de motores de búsqueda consideran que una página es relevante a un tópico si contiene el nombre de dicho tópico en su URL.

Inclusive, un nuevo campo de negocio relativo a este tema ha visto la luz recientemente [19] y es el de los SEOs (*search engine optimisers*). La actividad de estas compañías consiste en aumentar el *ranking* de los sitios *web* de sus clientes a partir de consultas que estén relacionadas con los negocios de aquellos y así poder aumentar el tráfico sobre estos lugares.

Aún más, existen casos ya conocidos que en la literatura son usados para ilustrar el alcance de dicho fenómeno. En el año 2004, una competencia llamada "*Darkblue* SEO *challenge*" fue celebrada. Los competidores tenían que arreglárselas para salir en el primer puesto arrojado por Google para la consulta "*nigritude ultramarine*"[2], la cual antes de la competencia arrojaba cero resultados y por la fecha de finalización del concurso retornaba unos 500.000 resultados [16]. Otro intento muy prominente en el mismo año fue el de la consulta "*miserable failure*" que retornaba la biografía del presidente estadounidense [3]. A este tipo de ataques se la ha denominado "Google *bombs*".

Aunque el problema de la manipulación de algoritmos de *ranking* fue previsto inicialmente por Page et al. en el artículo que presentaba el algoritmo PageRank, este fenómeno no fue considerado de gran importancia dado los altos costos que implicaba comprar un dominio de Internet en esa época. Hoy en día los precios son bajos y continuan con tendencia a disminuir de manera que el *spam* se expande sin mayor control. Aún más, con la popularización de foros, *blogs* y *wikis*, las oportunidades para hacer *spam* son mucho más amplias.

Hoy en día, unos diez años después, se conocen varios intentos para atacar este problema usando diferentes estrategias:

3.3.1. Uso de nociones de confianza/desconfianza

El uso de nociones de confianza consiste en aprovechar el juicio humano acerca de la calidad (alta o baja) de ciertas páginas. A sitios de "buena reputación" le son asignados puntajes que representan su "fiabilidad"y esta medida se propaga en cierta proporción hacia los sitios a los que éste apunta. De manera similar, si se conoce que determinados sitios son *link spam*, una medida de desconfianza es propagada en sentido inverso (hacia los sitios que apuntan a aquellos de baja reputación). Estos puntajes de confianza o desconfianza se utilizan entonces para premiar o penalizar el PageRank original de las páginas *web*. En esta categoría se pueden encontrar diferentes trabajos como se ilustra a continuación.

Metaxas et al. [34] sugieren un algoritmo que basado a partir de una estimación de la no confiabilidad de las páginas, y propaga hacia atrás en el grafo la medida de desconfianza en un cierto radio. El inconveniente más grande de esta aproximación es su falta de robustez: como el usuario final es quien define desde su *browser* la noción de desconfianza, los *spammers* pueden también participar para limpiar la imagen de sus sitios porque no hay un mecanismo de control. En general todas las técnicas colaborativas sufren de este problema.

Una técnica más robusta es usar una "lista negra" de sitios no confiables predefinida en el motor de búsqueda mismo. Aunque esto es más inmune al *spam*, no es práctico por la inhabilidad de incluir de manera automática nuevos sitios sospechosos [16].

Contrastando con el estudio sobre páginas "sospechosas", el trabajo de Gyöngyi et al. [26] presenta una técnica para identificar sitios o páginas de "confianza" llamada "TrustRank". Dicha técnica calcula que tanta probabilidad tiene una página de ser fiable usando una computación similar a la de PageRank. El algoritmo arranca con un concepto de confianza dado por un experto humano sobre un pequeño grupo de páginas y propaga la noción hacia adelante usando un factor de amortiguamiento.

Una manera más interesantes es la presentada por Wu et al. [39] que consiste en automáticamente hallar un conjunto inicial de sitios de *spam* y amplificar luego este conjunto base propagando la desconfianza. El conjunto inicial (semilla) se obtiene analizando la estructura de vínculos entrantes y salientes entre dominios.

[2] Una frase sin sentido

3.3.2. Clasificadores basados en reglas

Este tipo de intentos parten de la extracción de características en los nodos de la *web* y a partir de ellas determinar de manera separada si un nodo es *spam* o no. Una de las primeras contribuciones al respecto es la de Davison [15] quien en su trabajo anticipó la importancia de los vínculos nepotistas y discutió los efectos que este fenómeno puede tener en el *ranking*. Características tales como título de la página, descripción, dominio, nombre de *host*, dirección IP, número de vínculos entrantes y salientes, etc., son tenidos en cuenta para construir un árbol de decisión y determinar si hay vínculos "buenos" o "nepotistas". El árbol se construye a partir de conjuntos anotados manualmente. Aunque los resultados obtenidos en este trabajo son muy pobres, se logra dar la idea del reto que constituye implementar mecanismos de detección automatizada o semi-automatizada.

Una alternativa más reciente y acertada es la de Amitay et al. [5], en la cual se representa cada nodo como un vector en el que cada posición indica un característica relacionada con la conectividad (ver más detalles en [5]) y tras hallar medidas de similaridad entre parejas de vectores y aplicar *clustering* les fue posible descubrir sitios de *link spam*.

3.3.3. Modificaciones en el digrafo

La evidencia experimental [18] demuestra que los vínculos externos (vínculos entre sitios *web* distintos) generalmente apuntan a el nodo de más alto nivel del sitio de destino[3], esto es, al *homepage*. Esto sugiere usar una alternativa orientada a bloques para calcular el PageRank considerando todas las páginas de un sitio como una sola entidad de información. Sobre este grafo más pequeño en el que los nodos son nombres de *host* distintos, el PageRank o "HostRank" se puede calcular. Este cálculo es mucho más simple y se ha probado que es mucho menos sensitivo al *link spam* según los experimentos de Eiron et al.

Tratar un solo nombre de *host* como una unidad puede ser considerado como una simplificación no adecuada porque existen muchos nombres de *host* que alojan páginas con contenido variado como el caso de sitios *web* que contienen páginas personales de cientos de personas que no tienen ningún tipo de relación entre ellas (por ejemplo geocities.com). Esto sugiere que una granularidad más fina sería más apropiada. Por ejemplo, como la información es estructurada de manera jerárquica siguiendo la estructura de un directorio en disco, es más adecuado designar todas las URLs bajo un directorio como una unidad simple. Agrupar URLs que concuerden hasta su último '/' es una variación del HostRank conocida como "DirRank". También hay evidencia (ver [18]) de que el DirRank es menos propenso a la manipulación.

La ventaja más significativa de este tipo de métodos es el ahorro en recursos computacionales para calcular los *rankings* dado que la matriz de adyacencias es mucho más pequeña.

3.3.4. Modificaciones en el mecanismo del salto aleatorio

El fenómeno de salto aleatorio también ha sido estudiado para determinar su papel en relación con el *link spam*. Por ejemplo en [22] se brinda un método para identificar grupos de páginas fuertemente interconectadas definiendo un factor de amplificación equivalente al cociente entre el PageRank total de las páginas en el grupo y la "contribución" recibida desde otras páginas por fuera del grupo. Si esta amplificación es cercana a un umbral (más exactamente el recíproco del factor del salto aleatorio), se dice que el conjunto de páginas están en colusión. Esto significa que el grupo "estanca" el camino aleatorio por un tiempo equivalente a esta cantidad antes de que el salto aleatorio tome lugar. Para tratar este problema, se calcula el PageRank varias veces usando valores incrementales para d, y se analiza la correlación de los puntajes obtenidos con el factor de salto. Aquellas páginas en las que la dependencia con respecto a d es más fuerte son consideradas sospechosas, mientras que aquellas en las que el PageRank es poco sensitivo a los cambios en d son consideradas páginas regulares.

[3] Alrededor de un 70 % de los casos

Otra estrategia para aliviar el problema se sugiere en [18] donde el salto aleatorio se hace saltando únicamente hacia un pequeño conjunto de páginas "fiables". Este tipo de salto es conocido como "trusted teleportation" y según los experimentos presentados en este trabajo demostró ser de gran ayuda. Los efectos de este salto aleatorio controlado parecen tener un impacto profundo sobre los *rankings* obtenidos, sin embargo no hay un trabajo riguroso donde se demuestre su alcance. Este modelo es mucho más realista que el del salto aleatorio tradicional dado que los usuarios tienden a usar portales o buscadores como puntos de visita inicial y de ahí en adelante hacen algún tipo de exploración y finalmente retornan a la página inicial para reformular su consulta, por ejemplo, hasta que sus necesidades de información sean satisfechas.

Este par de métodos son robustos en el sentido de que las variaciones en el mecanismo del salto están fuera del alcance de los *spammers*.

3.3.5. *Outliers* en distribuciones

En el trabajo de Fetterly et al. [18] se caracteriza el *web spam* usando análisis estadístico. La distribución de variables como el número de nombres de *host* distintos que mapean la misma dirección IP, el número de vínculos entrantes y salientes, etc. son obtenidas. La mayoría de estas distribuciones resultan seguir patrones bien definidos y los *outliers* en los conjuntos de datos son productos del *spam*.

Para el caso del *link spam* las distribuciones más significativas son las que involucran el número de vínculos que entran y salen de cada página. Las regularidades de estas distribuciones son marcadas y siguen una forma "*power law*" [31]: La fracción de páginas *web* con *in-degree* i es proporcional a $1/i^x$ donde el valor del exponente x es un consistente 2,1 y para el caso de *out-degree* la misma forma aplica usando $x = 2{,}72$.

Esta no es una técnica de detección per se dado que los *outliers* solo representan que por ejemplo, para una variable como el *in-degree* puede existir una condición anómala para un valor particular i, en tanto que algunas de las páginas con dicho *in-degree* serán *spam*, sin embargo, no es posible saber cuáles son de manera automática. También, las formas de *spam* menos prominentes también pueden pasar desapercibidas por estar dentro de los valores de *in-degree* o *out-degree* aceptables.

Capítulo 4

Clustering Espectral

4.1. Cortes sobre grafos

El *clustering* espectral se presenta como una estrategia de solución al problema de partición de grafos el cual consiste en "cortar" un grafo dado en dos o más partes de tal forma que sean "lo más disyuntas que se pueda". De manera más formal, dado un grafo no dirigido $G = (V, E)$ se busca particionar el conjunto V en conjuntos disyuntos $V_1, V_2, \ldots, V_m$ tal que la similaridad entre los vértices en un conjunto V_i sea alta mientras que la similaridad entre los vértices de conjuntos distintos V_i, V_j sea baja. Para trabajar estas nociones de similaridad al interior de un conjunto y entre conjuntos distintos se acude al uso de una matriz de pesos asociada $W = (w_{ij})$ que representa la afinidad entre cada pareja de nodos: un valor w_{ij} alto indica que i es bastante similar a j, mientras que un valor bajo indica poca similitud. Se asume que esta matriz es simétrica.

En la figura ? se muestra un ejemplo de un *clustering* realizado sobre un conjunto de puntos. La distancia entre nodos representa inversamente la similaridad entre ellos: nodos más distantes se asumen más disímiles. Suponga que se desea obtener este agrupamiento y que sólo se puede usar una operación que consiste en dividir un subgrafo en dos subgrafos más pequeños. La manera de obtener estos grupos es realizando un primer corte y realizando bisecciones una y otra vez en cada trozo obtenido hasta llegar al resultado deseado, ver figura ?.

Esta bisección requiere en cada momento tomar un grafo $G = (V, E)$ y dividir V en dos conjuntos: S y $V \backslash S$. Esta división se conoce como un "corte" y a los conjuntos S y $V \backslash S$ se les denomina los "lados" del corte. Una manera de realizar un corte es escoger S de modo que el valor de $cut(S)$ sea muy pequeño, donde dicha cantidad es conocida como valor o expansión del corte:

$$cut(S) = \sum_{i \in S, j \in V \backslash S} w_{ij}. \tag{4.1}$$

Esto se hace con el fin de que el corte no biseccione algún *cluster* que haga parte del resultado final, logrando pasar únicamente entre las fronteras que separan los grupos. Nótese que esto va en favor de la baja similitud requerida entre *clusters*. Así mismo, si se obtiene un valor de corte muy alto para un conjunto dado, esto significa que no debe cortarse más por tratarse de un conjunto bien conectado, lo cual, está en favor de la alta similitud al interior del *cluster*.

Es fácil ver que este criterio favorece a las particiones desbalanceadas en tamaño, es decir, como la expresión 4.1 aumenta a medida que más aristas cruzan el corte, esto puede originar conjuntos S tal que están compuestos apenas por un nodo (ver figura 4.1).

Para contrarrestar el sesgo hacia los cortes con muy pocos nodos en uno de sus lados, se puede requerir que $|S| = |V \backslash S|$ sea una restricción en el problema de minimización. En vez de usar el tamaño de manera directa, otra estrategia consistiría en forzar $w(S) = w(V \backslash S)$, donde $w(A) = \sum_{i,j \in A} w_{ij}$.

Esto implica un balance en términos de conectividad total para cada lado del corte[1] .

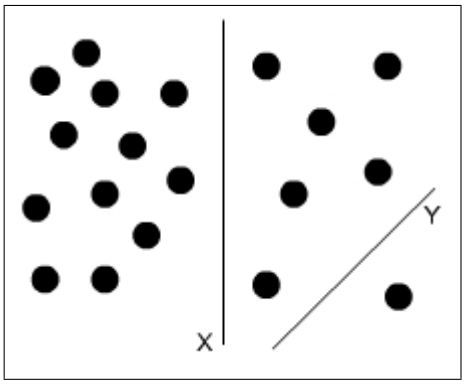

Figura 4.1: Corte mínimo y corte balanceado

En este grafo la similaridad entre dos nodos es inversamente proporcional a su distancia (un par de nodos i, j alejados tendran un valor de w_{ij} pequeño). En este sentido el corte B (de mínima expansión) lleva a un subconjunto con un nodo y otro con el restante de nodos. El corte balanceado A parecer ser más deseable.

Una manera que resulta equivalente es utilizar $vol(S) = vol(V \backslash S)$, donde $vol(A)$ se conoce como el "volumen"[2] del conjunto A y se define como:

$$vol(A) = \sum_{i \in A, j \in V} w_{ij}.$$

La equivalencia se da gracias a que $vol(A) = w(A) + cut(A)$ y $vol(V \backslash A) = w(V \backslash A) + cut(A)$. Lo anterior lleva a formular el siguiente problema de optimización que nos conduciría hacia un corte "balanceado":

$$Bcut(G) = \underset{S \subset V, vol(S) = vol(V \backslash S)}{\arg\min} cut(S) \tag{4.2}$$

Es claro que una búsqueda exhaustiva por el mejor corte no es factible porque existen $2^{n-1} - 1$ cortes distintos[3], lo que implica un espacio de búsqueda grande aún para valores pequeños de n.

Si se define $D = (d_{ij})$ como una matriz diagonal tal que $d_{ii} = \sum_j w_{ij}$y si se expresa el corte como un vector f tal que $f \in \{-1, 1\}^n$ donde $f_i = 1$ si $i \in S$ o $f_i = -1$ en caso contrario, se concluye a partir de los hechos expuestos en el apéndice A.1 que:

1. $vol(S) = vol(V \backslash S)$ implica $f^T D\mathbf{1} = 0$
2. $cut(S) = \frac{1}{4}\ f^T(D - W)f$

En tanto que el criterio se convierte en

$$Bcut(G) = \underset{f \in \{-1,1\}^n, f^T D\mathbf{1} = 0}{\arg\min} \left[\frac{1}{4} f^T (D - W) f\right]$$

[1] Esta es una estrategia más elaborada que la de obligar tamaños iguales: a cada nodo se le está dando un peso distinto en función de qué tan conectado se encuentra.

[2] El volumen es la conexión total desde los nodos de un conjunto específico hacia todos los nodos del grafo.

[3] El número de cortes distintos es $\frac{1}{2}(2^n - 2)$, es decir todas las combinaciones de dos elementos obligando a que aparezca al menos un elemento en uno de los lados y obviando las combinaciones complementarias.

y aprovechando el hecho que $f^T Df = vol(V)$, se puede introducir $4/vol(V)$ como una constante multiplicativa

$$\begin{aligned} Bcut(G) &= \underset{f\in\{-1,1\}^n, f^T D\mathbf{1}=0}{\arg\min} \left[\frac{f^T(D-W)f}{vol(V)}\right] \\ &= \underset{f\in\{-1,1\}^n, f^T D\mathbf{1}=0}{\arg\min} \left[\frac{f^T(D-W)f}{f^T Df}\right] \end{aligned} \tag{4.3}$$

Permitiendo a f_i tomar valores reales y no únicamente $\{1,-1\}$, el vector que minimizaría $Bcut$ es justamente el vector propio correspondiente al segundo valor propio más pequeño del siguiente sistema generalizado (ver apéndice A.3):

$$(D-W)f = \lambda Df \tag{4.4}$$

Relajar el problema de modo que f ya no sea un vector binario hace que la igualdad en volúmenes no se tenga de manera estricta.

Si se expande la expresión anterior:

$$\begin{aligned} Df - Wf &= \lambda Df \\ Wf &= (1-\lambda)Df \\ D^{-1}Wf &= (1-\lambda)f, \end{aligned}$$

se puede ver que f es vector propio de la matriz $D^{-1}W$ y que sus valores propios (llámense λ_i') se relacionan con los del sistema generalizado (4.4) según $\lambda_i = 1-\lambda_i'$. Nótese que $D^{-1}W$ es una matriz estocástica que correspondería a una cadena de Markov definida sobre G: la probabilidad de transición entre nodos $i,j \in E$ es w_{ij}/d_{ii} o cero si $i,j \notin E$.

Los valores propios de la matriz de transiciones de una cadena de Markov no necesariamente son reales, sin embargo según A.3 los λ_i son reales y por tanto los λ_i' también. Así mismo se espera que $\lambda_i' \in [-1,1]$ (ver [35]). Si se quisiera aplicar un algoritmo como el método de potencias (ver apéndice C) con el fin de hallar algún vector correspondiente a por ejemplo el r-ésimo valor propio más grande de dicha matriz, habría que asegurarse que todos los valores propios son positivos. Esto se puede lograr convirtiendo la cadena original en una cadena perezosa, en las que sus valores propios se encuentran entre 0 y 1 [8]. La manera de construir dicha cadena a partir de $D^{-1}W$ es la siguiente[4]:

$$X = \frac{1}{2}(I + D^{-1}W). \tag{4.5}$$

Los valores propios λ_i'' de esta nueva matriz se relacionan con los de $D^{-1}W$ según $\lambda_i' = \frac{\lambda_i''-1/2}{1/2}$ aplicando el resultado visto en el apéndice A.2. De lo anterior se concluye que:

1. $\lambda_i'' \in [0,1]$ porque $\lambda_i' \in [-1,1]$
2. $\lambda_i = 2(1-\lambda_i'')$
3. El segundo valor propio más chico del sistema 4.4 corresponde al segundo valor propio más grande de X.
4. Es posible aplicar el método de potencias del apéndice C sobre la matriz X para obtener f.

[4] Esta construcción implica debilitar a su mitad cada una de las transiciones desde cualquier nodo hacia todos los demás y agregar una transición desde cada nodo hacia sí mismo con probabilidad de 1/2.

4.2. Particionando de acuerdo a f

f determina la partición de un conjunto de nodos en un cierto paso del algoritmo, pero dado que se permitió $f \in \mathbb{R}^n$ es necesario establecer cómo realizar dicho corte a partir del vector. El corte puede ser construido usando diversas políticas [38]:

1. *Splitting value*: Se usa un valor real s de modo que $i \in A$ si $f_i \geq s$ o $i \in B$ si $f_i < s$, donde A y B son los lados del corte. Un valor para s podría ser cero (lo que se conocería como corte por signo) o la mediana de $\{f_1, f_2, \ldots, f_n\}$ (corte por bisección).

2. *Gap cut*: Si y es el vector con las componentes de f ordenadas, sea $t = \arg\max_i(y_{i+1} - y_i)$, entonces los conjuntos que componen el corte son $\{y_1, y_2, \ldots, y_t\}$ y $\{y_{t+1}, y_{t+2}, \ldots, y_n\}$.

3. *Ratio cut:* Se obtiene una secuencia de índices $i_1, i_2, \ldots, i_n$ tal que $f_{i_1} \leq f_{i_2} \leq \ldots \leq f_{i_n}$ y se evalúa $\phi(A, B)$ en los $n-1$ cortes definidos por $A = \{i_1, i_2, \ldots, i_t\}$ y $B = \{i_{t+1}, \ldots, i_n\}$ para $1 \leq t < n$, escogiendo los conjuntos A y B donde se minimice

$$\phi(A, B) = \frac{cut(A)}{vol(A)vol(B)}. \tag{4.6}$$

   ϕ se está usando para medir la calidad del corte: se prefiere un corte cuyo valor sea bajo y sus lados tengan volúmenes similares. Esto se puede ver dado que $vol(B) = vol(A \cup B) - vol(A)$, y por tanto el denominador de $\phi(A, B)$ equivale a $vol(A)vol(A \cup B) - (vol(A))^2$. Esta expresión toma su valor máximo justamente cuando $vol(A) = vol(B)$, es decir cuando $vol(A) = \frac{1}{2}vol(A \cup B)$ (ver figura 4.2)[5]. Esto implica que la ec. 4.6 es menor cuando ambos conjuntos están balanceados.

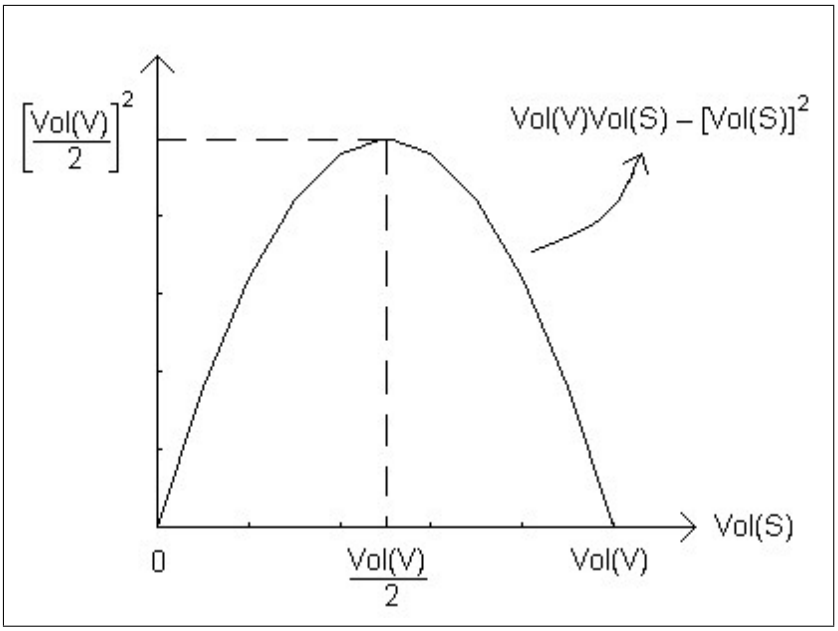

Figura 4.2: Volumen balanceado

[5] $vol(S)vol(V) - (vol(S))^2$ es una función cóncava con un máximo tal que $vol(V) - 2vol(S) = 0$

4.3. Algoritmo

Ya que el algoritmo de *clustering* espectral define cómo cortar un grafo en dos partes, es necesario aplicar la misma técnica sobre cada uno de los lados para obtener particiones más finas. Esto se realiza removiendo las aristas del grafo que atraviesan el corte, es decir, que conectan a S con $V \backslash S$ y aplicando *clustering* espectral sobre cada lado:

Algorithm 1 *SpectralClustering*

***Entradas**: un grafo $G = (V, E)$ y su matriz de similaridades W*

Obtener un vector f a partir de alguno de los sitemas 4.4 o 4.5

Decidir qué nodos componen cada uno de los lados del corte de acuerdo a una política (ver sección 4.2)

Decidir si se sigue subdividiendo a partir del valor del corte obtenido

Si es necesario, aplicar el algoritmo de nuevo sobre cada uno de los lados removiendo las aristas que atraviesan el corte

Capítulo 5

Detección de *link spam*

5.1. Requerimientos sobre un corte

Como se vio en la sección 3.1, las estructuras de vínculos en las que el navegador aleatorio quedaría atrapado fácilmente son un indicador clave de la presencia de *link spam*[1]. Para medir esta "fuerza de absorción" se puede hacer uso de un concepto de la teoría de cadenas de Markov llamado conductancia Φ_S [29] y que sirve para medir el chance que existe de dejar un conjunto S en un paso. La conductancia es usualmente utilizada para estudiar el tiempo de mezcla (*mixing time*[2]) de una cadena.

De manera concreta, para un subconjunto S del espacio de estados Ω de una cadena de Markov ergódica, Φ_S se define como:

$$\Phi_S = \frac{Q(S, \Omega \backslash S)}{\pi(S)} \tag{5.1}$$

donde π es la distribución estacionaria de la cadena y Q es conocido como el flujo ergódico entre dos conjuntos y se define como:

$$Q(A, B) = \sum_{x \in A, y \in B} \pi_x p_{xy} \tag{5.2}$$

con $A, B \subset \Omega$ y p_{xy} la probabilidad de transición entre los estados x y y de la cadena. Una importante propiedad del flujo es la simetría: para todo conjunto no vacío $S \subset \Omega$ se cumple que $Q(S, \Omega \backslash S) = Q(\Omega \backslash S, S)$ (ver apéndice B.1).

Φ_S se puede interpretar como la probabilidad de dejar S dado que actualmente se está en el interior de S, por tanto los subconjuntos del espacio de estados de la cadena definida por la caminata del navegador aleatorio que tengan conductancia baja son buenos candidatos para ser *link spam*.

Para usar un algoritmo basado en cortes como en el caso de *clustering* espectral, es necesario observar que se requiere que cada corte separe dos conjuntos entre los que el flujo sea bajo (sea difícil pasar del uno al otro), así mismo, que un conjunto en donde la conductancia sea baja no sea cortado. Usar sólamente el criterio de flujo para cortar puede degenerar en cortes desbalanceados de por ejemplo un nodo y $k-1$ nodos (donde k es el tamaño del conjunto que se está cortando) como en el caso visto en la figura 4.1. Para asegurar cortes mejor balanceados se puede usar por ejemplo una medida de igual tamaño (probabilístico) para cada lado del corte. Se puede pensar en $\pi(S) = \pi(\Omega \backslash S)$, o lo que es equivalente, en $\pi(S) = \frac{1}{2}$.

[1] La existencia de estas estructuras también puede darse por vías naturales como en el caso de sitios corporativos en los que una empresa nunca apuntaría al sitio web de alguno de sus competidores a menos que se trate de un convenio. Sin embargo, vale la pena considerar si la detección de estas regiones puede conducir hacia la identificación efectiva de *spam*.

[2] Cuánto demora en alcanzar estabilidad una cadena de Markov

5.2. *Clustering* espectral sobre la cadena de Markov de la *web*

Si se considera formar un grafo a partir de los estados de la cadena de Markov tal que su matriz de pesos sea $W = \pi_i q_{ij}$, se cumple que para todo $S \subset \Omega$

$$\begin{aligned} vol(S) &= \sum_{i\in S, j\in\Omega} \pi_i q_{ij} \\ &= \sum_{i\in S} \pi_i \sum_{j\in\Omega} q_{ij} \\ &= \sum_{i\in S} \pi_i = \pi(S), \end{aligned}$$

en tanto que la optimización descrita en 4.2 se convierte en

$$\begin{aligned} Bcut(G) &= \operatorname*{arg\,mín}_{S\subset\Omega, \pi(S)=\pi(\Omega\backslash S)} \sum_{i\in S, j\in\Omega\backslash S} \pi_i q_{ij} \\ &= \operatorname*{arg\,mín}_{S\subset\Omega, \pi(S)=\frac{1}{2}} Q(S, \Omega\backslash S) \end{aligned}$$

siendo equivalente a la estrategia de bisección descrita en la sección inmediatamente anterior.

Ahora, como la matriz W es obligatoriamente simétrica, esto implica que $\pi_i q_{ij} = \pi_j q_{ji}$, es decir, que la cadena tiene que ser reversible [17]. Es claro que este no es el caso de la *web* y por tanto es necesario construir una nueva cadena a partir de la original de modo que se conserven propiedades como el flujo y la distribución estacionaria. Dicha cadena puede ser construida tal que $r_{ij} = \frac{1}{2}\left(q_{ij} + \frac{\pi_j}{\pi_i} q_{ji}\right)$ siendo R su matriz de transición (ver apéndice B.2). Habiendo realizado esta reversibilización, queda demostrado que el algoritmo de la sección 4.3 puede ser aplicado en este contexto usando $W = \Pi R$, donde $\Pi = diag(\pi_1, \ldots, \pi_N)$.

Dado que $w_{ij} = \pi_i r_{ij}$ se tiene que $D = \Pi$:

$$\sum_j w_{ij} = \pi_i \sum_j r_{ij} = \pi_i \tag{5.3}$$

lo cual implica que la cadena perezosa sobre la cual se operará es

$$X = \frac{1}{2}(I + D^{-1}W) = \frac{1}{2}(I + R)$$

5.3. Identificación de *spam* y criterio de parada

Goel et al. [22] proponen el uso de una medida llamada amplificación para medir la "fuerza de colusión" existente dentro de un subconjunto de páginas web. Usando la notación aquí empleada, esta medida puede ser reescrita de la siguiente manera:

$$Amp(S) = \frac{\text{mín}\{\pi(S), \pi(\Omega_0\backslash S)\}}{Q(S, \Omega_0\backslash S)} \tag{5.4}$$

donde Ω_0 corresponde al espacio de estados de la cadena de Markov inicial (sin haberle aplicado ningún corte). Goel et al. establecen que cuando existe colusión en un grupo, este coeficiente es elevado.

Nótese que la amplificación corresponde exactamente con el recíproco de $\Phi(S)$ si $\pi(S) < \pi(\Omega_0\backslash S)$ y que $1/\Phi(S)$ cuantifica que tanto "gana" en PageRank un subconjunto con respecto a las "contribuciones" que éste recibe: la ganancia en PageRank de un subconjunto S es $\pi(S)$ y la contribución global que recibe es el flujo hacia ese conjunto, es decir,

$$Q(\Omega_0\backslash S, S) = \sum_{x\in\Omega_0\backslash S} \sum_{y\in S} \pi_x p_{xy}.$$

De este modo, se espera que en presencia de *link spam* $\Phi(S)$ sea pequeño.

La equivalencia entre *Amp* y $1/\Phi$ siempre se tendrá porque si se está tratando de decidir si S es *spam* o no, el conjunto S será generalmente mucho más pequeño que $\Omega_0 \backslash S$, en tanto que también se cumpliría que $\pi(S) < \pi(\Omega_0 \backslash S)$[3]. La decisión de si un subconjunto S se interpreta como *spam* (positivo) o no *spam* (negativo) se puede basar en la observación establecida en [22] que dice que si la amplificación es cercana a $\frac{1}{1-d}$ es porque existe colusión. Para el caso de la conductancia esto implica que si llegar a ser tan pequeña como 0,15 hay *spam*.

Cuando no se puede probar que un conjunto es *spam*, este tendrá que intentarse subdividir nuevamente (a menos que el conjunto ya sea de tamaño unitario, en cuyo caso definitivamente queda clasificado como negativo).

5.4. Detalles de la computación

5.4.1. Distribución estacionaria

Es claro que para hallar el vector propio de X es necesario conocer la matrix R y por tanto el vector de la distribución estacionaria correspondiente a la matriz Q. El calculo de π puede ser llevado a cabo aplicando un algoritmo iterativo para solucionar sistemas de ecuaciones lineales simultáneas a partir de la siguiente observación:

$$\begin{aligned} \pi_j &= \sum_i \pi_i q_{ij} = \sum_i \pi_i \left(\frac{1-d}{N} + d p_{ij} \right) \\ &= \frac{1-d}{N} \sum_i \pi_i + d \sum_i \pi_i p_{ij} \\ &= \frac{1-d}{N} + d \sum_i \pi_i p_{ij} \end{aligned}$$

que escrito en forma matricial es

$$\pi = \frac{1-d}{N}\mathbf{1} + d\pi P^T$$

o de manera equivalente

$$(I - dP^T)\pi = \frac{1-d}{N}\mathbf{1}$$

sujeto a la restricción $\sum_i \pi_i = 1$.

Para resolver este sistema se puede utilizar por ejemplo el método de Gauss-Seidel (ver apéndice C) sustituyendo $A = I - dP^T$, $x = \pi$ y $b = \frac{1-d}{N}\mathbf{1}$

En digrafos reales el valor de N esta en el orden de 10^9 nodos[4], en tanto que las componentes del vector b no representan ningún riesgo, por ejemplo, de desbordamiento inferior en la representación numérica. Así mismo la matriz A es dispersa dado que el número promedio de componentes diferentes a cero en cada fila de la matriz P^T es un valor pequeño y por tanto cada iteración del algoritmo está en $\Theta(N)$.

En la práctica, el número de iteraciones requeridas es una constante pequeña que depende del umbral de tolerancia establecido para la convergencia y del segundo valor propio de P^T. Haveliwala et al. ([27]) demostraron que dicho valor propio es exactamente igual al factor d, que para el caso del algoritmo de Gauss-Seidel estaría determinando la tasa de convergencia, esto es, a medida que $d \to 1$ son requeridas más iteraciones. La escogencia de un valor pequeño para d ($d \to 0$, por ejemplo) implica que el navegador aleatorio salta desde una página a cualquier otra de manera uniforme sin dar mayor

[3] Es decir $\pi(S) < 1/2$.

[4] N idealmente es el total de páginas en toda la web, sin embargo por razones de límites físicos en recursos computacionales no es posible tener un cubrimiento total. Inclusive se puede pensar que el número de páginas puede ser realmente infinito ([18]).

importancia a los vínculos que se encuentran en la página actual y obviamente esto degeneraría en una distribución uniforme para π que no es útil.

5.4.2. Transiciones que atraviesan un corte

Es claro que a partir de la cadena reversible definida por R el algoritmo de *clustering* espectral establece un corte que separa el espacio de estados de dicha cadena en dos partes. Llamése $R^{(2)}$ a la submatriz que contiene únicamente las transiciones de los estados que han quedado en alguno de los dos lados. Es claro que $R^{(2)}$ no es estocástica porque se pierden las transiciones que cruzan a lo largo del corte (ver figura 5.1). Es necesario entoces "arreglar" $R^{(2)}$ para que se pueda establecer nuevamente un corte sobre dicha parte:

$$r_{ii}^{(2)} = 1 - \sum_{j \neq i} r_{ij}$$

Esta nueva cadena tiene asociada una distribución de probabilidad

$$\pi^{(2)} = \frac{1}{\pi(S)}\pi$$

donde S es el conjunto de estados que han quedado en la matriz $R^{(2)}$ (ver [4]).

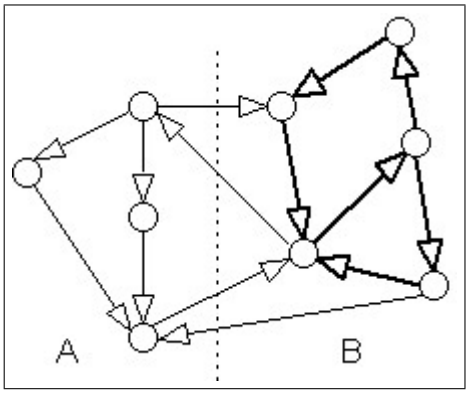

Figura 5.1: Pérdida de estocasticidad en un corte

Nótese cómo varias aristas atraviesan el corte. El lado B por ejemplo, deja de ser una cadena de Markov (sólo las aristas resaltadas se conservan)

El hecho de modificar la entrada (i, i) de la matriz $R^{(2)}$ no tiene mayores implicaciones en el algoritmo dado que la conductancia no toma en cuenta las probabilidades de transición desde un nodo hacia sí mismo. Igualmente durante todo el el procedimiento de corte recursivo se puede usar la misma matriz X prestando atención solo a aquellas componentes que hacen parte del conjunto que se está cortando y "ajustando" su diagonal en cada momento para que la estocasticidad sea conservada.

5.5. Algoritmo y complejidad computacional

A continuación se presenta el algoritmo de detección de *link spam* en pseudolenguaje:

Algorithm 2 *SpamDetection(G)*
***Entradas**: Un digrafo $G = (V, E)$*

Obtener matriz Q a partir de G (ver sección 2.2.1)
Calcular matriz diagonal $\Pi = diag(\pi_1, \ldots, \pi_N)$
Calcular matriz X (ver 5.5)
Aplicar ClusteringMarkov(X,Π,Q)

Algorithm 3 *ClusteringMarkov(X, Π, Q)*
***Entradas**: matriz X, matriz diagonal Π y matriz global de la web Q*
Computar el vector propio f correspondiente al segundo mayor valor propio de X
Usar f para obtener una partición conveniente A, B sobre el conjunto de nodos en cuestión (usando la política de ratio cut)
Si $\pi(A) < 1/2$, calcular $\Phi(A)$ usando Q para etiquetar el lado como positivo o negativo. de lo contrario etiquetarlo como negativo
Si $\pi(B) < 1/2$, calcular $\Phi(B)$ usando Q para etiquetar el lado como positivo o negativo. de lo contrario etiquetarlo como negativo
Aplicar el algoritmo de nuevo sobre cada lado señalado como negativo (siempre y cuando tenga más de un nodo) usando la submatriz de X y de Π que contengan únicamente las filas y columnas de interés (nodos pertenencientes a cada lado) y la matriz Q

Para el análisis del comportamiento asintótico del algoritmo de detección de *link spam* se asume que el número de arcos del grafo es M y su número de nodos es N. Así mismo, nótese que la matriz de transiciones de la cadena de Markov reversible se puede escribir en términos matriciales como

$$R = \frac{1}{2}(Q + \Pi^{-1}Q^T\Pi)$$

en tanto que la matriz X es tal que

$$X = \frac{1}{2}(I + R) = \frac{1}{2}I + \frac{1}{4}Q + \frac{1}{4}\Pi^{-1}Q^T\Pi \tag{5.5}$$

Como se verá a más adelante, el acceso a entradas en las matrices X y Q son primoridales para los cálculos. Inclusive, no se construye X de manera explícita dado que los valores x_{ij} se pueden obtener a partir de π y Q. Por ahora asúmase que acceder a un elemento q_{ij} toma $\Theta(1)$, y, aún más, que hacer una suma de la forma $\sum_{j\in A} q_{ij}$ para un i fijo y un conjunto arbitrario $A \subset V$ está también en $\Theta(1)$. De lo anterior se derivaría entonces que el acceso a un elemento x_{ij} está en $\Theta(1)$ porque $x_{ij} = \frac{1}{2}\delta_{ij} + \frac{1}{4}q_{ij} + \frac{\pi_j}{4\pi_i}q_{ji}$.

Los ítems a continuación describen la complejidad de cada uno de los pasos requeridos para hacer una partición binaria sobre un conjunto de n elementos:

- Obtener el segundo vector propio de una matriz de tamaño n usando el método de potencias está en $\Theta(n \log n)$. Esto se puede lograr gracias a que la matriz siempre se puede escribir como la suma de varias matrices dispersas y otras matrices cuyos elementos son una constante a lo largo de todas las entradas (epor ejemplo). Así, la operación básica del método (multiplicación de una matriz por un vector) está en $\Theta(n)$. Ver más detalles en [14].
- El ordenamiento de dicho vector toma $\Theta(n \log n)$ usando una técnica eficiente[5].
- Evaluar los $n-1$ cortes inducidos por el vector ordenado (ver *ratio cut* en 4.2) toma $\Theta(n)$. Esto se logra aprovechando el hecho que el flujo del t-ésimo corte $Q_t = Q(\{i_1, i_2, \ldots, i_t\}, \{i_{t+1}, \ldots, i_n\})$

[5] En la implementación realizada se usó una variante de *heapsort* llamada *non-recursive in-place heapsort*.

es tal que

$$\begin{aligned} Q_t &= \sum_{h=1}^{t}\sum_{k=t+1}^{n} \pi_{i_h} q_{i_h i_k} \\ &= \sum_{h=1}^{t-1}\sum_{k=t}^{n} \pi_{i_h} q_{i_h i_k} - \sum_{k=1}^{t-1} \pi_{i_k} q_{i_k i_t} + \sum_{k=t+1}^{n} \pi_{i_t} q_{i_t i_k} \\ &= Q_{t-1} - \sum_{k=1}^{t-1} \pi_{i_t} q_{i_t i_k} + \sum_{k=t+1}^{n} \pi_{i_t} q_{i_t i_k} \\ &= Q_{t-1} - \pi_{i_t} \left(\sum_{k=1}^{t-1} q_{i_t i_k} - \sum_{k=t+1}^{n} q_{i_t i_k} \right) \end{aligned}$$

con el caso base $Q_1 = \pi_{i_1} \sum_{k=2}^{n} q_{i_1 i_k}$. Esto implica que cada paso para actualizar Q_t toma únicamente $\Theta(1)$ (al igual que el cómputo de Q_1).

El valor de ϕ (ec. 4.6) es $\frac{Q_t}{\left(\sum_{k=1}^{t} \pi_{i_k}\right)\left(1-\sum_{k=1}^{t} \pi_{i_k}\right)}$, lo cual no aumenta la complejidad del cálculo porque cada factor en el denominador se actualiza en cada paso a un costo de $\Theta(1)$.

- Calcular las conductancias de cada lado del corte que fue escogido en el paso anterior toma $\Theta(n)$. Supóngase que los lados se denominan L y R respectivamente. Calcular por ejemplo la conductancia de L implica computar

$$\frac{1}{\pi(L)} \sum_{i \in L} \pi_i \sum_{j \in V \setminus L} q_{ij}$$

lo cual está en $\Theta(|L|)$ aplicando la suposición establecida en el inicio de esta sección. Así mismo la conductancia de R tomaría $\Theta(|R|)$ en tanto que el calculo general está en $\Theta(|L|) + \Theta(|R|)$.

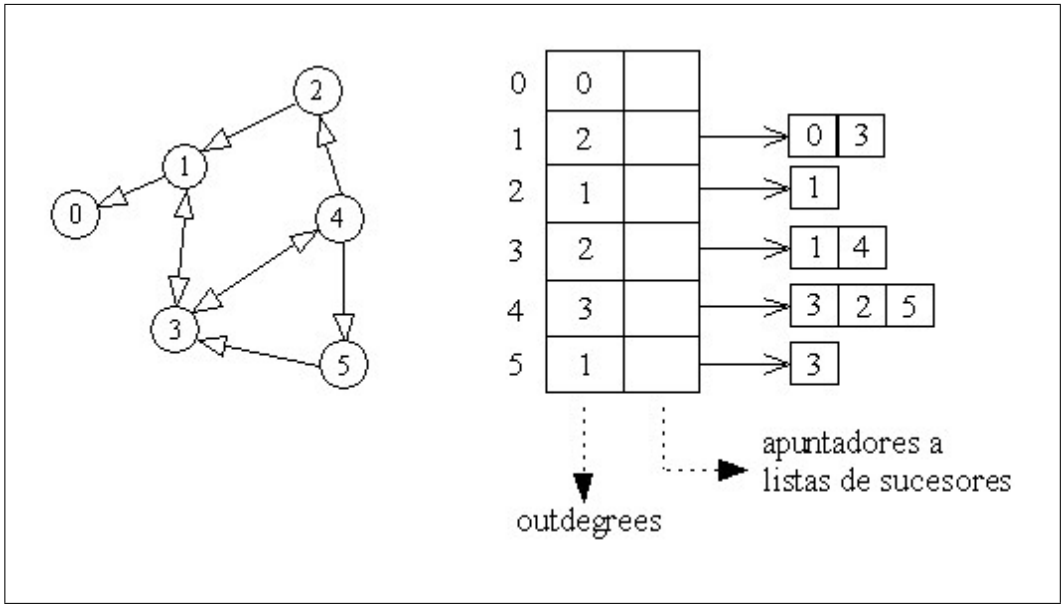

Figura 5.2: Representación en memoria de la matriz de adyacencias

El digrafo de ejemplo (izquierda) es representado como una lista de adyacencias (derecha). Los sucesores de cada nodo son alojados en arreglos de longitud variable.

De lo anterior se concluye que realizar una partición binaria toma $\Theta(n \log n)$ donde n es el tamaño del conjunto a biseccionar.

Como se ha visto, el acceso a las entradas q_{ij} es la operación clave en este problema y por esto es necesario asegurar un tiempo de acceso rápido a estos elementos[6]. Esto se puede lograr utilizando una representación como la que se muestra en la figura 5.2. Esta estructura de datos permite acceder al conjunto de los sucesores de un nodo particular a través un arreglo. El tamaño de dicho arreglo es claramente el grado del nodo en cuestión, es decir el *out-degree* de la página *web* que éste representa. En general el *out-degree* promedio de una página en la *web* actual está alrededor de 38 nodos ([13]), lo cual implica que este arreglo es en promedio una pequeña constante. Luego, para calcular una sumatoria como $\sum_{j \in A} q_{ij}$, se recorren uno o a uno los elementos del arreglo de sucesores de i y en caso que el elemento inspeccionado pertenezca a A, se puede aplicar la ec. 2.2 para averiguar el valor concreto de q_{ij}. Es así como cada evaluación de q_{ij} toma un tiempo constante y en general la sumatoria también porque requiere recorrer un arreglo cuyo tamaño es una constante independiente del tamaño de A.

Previa a la primera bisección sobre el conjunto V, es necesario realizar realizar los siguientes pasos:

- Construir Q (o más exactamente la representación en memoria de G, como se muestra en la figura 5.2), lo cual toma un tiempo proporcional a M
- Computar π usando Gauss-Seidel. Esta operación está en $\Theta(N \log N)$ (ver detalles en [9]).

Como el algoritmo de *clustering* espectral es aplicado recursivamente sobre cada lado y las condiciones de parada varían según la naturaleza del grafo, no es posible dar una estimación general acerca de la complejidad del algoritmo. Sin embargo se puede realizar el ejercicio de contar el número de pasos que toma procesar un grafo en el caso ideal en que las particiones estén balanceadas en tamaño y el algorimo no pare hasta alcanzar lados que tengan tamaño 1. En este escenario el primer corte tomaría $\Theta(N \log N)$, los dos cortes subsiguientes tomarían $\Theta(N \log \frac{N}{2})$ en total, y el razonamiento se puede seguir extendiendo lo que lleva a la siguiente sumatoria (asúmase que $k = \lg N$ es un número entero):

$$\begin{aligned}
\sum_{i=1}^{k} N \log \frac{N}{2^i} &= N \sum_{i=1}^{k} \left(\log N - \log \frac{1}{2^i} \right) \\
&= N \sum_{i=1}^{k} \left(\log N + \log 2^i \right) \\
&= N \left(k \log N + \sum_{i=1}^{k} i \log 2 \right) \\
&= N \left(\lg N \log N + \log 2 \sum_{i=1}^{k} i \right) \\
&= N \left[c_1 \lg^2 N + c_2 \lg N \, (1 + \lg N) \right]
\end{aligned}$$

En conclusión el algoritmo tomaría $\Theta(N \lg^2 N + M)$ pasos para procesar completamente el grafo.

En cuanto a los requerimientos de almacenamiento en memoria principal, se debe observar que tanto la representación del grafo y del vector π deben permanecer en memoria durante la ejecución de todo el algoritmo. En este sentido el espacio requerido está en $\Theta(M+N)$. El almacenamiento de otros vectores

[6] Nótese que representar de manera explícita esta matriz como un arreglo bidimensional no es factible por su gigantesco tamaño (p.e. una web de un millón de nodos requeriría más de 100GB de memoria).

tales como el del segundo vector propio tomado sobre el conjunto que se está cortando en cada paso, la permutación de índices generada por el ordenamiento de este vector y demás variables temporales requeridas en la implementación están en $O(N)$, en tanto que la complejidad arriba mencionada se mantiene.

Capítulo 6

Experimentación

6.1. Detalles preliminares

6.1.1. Fuentes de datos

Uno de los más grandes inconvenientes en el estudio del fenoméno del *web spam* ha sido la escasez de conjuntos de datos con los cuales realizar pruebas. Es así como la mayoría de autores han tenido que usar algún colector de documentos *web* (o inclusive codificar uno propio) para obtener porciones de la *web* y luego de esto, realizar etiquetado manual sobre las páginas. Esto constituye una ardua tarea que puede hacer tomar mucho más tiempo del que se espera en que se desarrolle un trabajo de investigación. Afortunadamente, en Junio de 2006 el Laboratorio de *Web* Algorithmics (Universita' degli Studi di Milano) puso disponible al público un par de conjunto de datos ([13]) constituídos por un significativo número de páginas (en el orden de millones) y etiquetas para un porcentaje significativo de estas. Dichos conjuntos de datos son de páginas pertenecientes al dominio de Internet .uk. En el cuadro 6.1 se muestran algunas de sus características más relevantes.

A partir de estos conjuntos de datos, que en adelante se llamarán superconjuntos, se definieron los conjuntos de datos con los cuales se realizó la experimentación en este trabajo (ver más adelante, en la sección 6.1.3).

Las etiquetas son de tres tipos en ambos superconjuntos (normal, *spam* o *suspicious*) y son asignadas sobre dominios completos y no sobre nodos particulares.

6.1.2. Implementación

El acceso a los grafos de los superconjuntos es realizando mediante un API de Java provista por sus autores [13], a través de la cual es posible obtener información tal como el grado o el conjunto de los sucesores de un nodo. Las funciones de esta API tienen que leer enormes archivos que se encuentran comprimidos[1], en tanto que las exigencias sobre CPU y acceso a disco son significativas. Por tanto, y en búsqueda de una implementación eficiente para el algoritmo de *clustering* espectral, se optó por escribir un programa el cual realizara el preprocesamiento sobre los datos de entrada de manera que el

[1] La técnica de compresión usada en este caso es WebGraph (ver más detalles en [11]).

Nombre del conjunto	Año de colección	Out-degree promedio	Nodos aprox.	Dominios de Internet	Dominios etiquetados
uk-2002	2002	16,09	$18{,}5 \times 10^6$	5345	5345
uk-2006	2006	38,14	$77{,}7 \times 10^6$	11402	8415

Cuadro 6.1: Conjuntos de datos de la colección de *web spam*

Nombre del conjunto	Superconjunto de origen	Número de nodos
3-uk2	uk-2002	10^3
3-uk6	uk-2006	10^3
4-uk2	uk-2002	10^4
4-uk6	uk-2006	10^4
5-uk2	uk-2002	10^5
5-uk6	uk-2006	10^5
6-uk2	uk-2002	10^6
6-uk6	uk-2006	10^6

Cuadro 6.2: Conjuntos de datos en la experimentación

programa principal (el cual se encarga únicamente de hacer el *clustering* espectral sobre un grafo dado) pudiera ser alimentado con la salida arrojada por el primero. Dicho programa de preprocesamiento (el cual fue obviamente codificado en Java) realiza las siguientes tareas:

- Carga alguno de los dos superconjuntos en memoria principal y realiza a partir de un nodo específico una exploración en anchura hasta obtener un conjunto (subgrafo) del tamaño deseado. Las páginas en dominios considerados como "*suspicious*" no eran exploradas e igualmente aquellas páginas que no fueron etiquetadas.
- Construye archivos binarios que representan la matriz de adyacencias, el vector de *out-degrees*, y el mapeo entre los identificadores de los nodos de el conjunto y el superconjunto. Dichos archivos son interpretables por el programa principal
- Construye un archivo de texto donde se encuentra la URL correspondiente a cada nodo

El programa principal (el cual fue construido en ANSI C con el objetivo de obtener tiempos de corrida en lo posible bajos) toma los archivos mencionados y efectúa el algoritmo de detección de *link spam*, generando una serie de archivos que contienen información acerca de cada bisección realizada.

Estos archivos de por sí no pueden aportar con resultados concretos a la investigación, en tanto que fue necesario realizar un proceso de posprocesamiento con el objetivo de obtener información consolidada y que permitiera realizar la evaluación de desempeño del algoritmo con respecto a los nodos que ya estaban previamente etiquetados. Este proceso fue realizado mediante un tercer programa construido en lenguaje Java.

Los experimentos se llevaron a cabo en un servidor Dell PowerEdge 6300 con 2.5GB de memoria RAM, cinco discos SCSI en un arreglo RAID nivel 0 y dos procesadores Intel Pentium III Xeon de 500MHz cada uno. El sistema operativo en que corrieron estos programas fue Windows 2000 Server.

6.1.3. Conjuntos de datos

A partir de los superconjuntos se extrajeron ocho conjuntos de datos como se muestra en el cuadro 6.2. Los conjuntos de datos provienen cuatro de cada superconjunto, y en tamaños desde los mil nodos y aumentando su tamaño diez veces progresivamente, hasta el mayor de tamaño un millón.

Los tiempos de colección de los conjunto de datos (realizar la búsqueda en anchura y escribir los archivos a disco) tomaron tiempos entre tres minutos (para los conjuntos menores) y seis horas (para los mayores). Dicho tiempo sigue una relación casi lineal con el número de nodos a colectar.

6.2. Resultados

6.2.1. Tratamiento de nodos positivos

El algoritmo de detección de *link spam* propuesto en este trabajo etiqueta subconjuntos de nodos (formados por al menos un nodo) como positivos (*spam*) o negativos (normal). Para poder establecer una comparación de los resultados obtenidos con las etiquetas ya conocidas que se encuentran asignadas a nivel de dominio y no de página, fue necesario llevar las etiquetas producidas al nivel de dominio de Internet. Para determinar si esto podría tener sentido, para cada conjunto de datos se extrajo un lista con los nombres de dominio que poseían al menos un nodo marcado como positivo. Sobre estos dominios, una rutina automática revisó todas las páginas que estaban marcadas como negativas (normales) y encontró que un gran procentaje de ellas (91 % en promedio, ver cuadro 6.3) estaban apuntando directamente a una página ya marcada como positiva en la lista (generalmente dentro de un dominio distinto). Esto aporta evidencia para poder generalizar y etiquetar un dominio completo como positivo si éste posee algún caso positivo. Como señala Becchetti et al., en muy pocos casos se observan mezclas de *spam* y de contenido normal en un mismo *host* [7]. Por otro lado este hecho tiene importante incidencia en la escalabilidad del algoritmo (ver al final del capítulo).

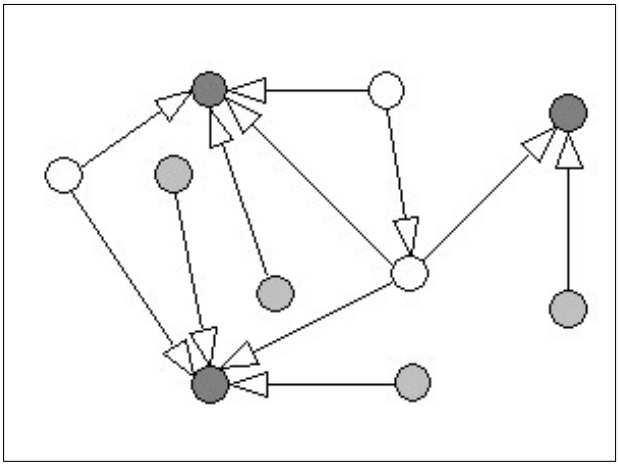

Figura 6.1: Objetivos de *spam* y sus dos tipos de páginas amplificadoras

Los nodos en gris oscuro representan dominios etiquetados como positivos (*spam*). Los nodos en gris claro son dominios de amplificación utilizados exclusivamente a un solo objetivo. Los nodos sin color corresponden a dominios desde donde se pueden crear links gratuitos sin control.

Aún con lo anterior el etiquetado está sesgado a favor de las páginas negativas en el sentido que el algoritmo tan sólo identifica los dominios donde hay amplificación de PageRank significativa, es decir, los "objetivos de colusión". Según lo expuesto en la sección 3.1.2 (ver figura 3.1), las páginas objetivo están siendo apuntadas por páginas amplificadoras (*boosting pages*), las cuales pueden ser propiedad de *spammers* (positivas) o no. De aquí que existe también la necesidad de ampliar el grupo de páginas positivas con aquellas páginas utilizadas para amplificar que son páginas desde las cuales no se puede "crear" un vínculo gratuito para favorecer a alguien.

Es posible identificar dominios desde lo cuales los *spammers* crean vínculos hacia sus páginas (*hijacked links*) observando que este tipo de páginas están abiertas a que cualquiera las use, y en este sentido, más de un *spammer* puede estarlas usufructuando (ver figura 6.1). Como aquellas páginas amplificadoras propiedad de un *spammer* sólo deben soportar un grupo específico (apuntar a un objetivo

Conjunto de datos	% en el mismo dominio	% en dominio distinto	Porcentaje total
3-uk2	2,1	89,0	91,1
3-uk6	1,0	90,1	91,1
4-uk2	3,2	73,9	77,1
4-uk6	0,9	91,4	92,3
5-uk2	1,1	90,1	91,2
5-uk6	0,8	88,9	89,7
6-uk2	0,9	96,3	97,2
6-uk6	0,6	97,4	98,0

Cuadro 6.3: Porcentaje de páginas dentro de dominios con páginas positivas que apuntan a páginas positivas directamente

particular) y no a varios con el objetivo de conseguir más ganancia, la manera de conocer los restantes dominios que también son positivos es tomar aquellos que están apuntando de manera exclusiva a uno y solo un dominio positivo. Nótese que esta estrategia permite entonces identificar si se lo quisiera, los *blogs*, *wikis* o cualquier otro tipo de recurso que está siendo aprovechado indiscriminadamente.

En la práctica cuando se identificaron estos nuevos dominios positivos, se observó que en todos los conjuntos de datos, dichos dominios estaban compuestos por muy pocas páginas, generalmente una sola la cual tenía *in-degree* cero.

6.2.2. Evaluación

En este apartado se describe el desempeño que tuvo el algoritmo en los experimentos descritos. La comparación se establece con respecto a las etiquetas ya preestablecidas en los superconjuntos. A continuación se muestran las matrices de confusión para cada experimento:

3-uk2		
	algoritmo	
superconjunto	*spam*	normal
spam	1	1
normal	1	14

3-uk6		
	algoritmo	
superconjunto	*spam*	normal
spam	2	0
normal	2	10

4-uk2		
	algoritmo	
superconjunto	*spam*	normal
spam	2	3
normal	2	45

4-uk6		
	algoritmo	
superconjunto	*spam*	normal
spam	12	1
normal	10	43

5-uk2		
	algoritmo	
superconjunto	*spam*	normal
spam	16	4
normal	6	100

5-uk6		
	algoritmo	
superconjunto	*spam*	normal
spam	14	2
normal	5	85

6-uk2		
	algoritmo	
superconjunto	*spam*	normal
spam	41	4
normal	8	280

6-uk6		
	algoritmo	
superconjunto	*spam*	normal
spam	40	9
normal	24	233

Conjunto de datos	Número de dominios	Recall (%)	Precisión (%)
3-uk2	17	50,0	50,0
3-uk6	12	100,0	50,0
4-uk2	52	40,0	50,0
4-uk6	66	92,3	54,5
5-uk2	126	80,0	72,7
5-uk6	106	87,5	73,6
6-uk2	333	91,1	83,6
6-uk6	306	81,6	62,5

Cuadro 6.4: Medidas de error en los experimentos

El cuadro 6.4 muestra un resumen del desempeño del algoritmo como clasificador al igual que el total de dominios para cada conjunto de datos. Las medidas muestran por ejemplo, que para los cuatro conjuntos de datos mayores, el valor de recall está entre 80 y 90 %, y el de precisión está generalmente sobre 70 %. Dichas cifras son comparables con los resultados de los más efectivos trabajos en detección de *link spam* hasta ahora realizados ([7], [16], [26], [39]), que reportan valores de recall y precisión alrededor de 80 %.

Los valores de precisión generalmente inferiores a los de recall, indican una alta presencia de falsos positivos, esto es, páginas normales que son clasificadas como *spam*. Esto puede estar siendo causado por tres factores:

- Existen dominios en los que hay a la vez páginas positivas y negativas, aunque finalmente quedan etiquetadas todas como positivas (ver el cuadro 6.3)
- Algunas páginas negativas que apuntan a páginas objetivo pueden estar siendo marcadas como positivas ya que el conjunto de datos puede no contener las demás páginas que también están aprovechando la posibilidad de robarse *links* desde aquella. Por ejemplo, un foro puede estar siendo explotado por varios dominios de Internet pero dentro el conjunto de datos tan sólo quedó incluido uno de ellos.
- Pueden existir comunidades que se referencian mutuamente y están siendo marcadas positivas por el hecho de estar "reteniendo" PageRank, aunque no exista la intención evidente de hacerlo.

Este último hecho puede ser una de las grandes debilidades del algoritmo pues puede confundir páginas de calidad con *spam* y no hay manera de corregirlo a menos que se use, por ejemplo, información extra como los contenidos, direcciones IP, composición de URLs, etc., para mejorar el discernimiento.

6.2.3. *Spam* más significativo

Los hechos expuestos en este apartado se derivan de la observación del comportamiento del algoritmo para el conjunto 6-uk6, uno de los más significativos por estar reflejando un situación más fiel a la *web* de hoy y tener un tamaño relativamente alto. Así mismo los hechos están descritos a nivel de nodos individuales, esto es, antes de considerar cada dominio de Internet como una sola entidad.

El cuadro 6.5 presenta para el conjunto de datos 6-uk6, los diez subconjuntos cuya amplificación se encontró como la más alta (conductancia más baja) dentro de todos los subconjuntos declarados como *spam*. Se reporta el *in-degree* de cada subconjunto, los nodos que lo componen (URLs), y en caso de ser un solo nodo, el rango de PageRank al que pertenece (en el posprocesamiento de los resultados, cada nodo es asignado a una "cubeta" de acuerdo a su PageRank)[2].

[2]Se utilizaron cubetas que dividen el rango de PageRank en cién partes iguales (entre el menor y el mayor valor). No se utilizó por ejemplo el índice dentro del ordenamiento de las páginas de acuerdo al PageRank dado que generalmente muchas se encuentran empatadas en valor y varios índices serían válidos para la misma página.

URLs	Conductancia	Rango PR	In-degree
casino69.co.uk	0,185	99	796
www.lesbian-nudes.co.uk	0,198	99	816
sexmole.co.uk/preview.shtml	0,201	98	766
loans.pra-mortgages.co.uk	0,212	96	801
www.avs-sex.co.uk/public-zone/main.htm	0,220	93	679
www.uk-online-finance.co.uk www.uk-online-shopping.co.uk	0,226	–	1272
www.gaydvds.co.uk/join.htm	0,231	82	467
www.games-books-cds-dvds-videos.co.uk	0,233	82	436
www.sex-with-swingers.co.uk www.sexy-big-boobs.co.uk	0,236	–	963
www.gamble-seek.co.uk www.casino-gambling-universe.co.uk	0,242	–	844

Cuadro 6.5: Subconjuntos de nodos con la conductancia más baja para 6-uk6

Como era de esperarse, la mayoría de estos subconjuntos son de tamaño uno (correspondiendo a *single-target link farms*). Adicionalmente tienen *in-degrees* muy altos[3] y la mayoría de URLs reflejan correspondencia con negocios de pornografía, apuestas en línea, asuntos financieros y descarga de archivos multimediales. Usualmente son las páginas raíz del dominio en que se encuentran.

Nótese que para el caso de *single-target link farms* una menor conductancia no necesariamente implica mayor *in-degree*, lo que significa que en dichos casos se está sacando provecho de páginas que tienen mejores valores de PageRank, generando mayores ganancias.

Adicionalmente, todos los dominios a los que pertenecían las páginas del cuadro estaban etiquetados como *spam* dentro del superconjunto al que pertenecían y generalmente tenían *out-degree* igual a cero.

6.2.4. Consumo de recursos de cómputo

En lo que concierne a PR, los cálculos de π para todos los conjuntos de datos fueron realizados usando $d = 0{,}85$. El cuadro 6.6 muestra el número de iteraciones requeridas en cada caso y el tiempo de corrida.

En lo que respecta al programa principal (el cual incluye también el cómputo de PR), se obtuvieron los tiempos de corrida que se muestran en el cuadro 6.7. Adicionalmente se incluye el tamaño máximo de memoria que fue utilizado por el programa en algún momento de su ejecución para cada uno de los conjuntos de datos.

Se puede apreciar que los tiempos de corrida son superiores para los conjuntos de datos provenientes del superconjunto uk-2006 con respecto a aquellos provenientes de uk-2002, especialmente para valores de N grandes. Esto se puede atribuir al hecho que el grafo de uk-2006 es más denso que el de uk-2002, y por tanto las operaciones que implican computar el flujo de un nodo hacia un suconjunto arbitrario

[3] Todas las páginas mostradas que corresponden a subconjuntos de tamaño uno tiene un indegree por encima del 99 % de las páginas del conjunto

Conjunto de datos	Número de iteraciones	Tiempo total aprox. (s)	Promedio por iteración (s)
3-uk2	48	0,1	0,002
3-uk6	44	0,1	0,002
4-uk2	46	1,9	0,041
4-uk6	38	1,4	0,036
5-uk2	41	20,0	0,487
5-uk6	45	21,3	0,473
6-uk2	48	266,3	5,547
6-uk6	42	240,1	5,716

Cuadro 6.6: Iteraciones y tiempos de corrida para el cómputo de PR

Conjunto de datos	Tiempo total (horas)	Máximo de memoria (MB)
3-uk2	0,05	0,9
3-uk6	0,05	1,5
4-uk2	0,19	3,2
4-uk6	0,17	6,1
5-uk2	1,75	90,0
5-uk6	3,69	197,9
6-uk2	26,78	899,8
6-uk6	32,76	2017,6

Cuadro 6.7: Consumo de memoria y CPU del programa principal

toman un tiempo mayor.

Para hallar una estimación del consumo de recursos de cómputo que este programa tomaría sobre un grafo de un tamaño comparable al que maneja un motor de búsqueda convencional en la actualidad (decena de miles de millones de nodos [24]), asúmase que el tiempo de corrida es de la forma $c_1 N \log^2 N$ y el consumo de memoria es $c_2 N$ donde c_1 y c_2 son constantes reales. Tomando los cuatro conjuntos de datos provenientes del superconjunto uk-2006, el valor de c_1 oscilaría entre $0{,}91 \times 10^{-6}$ y $5{,}5 \times 10^{-6}$ (siendo el menor valor el que corresponde a $N = 10^6$). Por otro lado el valor para c_2 sería un consistente 9×10^{-4} a excepción del caso $N = 10^4$ en el cual su valor sería $3{,}2 \times 10^{-4}$. Haciendo $N = 10^{10}$ se obtendrían tiempos de corrida del orden del millón de horas (alrededor de 114 años), lo cual es claramente no factible, y el almacenamiento estaría aproximadamente en 8,5TB!.

De lo anterior se concluye que este método solo puede ser llevado a la práctica si cada dominio de Internet se colapsa como un sólo nodo, lo cual, según lo visto previamente no es una simplificación muy fuerte. En este caso, el número de nodos del grafo estaría alrededor de algunos millones de nodos lo cual es factible con las capacidades de cómputo contemporáneas.

Capítulo 7

Conclusiones

Los motores de búsqueda se han convertido en un punto de entrada preferido para los usuarios de Internet. Esta situación ha provocado el advenimiento de numerosas técnicas de *spam* para engañar a estas importantes herramientas y apoderarse de posiciones privilegiadas en cuanto a *rankings* corresponde. Esto claramente representa una ventaja comercial para los *spammers* y molestias para los usuarios normales que perciben cómo la calidad de los resultados arrojados está siendo deteriorada.

Como es de suponer, las técnicas empleadas por los motores de búsqueda para combatir el *web spam* se encuentran en el terreno del secreto comercial. Lo único que ha podido decirse al respecto es que por las dificultades que aún experimentan los usuarios finales en sus búsquedas por la presencia de *spam*, dichos métodos aún no están lo suficientemente maduros.

En este trabajo se propuso una técnica de detección de *link spam* basada en la conductancia, un concepto nunca antes usado en este campo, con resultados muy favorables. Así mismo se mostró la interesante analogía entre la conductancia y el *clustering* espectral, y su novedosa aplicación a los grafos dirigidos gracias al mecanismo de la reversibilización en cadenas de Markov con la cual se logra conservar la distribución estacionaria, el flujo, y por ende la conductancia. Hasta donde se conoce, este es el primer trabajo en aplicar algún tipo de *clustering* sobre la estructura de vínculos (digrafo) de la *web* con el objetivo de estudiar el *web spam*.

Los resultados experimentales corroboran el significado teórico que le fue dado a la conductancia en el contexto del *link spam*, mostrando resultados comparables a los mejores procedimientos de detección existentes. Igualmente, sin requerir de ningún tipo de intervención humana (como en [26] o [39]) o incluir otro tipo de información que no sean los vínculos entre páginas (como en [26]), el método se hace adecuado para su aplicación a mayor escala como se vio en la sección 6.2.4.

Por ser un método que toma en cuenta la fuente principal de ganancia en *ranking* de los *spammers* (los *inlinks*), su robustez está garantizada, esto es, un *spammer* no podrá esconderse del algoritmo a menos que modifique la forma en que otros le apuntan disminuyendo su propio PageRank, quitándole el estatus de *spam* a sus propias páginas.

Una de las desventajas del algoritmo es que de por sí únicamente puede identificar las páginas objetivo (*link farm cores*). Sin embargo, en la sección 6.2.1 se describe una técnica para encontrar las páginas amplificadoras que también son *spam*. Adicionalmente se señaló el caso de las páginas que aunque pueden tener alta amplificación no son *spam*. Al respecto no parece haber nada que lo solucione sin tener que incluir mayor información al modelo.

Por último, se preven dos alternativas de trabajo futuro sobre este algoritmo para reducir su tiempo de corrida. Por un lado, aplicar algún método que permita computar el segundo vector propio de manera más rápida, o un método que permita obtener directamente el ordenamiento de índices, ya que en sí el vector no se utiliza. Esto no sólo favorecería a este problema sino en general a todo lo que el *clustering* espectral atañe. Al respecto Flake et al. [20] y Leighton et al. [32] parecen revelar caminos interesantes. Por otro lado, el aprovechamiento del paralelismo en el sentido que cada corte establece dos lados independientes que en sucesivas iteraciones no se influencian de ninguna manera

de modo que el algoritmo se podría distribuir sobre varias máquinas interconectadas.

Bibliografía

[1] AltaVista Company. The AltaVista Search Engine. http://www.altavista.com.

[2] Google Inc. Google Search Engine. http://www.google.com.

[3] S. Adali, T. Liu, and M. Magdon-Ismail. Optimal link bombs are uncoordinated. In *First International Workshop on Adversarial Information Retrieval on the Web*, 2005.

[4] N. Ailon, S. Chien, and C. Dwork. On clusters in markov chains. Technical report, Microsoft Research, 2004.

[5] E. Amitay, D. Carmel, A. Darlow, R. Lempel, and A. Soffer. The connectivity sonar: Detecting site functionality by structural patterns. In *ACM Conference on Hypertext And Hypermedia*, pages 38–47, 2003.

[6] R. Baeza-Yates, C. Castillo, and V. López. Pagerank increase under different collusion topologies. In *Workshop on Adversarial Information Retrieval on the Web*, 2005.

[7] L. Becchetti, C. Castillo, D. Donato, S. Leonardi, and R. Baeza-Yates. Using rank propagation and probabilistic counting for link-based spam detection. Technical report, DELIS (Dynamically Evolving, Large-Scale Information Systems), 2006.

[8] I. Benjamini, G. Kozma, L. Lovász, D. Romik, and G. Tardos. Waiting for a bat to fly by (in polynomial time). *Combinatorics, Probability and Computing*, 15(5):673–683, 2006.

[9] P. Berkhin. A survey on PageRank computing. *Internet Mathematics*, 22(1):73–120, 2005.

[10] K. Bharat and M. R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *21st ACM SIGIR Conference on Research and Development in Information Retrieval*, 1998.

[11] P. Boldi and S. Vigna. The WebGraph framework i: Compression techniques. Technical Report 293-03, Universit di Milano, Dipartimento di Scienze dell'Informazione, 2003.

[12] S. Brin and L. Page. Anatomy of a large-scale hypertextual web search engine. In *World Wide Web Conference*, 1998.

[13] C. Castillo, D. Donato, L. Becchetti, P. Boldi, M. Santini, and S. Vigna. A reference collection for web spam. *SIGIR Forum*, 40(2), 2006.

[14] D. Cheng, R. Kannan, S. Vempala, and G. Wang. On a recursive spectral algorithm for clustering from pairwise similarities. Technical Report MIT-LCS-TR-906, MIT, 2003.

[15] B. D. Davison. Recognizing nepotistic links on the web. In *AAAI Workshop on Artificial Intelligence for Web Search*, 2000.

[16] I. Drost and T. Scheffer. Thwarting the nigritude ultramarine: Learning to identify link spam. In *European Conference on Machine Learning*, 2005.

[17] M. Dyer, L. A. Goldberg, and M. Jerrum. Markov chain comparison. *Probability surveys*, 3:89–111, 2006.

[18] N. Eiron, K. S. McCurley, and J. A. Tomlin. Ranking the web frontier. In *World Wide Web Conference*, 2004.

[19] D. Fetterly, M. Manasse, and M.Ñajork. Spam, damn spam, and statistics: Using statistical analysis to locate spam web pages. In *International Workshop on the Web and Databases*, June 2004.

[20] G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1(4):385–408, 2004.

[21] H. Garcia-Molina and Z. Gyöngyi. Link spam alliances. Technical report, Computer Science Department, Stanford University, March 2, 2005.

[22] A. Goel, H. Zhang, R. Govindan, K. Mason, and B. V. Roy. Making eigenvector-based reputation systems robust to collusion. In *Workshop on Algorithms and Models for the Web Graph*, October 2004.

[23] G. H. Golub and C. F. V. Loan. *Matrix Computations*. The Johns Hopkins University Press, 3ł edición edition, 1996.

[24] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *14th International Conference on World Wide Web*, pages 902–903, 2005.

[25] Z. Gyöngyi and H. Garcia-Molina. Web spam taxonomy. Technical report, Computer Science Department, Stanford University, 2005.

[26] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with TrustRank. In *30th Very Large Databases Conference*, 2004.

[27] T. Haveliwala and S. Kamvar. The second eigenvalue of the google matrix. Technical report, Stanford University, 2003.

[28] M. Henzinger, R. Motwani, and C. Silverstein. Challenges in web search engines. *ACM SIGIR Forum*, 36(2), 2002.

[29] M. Jerrum and A. Sinclair. Conductance and the rapid mixing property for markov chains: The approximation of the permanent resolved. In *20th Annual ACM Symposium on Theory of Computing (STOC 1988)*, pages 235–243, 1988.

[30] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Symposium on Discrete Algorithms*, 1998.

[31] S. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling emerging cyber communities automatically. In *8th World Wide Web Conference*, 1999.

[32] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

[33] U. V. Luxburg, M. Belkin, and O. Bousquet. Consistency of spectral clustering. Technical Report TR-134, Max Planck Institute for Biological Cybernetics, September 2004.

[34] P. T. Metaxas and J. DeStefano. Web spam, propaganda and trust. In *1st International Workshop on Adversarial Information Retrieval on the Web*, 2005.

[35] R. Montenegro and P. Tetali. Mathematical aspects of mixing times in markov chains. *Foundations and Trends in Theoretical Computer Science*, 1(3):237–354, 2006.

[36] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[37] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 2000.

[38] D. Spielman and S. Teng. Spectral partitioning works: Planar graphs and finite element meshes. In *37th Annual Symposium on Foundations of Computer Science*, 1996.

[39] B. Wu and B. D. Davison. Identifying link farm spam pages. In *World Wide Web Conference*, 2005.

[40] B. Wu and B. D. Davison. Cloaking and redirection: A preliminary study. In *1st International Workshop on Adversarial Information Retrieval on the Web*, May 10, 2005.

Apéndice A

Hechos relevantes de algebra lineal

A.1. Hecho 1

Sea f un vector tal que $f \in \{-1, 1\}^n$, $W = (w_{ij})$ una matrix simétrica y D una matriz diagonal tal que $d_{ii} = \sum_j w_{ij}$, se tiene que

$$f^T(D - W)f = 4 \sum_{(i:f_i=1)} \sum_{(j:f_j=-1)} w_{ij}$$

Demostración:

$$\begin{aligned}
f^T(D-W)f &= \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}^T \begin{bmatrix} \sum_j w_{1j} - w_{11} & -w_{12} & \dots & -w_{1n} \\ -w_{21} & \sum_j w_{2j} - w_{22} & \dots & -w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -w_{n1} & -w_{n2} & \dots & \sum_j w_{nj} - w_{nn} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} \\
&= \begin{bmatrix} f_1 \sum_j w_{1j} - f \cdot w_{,1} \\ f_2 \sum_j w_{2j} - f \cdot w_{,2} \\ \dots \\ f_n \sum_j w_{nj} - f \cdot w_{,n} \end{bmatrix}^T \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} \\
&= \sum_i f_i \left(f_i \sum_j w_{ij} - f \cdot w_{.i} \right) = \sum_i \left(f_i^2 \sum_j w_{ij} - f_i \sum_j f_j w_{ji} \right) \\
&= \sum_i \sum_j f_i^2 w_{ij} - \sum_i \sum_j f_i f_j w_{ij} = \sum_i \sum_j \left(f_i^2 - f_i f_j \right) w_{ij}
\end{aligned}$$

donde $u \cdot v$ denota el producto punto entre los vectores u y v, y $a_{.j}$ hace referencia a la j-ésima columna de la matriz A.

La última expresión puede ser escrita como

$$\sum_{(i:f_i=1)} \sum_{(j:f_j=1)} (0) w_{ij} + \sum_{(i:f_i=1)} \sum_{(j:f_j=-1)} 2w_{ij} + \sum_{(i:f_i=-1)} \sum_{(j:f_j=1)} 2w_{ij} + \sum_{(i:f_i=-1)} \sum_{(j:f_j=-1)} (0) w_{ij},$$

luego $f^T(D - W)f = 4 \sum_{(i:f_i=1)} \sum_{(j:f_j=-1)} w_{ij}$.

Igualmente se puede demostrar que $f^T Df = \sum_i \sum_j w_{ij}$ y $f^T D\mathbf{1} = \sum_{(i,j:f_i=1)} w_{ij} - \sum_{(i,j:f_i=-1)} w_{ij}$:

$$\begin{aligned} f^T Df &= \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}^T \begin{bmatrix} \sum_j w_{1j} & 0 & \cdots & 0 \\ 0 & \sum_j w_{2j} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sum_j w_{nj} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} \\ &= \begin{bmatrix} f_1 \sum_j w_{1j} \\ f_2 \sum_j w_{2j} \\ \cdots \\ f_n \sum_j w_{nj} \end{bmatrix}^T \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} \\ &= \sum_i f_i^2 \sum_j w_{ij} = \sum_i \sum_j w_{ij} \end{aligned}$$

$$\begin{aligned} f^T D\mathbf{1} &= \begin{bmatrix} f_1 \sum_j w_{1j} \\ f_2 \sum_j w_{2j} \\ \cdots \\ f_n \sum_j w_{nj} \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \\ &= \sum_i f_i \sum_j w_{ij} = \sum_i \sum_j f_i w_{ij} \\ &= \sum_{(i,j:f_i=1)} w_{ij} + \sum_{(i,j:f_i=-1)} (-1) w_{ij} \\ &= \sum_{(i,j:f_i=1)} w_{ij} - \sum_{(i,j:f_i=-1)} w_{ij} \end{aligned}$$

A.2. Hecho 2

Sea A una matriz con valores propios reales, la matriz $cI + dA$ (donde c y d son constantes reales y $d \neq 0$) tiene todos sus valores propios de la forma $d\lambda + c$ donde λ es valor propio de A:

$$\begin{aligned} (cI + dA)v &= \lambda' v \\ cv + dAv &= \lambda' v \\ dAv &= (\lambda' - c)v \\ Av &= \frac{\lambda' - c}{d} v \end{aligned}$$

A.3. Hecho 3

Sea $W = (w_{ij})$ una matriz simétrica con entradas reales positivas y D una matriz diagonal definida según $d_{ii} = \sum_j w_{ij}$, el sistema generalizado $(D - W)f = \lambda Df$ equivale al sistema $Az = \lambda z$ donde $z = D^{1/2} f$ y $A = D^{-1/2}(D - W)D^{-1/2}$:

$$\begin{aligned} (D - W)f &= \lambda Df \\ (D - W)D^{-1/2} z &= \lambda D D^{-1/2} z \\ D^{-1/2}(D - W)D^{-1/2} z &= \lambda z. \end{aligned}$$

Los valores λ (que en ambos sistemas son los mismos) son reales no negativos ya que la matriz $D^{-1/2}(D - W)D^{-1/2}$ es semidefinida positiva [37].

Por otro lado, $D^{1/2}\mathbf{1}$ es el vector propio de A que corresponde al valor propio cero, es decir $AD^{1/2}\mathbf{1} = \mathbf{0}$:

$$\begin{aligned} D^{-1/2}(D-W)D^{-1/2}D^{1/2}\mathbf{1} &= \mathbf{0} \\ D^{-1/2}(D-W)\mathbf{1} &= \mathbf{0} \\ D^{1/2}\mathbf{1} &= D^{-1/2}W\mathbf{1} \\ \mathbf{1} &= D^{-1}W\mathbf{1}. \end{aligned}$$

Este último hecho es cierto porque la fila i-ésima de $D^{-1}W\mathbf{1}$ contiene la suma de los valores de la fila i de la matriz $D^{-1}W$ lo cual es uno: $\sum_j d_{ii}^{-1} w_{ij} = d_{ii}^{-1} \sum_j w_{ij} = d_{ii}^{-1} d_{ii} = 1$.

Ahora, de acuerdo con las propiedades del cociente de Rayleigh ([23]), si se tiene una matriz real simétrica C, bajo la restricción de que x es un vector ortogonal al primer vector propio, el cociente $\frac{x^T C x}{x^T x}$ es minimizado por el vector propio correspondiente al segundo valor propio más pequeño de C. Por tanto el vector que minimiza la expresión

$$\underset{z^T D^{1/2}\mathbf{1}=0}{\arg\min} \frac{z^T A z}{z^T z}$$

es el segundo vector propio de A.

La expresión anterior es equivalente a $\underset{f^T D\mathbf{1}=0}{\arg\min} \left[\frac{f^T(D-W)f}{f^T D f}\right]$:

$$\begin{aligned} \underset{f^T D\mathbf{1}=0}{\arg\min} \left[\frac{f^T(D-W)f}{f^T D f}\right] &\equiv \underset{(D^{-1/2}z)^T D\mathbf{1}=0}{\arg\min} \left[\frac{(D^{-1/2}z)^T(D-W)(D^{-1/2}z)}{(D^{-1/2}z)^T D(D^{-1/2}z)}\right] \\ &\equiv \underset{z^T D^{-1/2} D\mathbf{1}=0}{\arg\min} \left[\frac{z^T D^{-1/2}(D-W)D^{-1/2}z}{z^T D^{-1/2} D D^{-1/2} z}\right] \\ &\equiv \underset{z^T D^{1/2}\mathbf{1}=0}{\arg\min} \left[\frac{z^T D^{-1/2}(D-W)D^{-1/2}z}{z^T z}\right] \end{aligned} \tag{A.1}$$

Por lo anterior el vector que minimiza A.1 es el vector correspondiente al segundo menor valor propio de $(D-W)f = \lambda D f$.

Apéndice B

Hechos relevantes de cadenas de Markov

B.1. Simetría del flujo

Sea $\Omega = \{1, 2, \ldots\}$ el espacio de estados de una cadena de Markov ergódica, a continuación se demuestra que $Q(A, \Omega \backslash A) = Q(\Omega \backslash A, A)$ para cualquier subconjunto A (no vacío) de Ω:

1. Sean $B, C \subseteq \Omega$ subconjuntos disyuntos no vacíos nótese que $Q(A, B \cup C) = Q(A, B) + Q(A, C)$ y $Q(B \cup C, A) = Q(B, A) + Q(C, A)$

$$\sum_{i \in A, j \in B} \pi_i p_{ij} + \sum_{i \in A, j \in C} \pi_i p_{ij} = \sum_{i \in A, j \in B \cup C} \pi_i p_{ij}$$

$$\sum_{i \in B, j \in A} \pi_i p_{ij} + \sum_{i \in C, j \in A} \pi_i p_{ij} = \sum_{i \in B \cup C, j \in A} \pi_i p_{ij}$$

2. Para cualquier k, $Q(\Omega, \{k\}) = Q(\Omega, \{k\})$

$$\begin{aligned} \sum_{i \in \Omega} \pi_i p_{ik} &= \pi_k \\ \sum_{j \in \Omega} \pi_k p_{kj} &= \pi_k \sum_{j \in \Omega} p_{kj} = \pi_k \end{aligned}$$

3. $Q(\Omega, A) = Q(A, \Omega)$ (asumiendo que A se compone de la forma $\{a_1, a_2, \ldots, a_{|A|}\}$)

$$\begin{aligned} Q(\Omega, A) &= Q(\Omega, \{a_1, a_2, \ldots, a_{|A|}\}) \\ &= Q(\Omega, \{a_1\}) + Q(\Omega, \{a_2, \ldots, a_{|A|}\}) \\ &= \sum_{i=1}^{|A|} Q(\Omega, \{a_i\}) \end{aligned}$$

$$\begin{aligned} Q(A, \Omega) &= Q(\{a_1, a_2, \ldots, a_{|A|}\}, \Omega) \\ &= Q(\{a_1\}, \Omega) + Q(\{a_2, \ldots, a_{|A|}\}, \Omega) \\ &= \sum_{i=1}^{|A|} Q(\{a_i\}, \Omega) = \sum_{i=1}^{|A|} Q(\Omega, \{a_i\}) \end{aligned}$$

4. Finalmente para completar la prueba

$$\begin{aligned} Q(\Omega, A) &= Q(A, \Omega) \\ Q(\Omega\backslash A \cup A, A) &= Q(A, \Omega\backslash A \cup A) \\ Q(\Omega\backslash A, A) + Q(A, A) &= Q(A, \Omega\backslash A) + Q(A, A) \\ Q(\Omega\backslash A, A) &= Q(A, \Omega\backslash A) \end{aligned}$$

B.2. Distribución estacionaria y conductancia al aplicar reversibilización

Sea una cadena de Markov cuya matriz de transición R está dada por $r_{ij} = \frac{1}{2}\left(q_{ij} + \frac{\pi_j}{\pi_i} q_{ji}\right)$ donde $Q = (q_{ij})$ es la matriz de transición de una cadena de Markov ergódica con distribución estacionaria π, se tiene que ambas cadenas poseen la misma distribución estacionaria y la conductancia es equivalente para cualquier subconjunto del espacio de estados.

La primer condición básica a satisfacer es $\sum_j r_{ij} = 1$ para cualquier i:

$$\begin{aligned} \sum_j \frac{1}{2}\left(q_{ij} + \frac{\pi_j}{\pi_i} q_{ji}\right) &= \frac{1}{2}\sum_j q_{ij} + \frac{1}{2}\sum_j \frac{\pi_j}{\pi_i} q_{ji} \\ &= \frac{1}{2} + \frac{1}{2\pi_i}\sum_j \pi_j q_{ji} \\ &= \frac{1}{2} + \frac{1}{2\pi_i}\pi_i = 1 \end{aligned}$$

Si $\pi^T = \pi^T R$, para todo j se cumpliría que $\sum_i \pi_i r_{ij} = \pi_j$, ahora

$$\begin{aligned} \sum_i \pi_i r_{ij} &= \sum_i \pi_i \frac{1}{2}\left(q_{ij} + \frac{\pi_j}{\pi_i} q_{ji}\right) = \frac{1}{2}\sum_i \pi_i q_{ij} + \frac{1}{2}\sum_i \pi_j q_{ji} \\ &= \frac{1}{2}\pi_j + \frac{1}{2}\pi_j \sum_i p_{ji} = \frac{1}{2}\pi_j + \frac{1}{2}\pi_j = \pi_j \end{aligned}$$

En vista que π se preserva, para la conductancia basta con verificar que el flujo $Q(A, \Omega\backslash A)$ es equivalente:

$$\begin{aligned} \sum_{i\in A, j\in\Omega\backslash A} \pi_i r_{ij} &= \frac{1}{2}\sum_{i\in A, j\in\Omega\backslash A} \pi_i \left(q_{ij} + \frac{\pi_j}{\pi_i} q_{ji}\right) \\ &= \frac{1}{2}\left[\sum_{i\in A, j\in\Omega\backslash A} \pi_i q_{ij} + \sum_{i\in A, j\in\Omega\backslash A} \pi_j q_{ji}\right] \\ &= \frac{1}{2}\left[Q(A, \Omega\backslash A) + Q(\Omega\backslash A, A)\right] = Q(A, \Omega\backslash A) \end{aligned}$$

Ahora suponga una nueva cadena de Markov tal que su matriz de transición X es dada según $x_{ij} = \frac{1}{2}(\delta_{ij} + r_{ij})$ donde δ es la función delta de Kronecker. Es posible demostrar que la distribución

estacionaria se sigue conservando:

$$\begin{aligned}\sum_i \pi_i x_{ij} &= \frac{1}{2}\sum_i \pi_i \left(\delta_{ij} + r_{ij}\right) \\ &= \frac{1}{2}\left(\sum_i \pi_i \delta_{ij} + \sum_i \pi_i r_{ij}\right) \\ &= \frac{1}{2}\pi_j + \frac{1}{2}\pi_j = \pi_j\end{aligned}$$

Esta vez el flujo es disminuido a su mitad:

$$\begin{aligned}\sum_{i\in A, j\in\Omega\backslash A} \pi_i x_{ij} &= \frac{1}{2}\sum_{i\in A, j\in\Omega\backslash A} \pi_i \left(\delta_{ij} + r_{ij}\right) \\ &= \frac{1}{2}\left[\sum_{i\in A, j\in\Omega\backslash A} \pi_i \delta_{ij} + \sum_{i\in A, j\in\Omega\backslash A} \pi_i r_{ij}\right] \\ &= \frac{1}{2}\left[0 + Q(A, \Omega\backslash A)\right] = \frac{1}{2}Q(A, \Omega\backslash A)\end{aligned}$$

lo cual es un efecto obvio por haber añadido un "*loop*" sobre cada estado, lo cual dificulta el paso de un estado hacia otro distinto sobre el camino aleatorio.

Apéndice C

Algoritmos iterativos sobre matrices

C.1. Método de potencias

Sea A una matriz real arbitraria de tamaño $n \times n$ con vectores propios x_i tal que

$$Ax_i = \lambda_i x_i$$

para $i = 1, 2, \ldots, n$ donde

$$|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|.$$

El siguiente algoritmo llamado método de potencias calcula aproximaciones sucesivas $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ del vector x_1:

Algorithm 4 *Power Method*
Sea $u^{(0)}$ *un vector arbitrario tal que* $||u^{(0)}||_2 = 1$
$k \longleftarrow 0$
Repetir
$u^{(k+1)} = Au^{(k)}$
$u^{(k+1)} = u^{(k+1)}/||u^{(k+1)}||_2$
$k \longleftarrow k + 1$
Hasta alcanzar convergencia

Este método puede ser empleado para calcular el segundo vector propio (en módulo) de A[1] de la siguiente forma:

Algorithm 5 *Power Method 2*
Sea c *la primera componente del vector* x_1
Sea $u^{(0)}$ *un vector arbitrario tal que* $||u^{(0)}||_2 = 1$
$k \longleftarrow 0$
Repetir
$u_1^{(k)} = (cu_1^{(k)} - x_1^T \cdot u^{(k)})/c$
$u^{(k+1)} = Au^{(k)}$
$u^{(k+1)} = u^{(k+1)}/||u^{(k+1)}||_2$
$k \longleftarrow k + 1$
Hasta alcanzar convergencia

Obsérvese que este algoritmo es muy similar al anterior, sin embargo, a cada paso se ortogonaliza el vector $u^{(k)}$ con respecto al primer vector propio x_1.

[1] Y en general para computar los r vectores correspondientes a los r valores propios más grandes (en módulo) de A

C.2. Método de Gauss-Seidel

El método de Gauss-Seidel es una técnica para resolver las n ecuaciones del sistema $Ax = b$ una a la vez en secuencia. El algoritmo es el siguiente:

Algorithm 6 *Gauss-Seidel*
Sea $x^{(0)}$ *un vector arbitrario*
$k \longleftarrow 1$
Repetir
Para $i \longleftarrow 1$ *hasta* n *hacer*

$$x_i^{(k)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right) / a_{ii}$$

Fin para
$k \longleftarrow k + 1$
Hasta alcanzar convergencia