Instituto de Matemática e Estatística - USP

MAC0499 - Trabalho de Formatura Supervisionado

Classificação de Conteúdo Web

Estudo comparativo e implementações

Caio de Moraes Braz

Gustavo Perez Katague

Supervisor: José Coelho de Pina

11 de fevereiro de 2013

Resumo

Classificar informações por relevância é um grande desafio, principalmente por se tratar de algo subjetivo e que depende do interesse do usuário, do nível de conhecimento relativo ao assunto, entre outros.

Um dos paradigmas mais conhecidos atualmente é usar a estrutura de links das páginas web para conseguir informações relevantes e, no caso de uma busca, ordenar os resultados de forma que as melhores páginas apareçam primeiro para o usuário.

Neste contexto, dois destes métodos se destacam: o *PageRank* e o HITS, os quais, neste trabalho, estudamos, implementamos e comparamos.

Sumário

1 Introdução

1.1 Motivação

A necessidade de organizar informação existe desde os tempos mais primórdios. Estudos mostram que, na antiga biblioteca de Pergamum, os "livros" foram inventados após a falta de papiro vindo do Egito. Estes "livros" eram mais fáceis de se manusear do que os pergaminhos de papiro e logo os substituíram.

Atualmente, a internet revolucionou o mundo no que diz respeito à criação e recuperação de informação. A quantidade de dados aumenta rapidamente, de forma não organizada e muitas vezes com conteúdo equivocado, portanto seria conveniente que houvesse alguma forma de classificar a confiabilidade das informações, assim como sua relevância. Neste contexto, essa classificação se torna altamente prioritária, tornando-a um excelente alvo de estudos de como resolver este problema.

É possível perceber que, junto com a popularidade da internet, vieram mecanismos de busca via web (Web Search Engines). Estes mecanismos se utilizam de diversos métodos para classificar a importância relativa entre essas páginas e com isso conseguir distinguir os conteúdos mais relevantes em uma busca, por exemplo.

Um dos métodos existentes para classificar páginas na Internet é o **PageRank**, que classifica as páginas, atribuindo um valor (que chamaremos de *rank*) a cada uma delas, calculado a partir de sua estrutura de links imersas na Internet.

Outro método que também classifica páginas web é o **HITS**, que tem muitas semelhanças com o PageRank, no entanto, para cada página ele atribui dois valores distintos.

1.2 Objetivos

Neste trabalho de formatura supervisionado, pretendemos estudar e implementar métodos de classificação de conteúdo por popularidade, analisando aspectos como características, performance computacional e sensibilidade a parâmetros.

Neste estudo, veremos dois algoritmos desta família: o *PageRank* e o HITS.

2 Grafo da Web

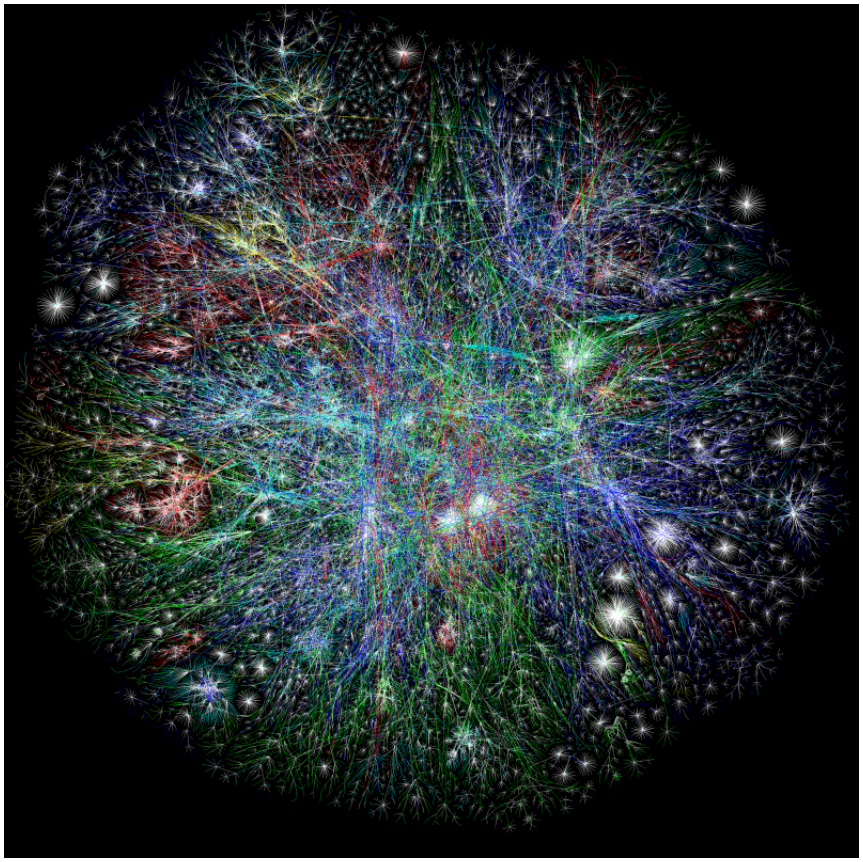

Figura 2.1: Representação dos *links* da web.
Fonte: www.opte.org/maps

A figura anterior, apesar de não representar o verdadeiro tamanho da internet, nos dá uma ideia de como ela se "organiza" e, portanto, seria interessante se pudéssemos modelar essa estrutura amorfa em algo que faça sentido do ponto de vista algorítmico.

Felizmente, estrutura de *hyperlinks* da internet forma um grafo dirigido. Os nós do grafo representam as páginas e as arestas dirigidas representam os *links*. Podemos representar um grafo por meio de uma matriz de adjacência, o que seria conveniente para este trabalho, pois mais adiante serão apresentados alguns cálculos que se utilizam de tal modelagem. Seja P_i uma página da web indexada com o inteiro i, então tomemos a matriz L como representação de um conjunto de páginas e seus *links*:

$$L_{ij} = \begin{cases} 1, & \text{se existe um } \textit{link} \text{ de } P_i \text{ a } P_j \\ 0, & \text{caso contrário} \end{cases}$$

Mais adiante veremos que esta abstração da internet nos permite aplicar os métodos de classificação, como o *PageRank* e o HITS.

Para ilustrar, daremos um exemplo ao longo do texto que servirá para melhor compreensão do *PageRank*. Levemos em consideração o seguinte grafo como sendo um pequeno grupo de páginas com seus *links*:

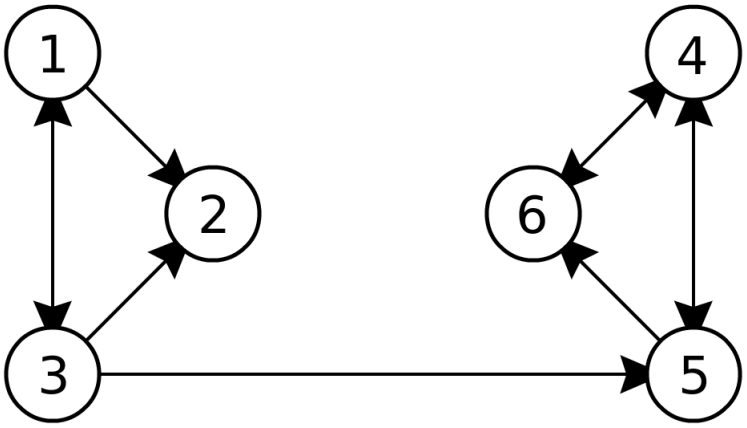

Figura 2.2: Grafo direcionado representando uma web de seis páginas.

Cuja matriz de adjacência é:

$$L = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

3 PageRank

O *PageRank* foi o alvo principal de estudos neste trabalho. Por alavancar uma das maiores empresas do segmento de computação do mundo, a Google, é pertinente atribuir um pouco mais de atenção ao método.

3.1 História

Antes de 1998, não haviam mecanismos de busca que levavam em conta a estrutura de *hyperlinks* da rede, no entanto, alguns pesquisadores, como Larry Page e Sergey Brin já tinham ideias de como retirar informações desta estrutura da web.

O PageRank foi desenvolvido na Universidade de Stanford como parte de um protótipo de um mecanismo de busca textual em páginas da web chamado *Google* [4], que usa de maneira crucial a estrutura de links das páginas.

Este protótipo foi a base para o *Google* que conhecemos hoje. O endereço original: `google.stanford.edu` ainda existe, e hoje é o mecanismo de busca oficial da Universidade de Stanford.

3.2 Ideia

A ideia do *PageRank* consiste em considerar cada hyperlink como sendo uma recomendação, isto é, um link de uma página para outra significa que a primeira serve como um aval para a segunda, portanto, uma página com mais recomendações, provavelmente, é mais importante que uma outra com menos recomendações.

No entanto, devemos também levar em conta a fonte da recomendação, pois dependendo da onde ela veio, pode ser mais relevante. Por exemplo, uma recomendação de *Donald Knuth* para uma página sobre algoritmos parece ser mais importante do que uma recomendação dele para uma página sobre esportes. Por outro lado, se descobrirmos que o *Donald Knuth* é uma pessoa muito gentil e recomenda muitas páginas sobre algoritmos, então o valor desta recomendação deve ser menor que o esperado.

Resumidamente, é assim que o *PageRank* funciona: "uma página é importante se ela é recomendada por outras páginas importantes".

3.3 Abstração

Para entender como o *PageRank* funciona, descreveremos as ações de um usuário "especial": o *Random Surfer*.

O *random surfer* faz o papel de um usuário com ações aleatórias, que clica, com probabilidade uniforme, em algum link da página que ele se encontra, assim dirigindo-se para outra página e repetindo o processo. A estrutura pura de *links* está descrita posteriormente pela matriz H, onde H_{ij} representa a probabilidade de ele usar um *link*

para ir da sua página atual i para a página j. É interessante notar, que se uma página é visitada mais vezes pelo *random surfer* do que outras, significa que existe na estrutura da web uma quantidade grande de caminhos chegando nela. Ou seja, quanto mais vezes o *random surfer* passar em uma página, mais importante ela deve ser.

Eventualmente, dependendo da estrutura dos *links*, o *random surfer* pode cair em uma página que não possua nenhuma outra como referência, chamada de *dangling node*, o que interromperia o processo de navegar aleatoriamente na internet. Quando isso ocorrer, seria ideal que o *random surfer* escolhesse qualquer página da internet para continuar o processo. Isso pode ser observado com a transição da matriz H para S, onde as páginas sem *links*, indiretamente, apontam para todas as outras.

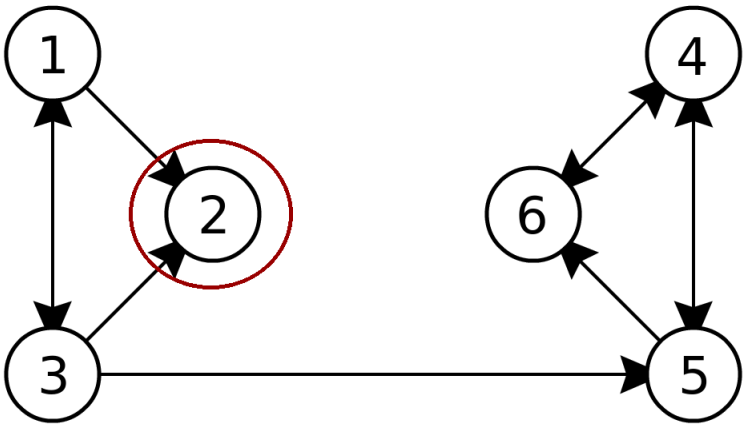

Figura 3.1: *Dangling node*, uma página sem *links*.

Para deixar o cenário um pouco mais realista (e também solucionar o problema dos *rank sinks* com ciclos), o *random surfer* pode eventualmente desistir de escolher um *link* da sua página atual e começar o processo novamente por outra página qualquer. A probabilidade α representa a probabilidade do *random surfer* continuar seguindo a estrutura original da internet e $1 - \alpha$ a probabilidade dele escolher qualquer outra página para começar novamente o processo. Essa nova matriz de probabilidade é descrita em G.

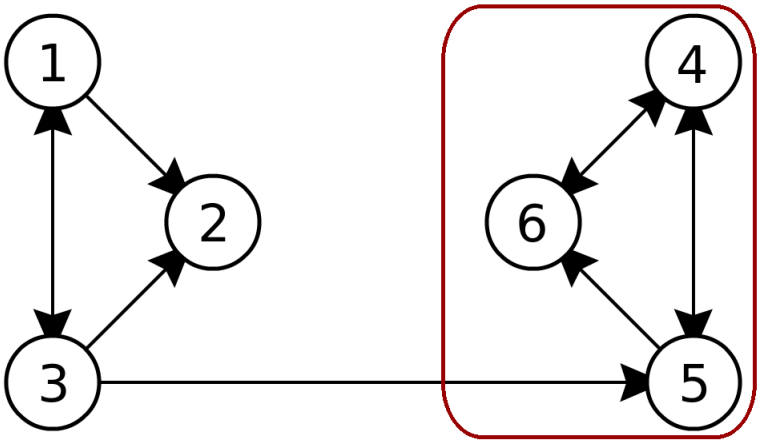

Figura 3.2: Conjunto de páginas que formam um *ciclo*.

3.4 Definição

A primeira fórmula, apresentada em [2], define o *PageRank* de uma página P_i, denotado como $r(P_i)$, como:

$$r(P_i) = \sum_{P_j \in B_{P_i}} \frac{r(P_j)}{|P_j|}$$

Onde B_{P_i} é o conjunto das páginas que apontam para P_i e $|P_j|$ é o número de links contidos em P_j.

De uma maneira intuitiva, ela tenta descrever que quanto mais referências uma página possuir, maior o seu *rank*, e mais ainda, se for referenciada por uma página muito importante, o seu *rank* será maior ainda.

Podemos ver que a definição é recursiva, se estendendo por todo o conjunto de páginas analisadas, e, eventualmente, para calcular o *rank* de uma página, recaímos na necessidade de saber o seu rank.

A representação matricial nos permite mudar um pouco a abstração e entrar no universo dos processos estocásticos, mais precisamente abordando as cadeias de Markov. Podemos imaginar que cada página representa um estado da cadeia, e cada *link* representa uma aresta de transição entre estados, então nos faltaria somente atribuir as probabilidades de transição entre eles. Caso conseguíssemos, teríamos uma matriz A de probabilidade tal que:

$$\pi' = \pi' A$$

onde π_i é o valor do *rank* atribuído à página P_i.

A fórmula pode ser interpretada como a distribuição estacionária de uma cadeia de Markov. O vetor π indica a probabilidade de se estar no estado i (ou no caso, na página P_i após infinitas mudanças de estado. É claro que para isso teríamos que possuir em mãos uma matriz A moldada e com algumas propriedades para a obtenção do vetor estacionário.

Além do ponto de vista estocástico, podemos também imaginar a matriz A como uma transformação linear. O vetor π, neste caso, representa o autovetor da transformação linear associado ao autovalor 1.

3.5 Primeira Representação Matricial: H

Podemos tentar fazer uma adaptação da matriz L para que se pareça um pouco mais com uma matriz de transição de um processo estocástico. Tentaremos então atribuir um pouco de probabilidade nos valores não nulos de L, assim obtendo a seguinte matriz H.

$$H_{ij} = \begin{cases} 1/|P_i|, & \text{se existe um arco ligando } i \text{ a } j \\ 0, & \text{caso contrário} \end{cases}$$

onde $|P_i|$ é o número de *links* contidos em P_i.

Podemos perceber também que H é muito parecida com uma *matriz estocástica*, excetuando as linhas cujas entradas são todas nulas. Estas linhas são geradas por páginas que não possuem nenhuma outra como referência, e, no contexto do grafo da web, são chamadas de *dangling nodes*. Com alguns ajustes, podemos utilizá-la como a matriz de transição A, descrita na subseção anterior, então temos:

$$\pi' = \pi' H$$

A partir da fórmula acima, podemos observar que a complexidade de cada iteração do algoritmo é $O(n^2)$, dado a multiplicação matriz-vetor. Entretanto, a matriz H é esparsa, e apenas suas entradas não-nulas são armazenadas. Segundo [5], a média de links é de aproximadamente 10 por página. Mais especificamente, H possui algo próximo de $10n$ entradas não-nulas, ao contrário de n^2 de uma matriz densa. Dessa maneira, a complexidade da iteração é da ordem de $O(n)$.

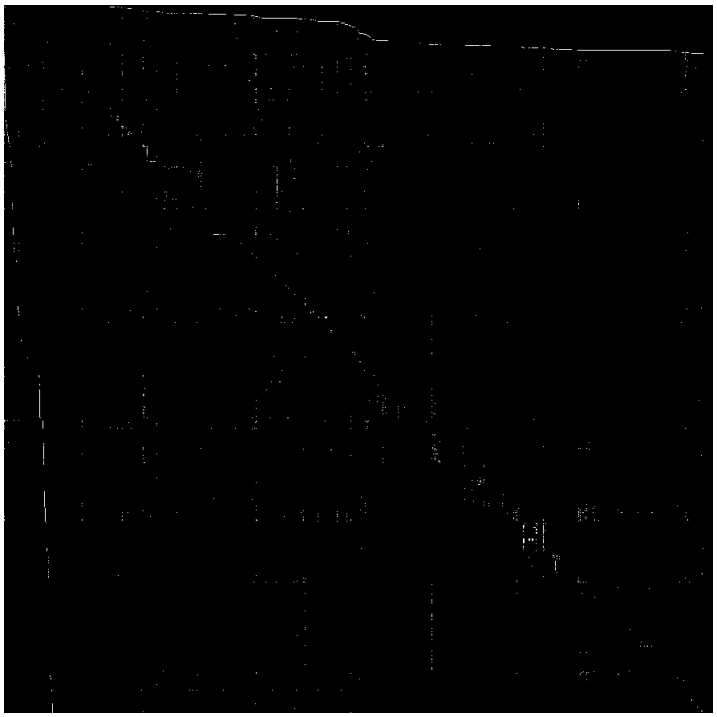

Figura 3.3: Matriz esparsa. Os pixels brancos representam entradas não nulas.

Devemos contornar um problema proveniente da modelagem escolhida: os *rank sinks*, que são páginas que acumulam *rank*, não redistribuindo para outras páginas. Existem dois tipos de *rank sink*: os *dangling nodes*, que são um *rank sink* contendo uma única página, e os ciclos, que repassam *rank* entre si, formando um subgrupo de páginas que não repassam *rank* para as outras.

Tendo como exemplo o grafo da figura 2.2, temos a seguinte matriz H que representa sua estrutura modificada pelas probabilidades de escolha dos *links* em cada página:

$$H = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

3.6 Segunda Representação Matricial: S

Para corrigir o problema dos *dangling links*, é feito um ajuste na matriz H. As linhas nulas em H serão substituídas pelo vetor $(1/n)e'$, gerando uma nova matriz que chamaremos de S, que agora é estocástica. Podemos expressar S em função de H, visto que a mudança que ocorre de uma matriz para outra é a adaptação das linhas nulas. Temos então:

$$S = H + a((1/n)e')$$

onde e é um vetor com valor 1 em todas as suas entradas.

O vetor binário a é chamado *dangling node vector*. O ajuste garante que S é estocástica, e então, uma matriz de transição de probabilidade para uma cadeia de Markov.

$$a_i = \begin{cases} 1, & \text{se a } i\text{-ésima linha de } H \text{ for nula} \\ 0, & \text{caso contrário} \end{cases}$$

Tomando como exemplo o grafo da figura 2.2, temos a matriz S, derivada da matriz H, sem o *dangling node* correspondente ao vértice 2:

$$S = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/3 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

E a representação do mesmo grafo com as arestas que surgiram após o ajuste da matriz S:

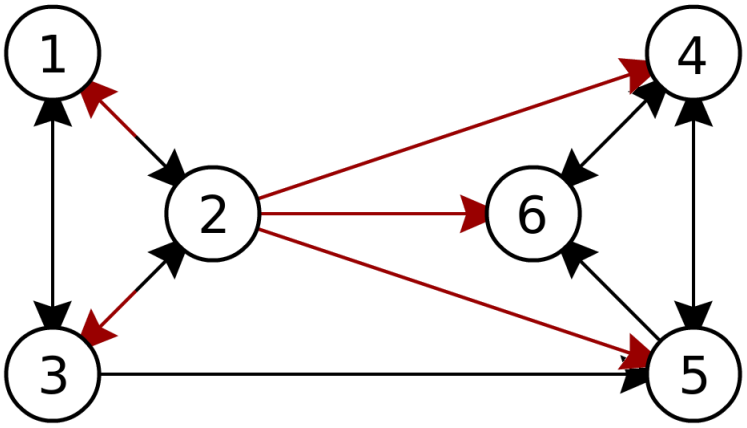

Figura 3.4: Grafo da figura 2.2 após as mudanças da matriz H para S.

3.7 Terceira Representação Matricial: G

Apesar das mudanças estocásticas que levaram a matriz H à matriz S, os resultados de convergência para o vetor π ainda não são garantidos. Seria necessário ter em mãos uma matriz que fosse primitiva além de estocástica. Uma matriz primitiva é irredutível e aperiódica, e assim o vetor de probabilidade estacionário existiria e seria único (pelo teorema de Perron-Frobenius). Esta segunda modificação resolveria o problema dos *rank sinks* que possuem ciclos.

A ideia que Brin e Page tiveram para solucionar este problema foi selecionar um escalar α entre 0 e 1, que representa a probabilidade de seguir a estrutura de *links* representada em S. A matriz resultante, G, pode ser expressa da seguinte maneira:

$$G = \alpha S + (1 - \alpha)(1/n)ee'$$

G é chamada de *Google matrix*. É interessante observá-la nos seguintes aspectos:

- G é *estocástica*, sendo combinação convexa de duas matrizes estocásticas S e $E = (1/n)ee'$
- G é *irredutível*, pois cada página está diretamente conectada com qualquer outra.
- G é *aperiódica*, pois $G_{ii} > 0$ para todo i, quebrando a periodicidade.

A matriz G é completamente densa, o que é indesejável do ponto de vista computacional. No entanto ela pode ser escrita em função de S que, por sua vez, pode ser escrita em função de H, que é esparsa, nos garantindo que podemos armazenar somente a matriz H

Finalmente, podemos escrever então

$$\pi' = \pi' G$$

Utilizando como exemplo o grafo da figura 2.2, e tomando $\alpha = 0.9$, temos a seguinte matriz G:

$$G = 0.9H + (0.9 \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + 0.1 \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix})1/6 \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$G = \begin{pmatrix} 1/60 & 7/15 & 1/60 & 1/60 & 1/60 & 1/60 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 19/60 & 19/60 & 1/60 & 1/60 & 19/60 & 1/60 \\ 1/60 & 1/60 & 1/60 & 1/60 & 7/15 & 7/15 \\ 1/60 & 1/60 & 1/60 & 7/15 & 1/60 & 7/15 \\ 1/60 & 1/60 & 1/60 & 11/12 & 1/60 & 1/60 \end{pmatrix}$$

Podemos ver que o grafo representado por G é completo se $\alpha \neq 1$.

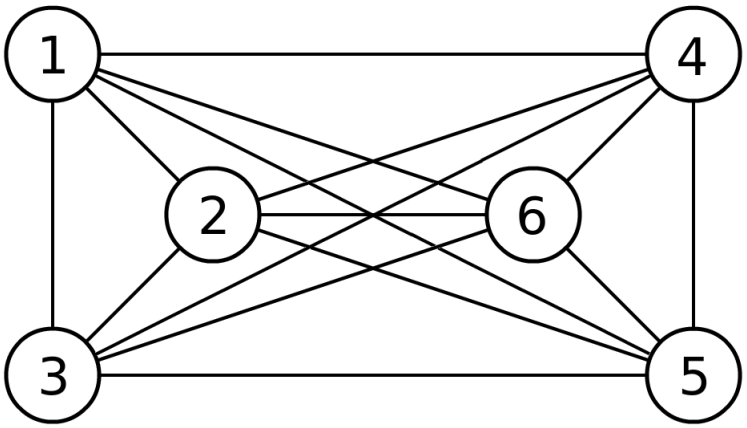

Figura 3.5: Grafo da figura 3.4 após as mudanças da matriz S para G.

3.8 Implementação

Computacionalmente, não há um algoritmo que determina com exatidão os números do vetor π, portanto, utilizaremos um algoritmo iterativo, que aproxima rapidamente o valor da distribuição estacionária. Este algoritmo é conhecido como o método da potência, que calcula uma aproximação do autovetor associado ao maior autovalor de G. Neste caso, nosso autovetor é π e o autovalor é 1.

A escolha do método da potência é dada pela necessidade de armazenar apenas a matriz H e dois vetores π para a iteração, que são o atual e o anterior. A matriz G é acessada apenas pelos cálculos realizados.

Em particular, em nossa implementação, a matriz armazenada foi a L, em uma lista de adjacência e mais um vetor que guarda o grau de saída de cada vértice. No lugar de

armazenar uma lista de adjacência de `double`, armazenamos uma lista de `int` mais um vetor de `int`.

Tendo isso, pudemos implementar nosso *PageRank* da seguinte forma:

$res \leftarrow 1$
$pi \leftarrow pi_0$
while $res \geqslant \epsilon$ **do**
  $prev\_pi \leftarrow pi$
  $pi' \leftarrow pi' * G$
  $res \leftarrow ||pi - prev\_pi||_1$
end while

É interessante notar que a linha onde é feita o produto vetor-matriz é calculada da seguinte forma:

$$\begin{aligned} G &= \alpha S + (1-\alpha)(1/n)ee' \\ &= \alpha(H + (1/n)ae') + (1-\alpha)(1/n)ee' \\ &= \alpha H + (\alpha a + (1-\alpha)e)(1/n)e' \end{aligned}$$

Como comentado acima, sendo necessárias apenas as informações contidas em H.

4 HITS

O HITS foi objeto de estudo secundário neste trabalho. Como queríamos fazer um estudo comparativo entre os mecanismos de busca baseados em popularidade e referência entre páginas, estudamos esse método por também ser importante nesse contexto.

4.1 História

O método HITS, acrônimo para *Hypertext Induced Topic Search*, foi concebido por Jon Kleinberg em 1998, durante o mesmo tempo em que Page e Brin estavam trabalhando no *PageRank*. O HITS é usado em um buscador web chamado Teoma (www.teoma.com).

4.2 Método

Podemos perceber uma mudança para o *PageRank* logo de início, ao saber que o HITS é dependente dos termos da busca realizada, enquanto o *PageRank* não. O *PageRank* atribui apenas um valor para cada página, enquanto o HITS atribui dois. Estes valores são associados às qualidades *authority* e *hub*. Uma página *authority* é uma página muito referenciada, e uma página *hub* é uma página com muitas referências. A ideia central do método diz que: "boas *authorities* são referenciadas por bons *hubs*, e bons *hubs* referenciam boas *authorities*".

Com isso, podemos dizer que cada página i tem então um valor *authority* x_i e um valor *hub* y_i. Sendo E o conjunto de todos os *links* do grafo da web e que e_{ij} represente a referência contida na página i para a página j, a partir de um valor inicial $x_i^{(0)}$ e $y_i^{(0)}$, o método HITS itera estes valores calculando:

$$x_i^{(k)} = \sum_{j:e_{ji}\in E} y_j^{(k-1)} \quad \text{e} \quad y_i^{(k)} = \sum_{j:e_{ij}\in E} x_j^{(k)} \quad \text{para } k = 1, 2, 3, ...$$

Estas equações podem ser escritas na forma de multiplicação matriz-vetor, utilizando a matriz L da seção 2, que modela o grafo da web.

$$x^{(k)} = L' y^{(k-1)} \quad \text{e} \quad y^{(k)} = L x^{(k)}$$

4.3 Algoritmo inicial

As fórmulas anteriores nos levam a um algoritmo iterativo para o cálculo dos vetores *authority* e *hub*.

$y^{(0)} \leftarrow e$
while nao convergiu **do**
  $x^{(k)} \leftarrow L' y^{(k-1)}$
  $y^{(k)} \leftarrow L x^{(k)}$
  $k \leftarrow k + 1$
end while
Normalize $x^{(k)}$ e $y^{(k)}$

Podemos perceber que as duas atribuições dentro do laço do algoritmo podem ser substituídas por:

$$x^{(k)} = L'Lx^{(k-1)}$$
$$y^{(k)} = LL'y^{(k-1)}$$

Estas duas novas equações definem o método da potência iterativo para calcular o autovetor associado ao autovalor dominante das matrizes $L'L$ e LL'. Elas são similares à convergência utilizada no *PageRank*, utilizando as duas matrizes citadas acima no lugar da *Google matrix* G. Como a matriz $L'L$ define os valores *authority*, ela é chamada *authority matrix*, a matriz LL' define os valores *hub*, sendo chamada *hub matrix*. Ambas as matrizes são *simétricas* e *semi-definidas positivas*.

4.4 Implementação

Originalmente, a implementação do HITS consistiria em dois passos:

- Gerar grafo de vizinhança.

  A ideia básica consiste em pegar as páginas que contém os termos de busca e montar um grafo N no qual aparecem estas páginas e seus vizinhos. Claramente este grafo é menor que o grafo da web.

- Calcular os valores "authority" e "hub".

  Consistiria em utilizar um método da potência para calcular o vetor x como sendo: $x'_{(k+1)} = x'_k N'N$ e após a convergência, ter $y' = x'N'$. (Aqui, N é a matriz de adjacência do grafo de vizinhança do item anterior).

No entanto, o algoritmo implementado foi uma adaptação do HITS original, para que ele fosse independente dos termos de busca e também para garantir que os resultados não dependam do valor inicial utilizado no método da potência.

As modificações no algoritmo foram:

- Usar o grafo da web no lugar do grafo de vizinhança N.

- Utilizar novas matrizes no lugar das matrizes $L'L$ e LL', a saber:
  $\xi L'L + (1-\xi)/n\, ee'$ e $\xi LL' + (1-\xi)/n\, ee'$ respectivamente, com $0 < \xi < 1$.
  Estas novas matrizes são irredutíveis, garantindo assim a unicidade dos vetores x e y pelo teorema de Perron-Frobenius.

Novamente usaremos o método da potência para calcular os vetores x e y, cujo pseudo-código é:

$x^{(0)} \leftarrow (1/n)e$
$y^{(0)} \leftarrow (1/n)e$
while $res_1 \geqslant \epsilon$ **or** $res_2 \geqslant \epsilon$ **do**
  $x^{(k)} \leftarrow x^{(k-1)}L'$
  $x^{(k)} \leftarrow x^{(k)}L$
  $y^{(k)} \leftarrow y^{(k-1)}L$
  $y^{(k)} \leftarrow y^{(k)}L'$
  $x^{(k)} \leftarrow x^{(k)}/||x^{(k)}||_1$
  $y^{(k)} \leftarrow y^{(k)}/||y^{(k)}||_1$
  $res_1 \leftarrow ||x^{(k)} - x^{(k-1)}||_1$
  $res_2 \leftarrow ||y^{(k)} - y^{(k-1)}||_1$
end while

Note que com esta modificação, não é mais possível iterar apenas para calcular x, conseguindo um y no final. Foi necessário colocar o cálculo de y para dentro da parte iterativa do algoritmo.

Esta é uma desvantagem quanto ao método original, pois cada iteração do método da potência agora é duas vezes mais custosa. No entanto, isso é compensado pelo número menor de iterações que o HITS leva para convergir.

5 Coleta de Dados

A aplicação dos métodos de classificação acima depende dos dados iniciais que representam o conjunto de páginas a ser analisado. Para obtê-los, é necessário percorrer as páginas e descobrir quais são as URLs que representam links válidos (que apontam para outras páginas, e não para algum tipo de arquivo, por exemplo). Para isso, é utilizado uma espécie de "robô", conhecido como *crawler*, que percorre as páginas automaticamente e obtém a estrutura de links desejada.

O *crawler* possui a função de acessar as páginas e baixá-las, mas o importante nesse contexto é identificar os links que uma página contém. Para isso, foi utilizado um *parser*, que tem como objetivo filtrar todo o conteúdo de um código fonte em busca de possíveis links.

5.1 Crawler

A implementação do crawler que foi feita tem um aspecto diferente dos demais: ele limita os links válidos de acordo com o domínio oferecido. A ideia principal é verificar a estrutura de páginas sob um determinado domínio, por exemplo, o domínio `www.ime.usp.br`. Inicialmente, a página principal do instituto será analisada pelo parser e então avaliaremos todos as links contidos nela. Com essa mudança, conseguimos filtrar algumas páginas cuja análise não é interessante, como `www.usp.br`, pois esta se encontra fora do domínio `www.ime.usp.br`. Por outro lado a página do Departamento de Ciência da Computação (`www.ime.usp.br/dcc`) será guardada para posterior análise de seus links.

O crawler recebe dois argumentos, sendo um arquivo com um domínio e um inteiro que limita o número de páginas que serão analisadas.

O algoritmo é baseado na seguinte ideia:

- Lista de URLs
  Pega-se a próxima URL da lista para análise. Esta lista de URLs contém apenas as páginas que foram encontradas durante o processo de busca e ainda não foram analisadas.

- Download
  Uma vez com a nova URL em mãos, o crawler, por meio da biblioteca cURL, acessa esta URL e, se possível, faz o download do código fonte dela.

- Análise
  O código fonte é passado para o parser, para que este analise a página, procurando os links que estão hierarquicamente sob o domínio dado. Uma vez terminada a análise, o parser devolve uma lista com todas as URLs encontradas.

- Atualização
  O crawler recebe a lista do parser e atualiza a lista de URLs, com o cuidado de não

colocar neste lista um endereço que está ou já esteve nela. Isso é feio por meio de uma função de espalhamento que controla as URLs já encontradas.

- Término
  O processo termina quando o crawler chega ao número máximo de páginas passado como argumento ou se a fila terminar antes deste número, caso contrário, segue-se pegando a próxima URL da fila.

Utilizando-se de uma perspectiva diferente, podemos entender o crawler como uma espécie de busca em largura pelos vértices de um grafo, no caso o grafo da web. A página em análise corresponde ao vértice em que o algoritmo se encontra e as páginas com URL válidas correspondem aos vértices vizinhos. A lista de URLs que o crawler acessa nada mais é do que a fila que existe em um algoritmo de busca em largura tradicional.

A tecnologia que utilizamos para integrar a linguagem C à internet foi uma biblioteca chamada cURL (curl.haxx.se/libcurl), que nos permitiu acessar e obter o código fonte de uma página na web. Para que a execução do crawler seja correta, é necessário que haja conexão com a internet.

5.2 Parser

```
<html>
  <head>
    <title>Katague's page</title>
  </head>
  <body>
    "By knowing things that exist, you can know that which does not exist."
    - Miyamoto Musashi

    <a href="1.html">1</a>
  </body>
</html>
```

O trecho acima ilustra um exemplo simplificado de código fonte de uma página html. Para conseguirmos extrair os links de uma página, foi implementado um parser em linguagem C. Um parser é um programa que realiza uma análise sintática em uma sequência de entrada, onde no nosso caso temos o código fonte de uma página (html, htm, php). Procuramos ocorrências da tag `<a href="...">`.

"The <a> tag defines a hyperlink, which is used to link from one page to another." - www.w3schools.com

O parser manipula o arquivo que contém o código fonte da página que está sendo analisada. Além de extrair os links das páginas, ele também faz a análise da URL, para conferir se esta pertence ao domínio ou não, ou também se não é um tipo de arquivo. Além disso, o parser também é responsável pela extração do título da página, identificado pelas tags `<title>` e `</title>`.

Inicialmente, é necessário definir no parser sob qual domínio as páginas serão analisadas. Antes de encaminhar qualquer página para análise, o crawler deve chamar a função `parser_init(char*)` que recebe a string correspondente ao domínio. Em vários momentos o parser manipula as URLs para um formato que seja mais conveniente trabalhar. Se o domínio, por exemplo, for encaminhado pelo crawler como `"http://www.ime.usp.br/"`, o parser ignora o trecho `"http://"` e a `"/"` final, transformando-o em `"www.ime.usp.br"`.

Após as configurações iniciais, o crawler pode fazer a chamada da função principal do parser, `parse(char*, char*, char*)`, onde o primeiro argumento é o nome do arquivo contendo o código fonte a ser analisado, o segundo é o nome do arquivo onde o parser escreverá os links válidos, e o terceiro é a URL global correspondente ao código fonte. Nesse momento, o parser começa a procurar os links da página, e cada um deles passa por um filtro, que determina se a URL do link aponta para outra página dentro do domínio ou não.

O primeiro filtro consiste em procurar na URL extensões de arquivos não relacionados a formatos web, como `jpg`, `mp3`, entre outros. O parser está programado para filtrar cerca de 80 tipos de formatos de arquivos diferentes. Após essa verificação, ele procura por padrões como `mailto:` ou `file:`. Se o parser encontrar algum desses padrões, a URL é descartada.

Entretanto, a URL pode conter caracteres em hexadecimal, cujo número corresponde ao caracter em ASCII. Se a URL não foi descartada até aqui, ocorre a conversão desses caracteres para o padrão usado pela linguagem C, por exemplo, se no meio da URL houver uma ocorrência `"/%7Ecoelho"`, o parser fará o ajuste para `"/~coelho"`.

Alguns caracteres especiais podem aparecer na URL [1]. Por exemplo, `'#'` representa uma âncora dentro de uma página, isto significa que `pt.wikipedia.org/wiki/USP` e `pt.wikipedia.org/wiki/USP#Estrutura` apontam para a mesma página, e `'?'` representa os parâmetros de uma busca em determinada URL. O parser identifica estes caracteres especiais e os elimina, junto com o seus parâmetros, deixando a URL mais "limpa".

O fato de duas URLs distintas poderem apontar para uma mesma página foi uma das maiores dificuldades a ser solucionada. Houve uma tentativa de reduzir os efeitos desse problema, mas não é correto afirmar que o parser implementado é completamente imune a ele. Uma das situações é a ocorrência de `"index.html"` na URL. Caso ocorra, o parser o retira da URL.

Passado por todos esses filtros, o parser procura pelo padrão do domínio na URL. Por isso, ao ajustar o domínio no começo do processo (em `set_domain()`) fizemos com que o `http://` e a `'/'` final não fossem incluídas no domínio. Caso o domínio ocorra na URL, temos uma URL correspondente a um link válido. Caso contrário, procuramos a ocorrência de `http://` e alguns dos domínios mais usados na internet, como `.com`, `.net`, `.org`, e caso alguma dessas ocorrências for positiva, sabemos que a URL é global, e se o domínio não ocorre nela, então não é válida.

Se a URL chegou até esse ponto, pode-se afirmar que ela é local. A presença destas gera muitos casos de tratamento, e cabe ao parser fazer a conversão para as URLs globais,

e é por isso que existem variáveis, `domain`, `actual` e `http_domain` (esta última guarda a raiz http de onde se encontra o domínio) que guardam algumas URLs especiais. Para exemplificar, se o domínio escolhido é `www.ime.usp.br/~cef`, e a página atual que está sendo analisada pelo parser é `www.ime.usp.br/~cef/mac499-10/`, temos:

- `domain = "www.ime.usp.br/~cef"`
- `http_domain = "www.ime.usp.br"`
- `actual_url = "www.ime.usp.br/~cef/mac499-10/"`

Existe um caso especial de tratamento, que são as URLs começadas com `/`, como `"/~cef"`. Uma URL que começa com `/` faz referência ao diretório raiz http. No caso, o parser concatena `http_domain` com `"/~cef"`, gerando a URL global `www.ime.usp.br/~cef`

As demais URLs podem ter ocorrências de `./` ou `../`, como `"./monografias/"` ou `"../orient.html"`, ou mesmo simplesmente `"plagio.html"`. Nestes casos, o parser monta a URL global concatenando `actual_url` com a URL encontrada. Ao concatenar, teremos como URL `"www.ime.usp.br/~cef/mac499-10/../orient.html"` por exemplo. O parser trata a URL e a transforma em `"www.ime.usp.br/~cef/orient.html"`. O parser também trata um caso especial, onde no diretório raiz `../` recai no próprio diretório raiz.

Essa padronização das URLs é necessária, porque é essencial que o crawler consiga distinguir quais páginas já foram analisadas e quais não foram.

5.3 Hash

Durante sua execução, o crawler eventualmente vai encontrar uma página que já foi acessada, logo seria conveniente se houvesse uma maneira de distinguir quais páginas já foram visitadas, algo análogo a pintar os vértices já visitados em uma busca. Para isso, utilizamos uma função de espalhamento (*hash*), adaptada de um código escrito por Donald Knuth, que nos foi fornecido pelo Prof. Coelho. Nesta função de espalhamento a chave escolhida para distinguir as páginas é a URL.

Entretanto, duas cadeias de caracteres distintas podem apontar para a mesma página. O parser trata as URLs locais, mas não filtra o `http://` e a `/` final ao devolvê-las para o crawler. Ao aplicar a função de espalhamento numa URL, o programa verifica se estes padrões aparecem na URL e os filtram, para tentar garantir a unicidade de cada página.

5.4 Arquivos de Saída

No final do processo, o crawler disponibiliza dois arquivos de saída:

- nodes.txt
  Neste arquivo temos a lista com as páginas analisadas, de modo que cada página gera uma entrada da forma:

```
12
http://www.ime.usp.br/~coelho/geocomp2002
MAC0331 / MAC5747 Geometria Computacional
16
```

Em que cada linha representa, respectivamente: índice, URL, título e número de links.

É importante ressaltar que o número de links se restringe apenas às páginas que se encontram no domínio. Além disso, se houver mais de um link para uma mesma página, apenas um deles será considerado.

- adj_list.txt
  Neste arquivo, temos a lista de adjacência do grafo construído pelo crawler, no formato exato para que sirva de entrada para o programa que calcula o PageRank e o HITS.

6 Resultados

Em nossos testes utilizamos dados provenientes de duas fontes diferentes. Uma dessas fontes é a **página** de P. Tsaparas, da Universidade de Toronto. Nela encontramos conjuntos de arquivos que representam sub-grafos reais da web. Embora eles sejam uma forma de grafo de vizinhança, nos moldes vistos no algoritmo HITS, eles foram utilizados como amostras do real grafo da web, por terem as mesmas características e serem menores em escala.

A segunda fonte foi o crawler implementado nesse trabalho, que gera os arquivos que são utilizados como base para os métodos. Estes arquivos estão no mesmo formato que os encontrados na página de Tsaparas, com a diferença que as páginas contidas neles fazem parte de um único domínio.

6.1 PageRank

- Número de iterações conforme α

  Resultados mostrando como o parâmetro α influencia diretamente no número de iterações do método da potência que calcula o *PageRank*. Dados na tabela a seguir, onde V e A são respectivamente o número de vértices e arcos dos *datasets* utilizados.

α	0,8	0,9	0,95	0,99	0,999	0,9999	0,99999
`(V,A) = 2.293, 9.644`	43	88	173	800	8.017	80.212	802.052
`(V,A) = 4.298, 21.956`	42	83	167	802	8.007	80.104	801.068
`(V,A) = 4.334, 17.424`	46	95	192	971	9.745	97.494	974.975
`(V,A) = 5.354, 24.389`	46	97	196	993	9.966	99.705	997.092
`(V,A) = 7.399, 36.121`	48	100	203	1.032	10.362	103.660	1.036.641
`(V,A) = 8.011, 34.672`	43	90	183	910	9.135	91.390	913.940
`(V,A) = 11.659,292.236`	41	86	174	855	8.554	85.577	855.806

  É possível observar que quando $\alpha \to 1$, o número de iterações tende a ficar muito alto. Isso se deve ao fato de α ser diretamente proporcional ao segundo maior autovalor da matriz G[5]. Ainda segundo [5], quanto mais próximo de 0 valor da razão λ_2/λ_1 (onde λ_1 é o maior autovalor e λ_2 o segundo maior), menos iterações o método da potência requer para convergir. Como temos $\lambda_1 = 1$, a convergência é determinada diretamente por λ_2, logo, por α.

- Sensibilidade de π em relação ao α

  Como a matriz G está diretamente ligada por construção ao parâmetro α, gostaríamos de testar como o resultado do processo, representado por $\pi' = \pi' G$, é influenciado pela escolha do α.

Estes resultados foram obtidos com a utilização do algoritmo implementado, aplicado ao dataset que representa a busca por "basketball".

α	0,6	0,85
1ª posição	www.studentadvantage.com	www.studentadvantage.com
2ª posição	www.nba.com	www.nba.com
3ª posição	www.knightridder.com	www.knightridder.com
4ª posição	www.fiba.com	www.ncaa.org
5ª posição	www.ncaa.org	www.fiba.com

α	0,9	0,99
1ª posição	www.studentadvantage.com	asp.personello.com/basketball
2ª posição	www.nba.com	www.studentadvantage.com
3ª posição	www.knightridder.com	www.nba.com
4ª posição	www.fiba.com	www.knightridder.com
5ª posição	www.ncaa.org	www.hoophall.com

Já estes resultados foram obtidos utilizando o dataset que representam a busca por "search engines"

α	0,6	0,85
1ª posição	www.google.com	www.dejanews.com
2ª posição	www.dejanews.com	www.terralycos.com
3ª posição	www.terralycos.com	www.google.com
4ª posição	www.yahoo.com	www.bruceclay.com
5ª posição	www.altavista.com	www.yahoo.com

α	0,9	0,99
1ª posição	www.dejanews.com	www.web-site-promotion-services.co.uk
2ª posição	www.terralycos.com	www.topwebsite.co.uk
3ª posição	www.google.com	www.ftpsearchengines.com
4ª posição	www.bruceclay.com	members.cox.net/ftpsearch/ftp1.html
5ª posição	www.yahoo.com	www.webmasterworld.com

6.2 HITS

- Sensibilidade de ξ em relação ao α

O parâmetro ξ, é usado para transformar a matriz original em uma matriz primitiva. Nossos testes mostraram que o número de iterações do HITS não é afetado pelo ξ escolhido, exceção feita a valores muito próximos de 0.

- Tamanho do grafo e número de iterações.

  Curiosamente, o número de iterações não está relacionado ao tamanho do grafo, nos levando a crer que o critério de convergência está fortemente relacionado à estrutura dos autovalores da matriz primitiva criada com o ξ.

Vértices	Iterações
742	73
1.075	37
2.188	21
3.266	10
4.326	17
5.243	136
6.049	12
7.967	191
8.011	20
11.659	6

6.3 Páginas obtidas pelo crawler

O crawler possibilitou a manipulação dos dados extraídos da internet. É importante perceber que contexo dos dados obtidos pelo crawler implementado é diferente dos apresentados pela página de Tsaparas, pois o primeiro representa as páginas de um determinado domínio, e a segunda representa o resultado de uma busca numa *web search engine*. Escolhemos alguns domínios conhecidos para realizar a extração das páginas:

- `www.ime.usp.br`

  Número de páginas: 20000+

  A tabela a seguir mostra a classificação feita pelo PageRank dos 10 resultados mais relevantes.

α	0,85
1ª posição	/
2ª posição	/pessoas
3ª posição	/pessoas/busca_pessoas.php
4ª posição	/dcc
5ª posição	/~gold/cursos/2004/mac438/discussao
6ª posição	/~gold/cursos/2004/mac438/discussao/threads.html
7ª posição	/~yoshi
8ª posição	/~webadmin/joomlantigo
9ª posição	/pessoas/pessoas.php
10ª posição	/~durham/cursos/mac316/pub/oldMail/msg00080.html

A tabela a seguir mostra a classificação feita pelo HITS dos 10 resultados mais relevantes de páginas *authority* e *hub*.

ξ	0,85 (authority)	0,85 (hub)
1ª posição	/	/dcc/docentes
2ª posição	/dcc	/posgrad
3ª posição	/pessoas	/posgraduacao.php
4ª posição	/~kon	/pos
5ª posição	/~webadmin/joomlantigo	/~webadmin/.../1sem2011 *
6ª posição	/~pf	/dcc/premios
7ª posição	/dcc/pos	/dcc/historia
8ª posição	/~kon/MAC211	/dcc/publicacoes
9ª posição	/dcc/grad	/dcc/naps
10ª posição	/dcc/areas	/dcc/docentes

* /~webadmin/joomlantigo/dcc/posgrad/horarios/disciplinas/1sem2011

Nota: Enquanto o crawler estava sendo executado, percebemos que dentro do domínio existe um diretório que contém uma cópia do site do instituto. Isso faz com que o número de páginas acessíveis a partir de `www.ime.usp.br` praticamente dobre. (www.ime.usp.br/~webadmin/joomlantigo)

- `www.ime.usp.br/~coelho`

  Número de páginas: 1433

  A tabela a seguir mostra a classificação feita pelo PageRank dos 5 resultados mais relevantes.

α	0,85
1ª posição	/grafos/programas/emacs/faq/4.html
2ª posição	/geocomp2001/projetos
3ª posição	/grafos/programas/emacs/faq/20.html
4ª posição	/
5ª posição	/mac0328-2003/lista/msg00272.html

  A tabela a seguir mostra a classificação feita pelo HITS dos 5 resultados mais relevantes de páginas *authority* e *hub*. Todos os resultados encontrados estão dentro do diretório `/grafos/programas/emacs/faq/`.

ξ	0,85 (authority)	0,85 (hub)
1ª posição	/89.html	/
2ª posição	/87.html	/21.html
3ª posição	/4.html	/85.html
4ª posição	/20.html	/16.html
5ª posição	/81.html	/15.html

- `www.ime.usp.br/~cef`

  Número de páginas: 1113

  A tabela a seguir mostra a classificação feita pelo PageRank dos 5 resultados mais relevantes.

α	0,85
1ª posição	/mac499-09/monografias/roberto/doc/files.html
2ª posição	/mac499-09/monografias/roberto/doc/annotated.html
3ª posição	/mac499-03/lista/maillist.html
4ª posição	/mac499-03/lista/threads.html
5ª posição	/mac499-09/monografias/roberto/doc

  A tabela a seguir mostra a classificação feita pelo HITS dos 5 resultados mais relevantes de páginas *authority* e *hub*. Todos os resultados encontrados estão dentro do diretório `/mac499-09/monografias/roberto/doc`.

ξ	0,85 (authority)	0,85 (hub)
1ª posição	/	/files.html
2ª posição	/annotated.html	/globals_func.html
3ª posição	/files.html	/globals.html
4ª posição	/globals.html	/globals_vars.html
5ª posição	/table_8h.html	/player_8h_source.html

6.4 Comparação

Quanto à implementação, os dois métodos são bem semelhantes. Ambos usam o método da potência para calcular os autovetores associados aos autovalores dominantes das respectivas matrizes. O tempo utilizado em cada iteração varia conforme o número de vértices no grafo.

Após a implementação do crawler, foi possível fazer alguns testes em domínio controlado. O exemplo da figura 2.2 pode ser encontrado na página `www.linux.ime.usp.br/~katague/mac499/figura-2.2`.

A tabela seguinte mostra a classificação de ambos os métodos para as páginas representadas pela figura 2.2. (Para $\alpha = \xi = 0,85$)

	PageRank	HITS Authority	HITS Hub
1ª posição	4	5	3
2ª posição	6	2	4
3ª posição	5	6	1
4ª posição	2	1	5
5ª posição	3	4	6
6ª posição	1	3	2

Um outro exemplo, ilustrado pela figura abaixo, pode ser encontrado em www.linux.ime.usp.br/~katague/mac499/teste.

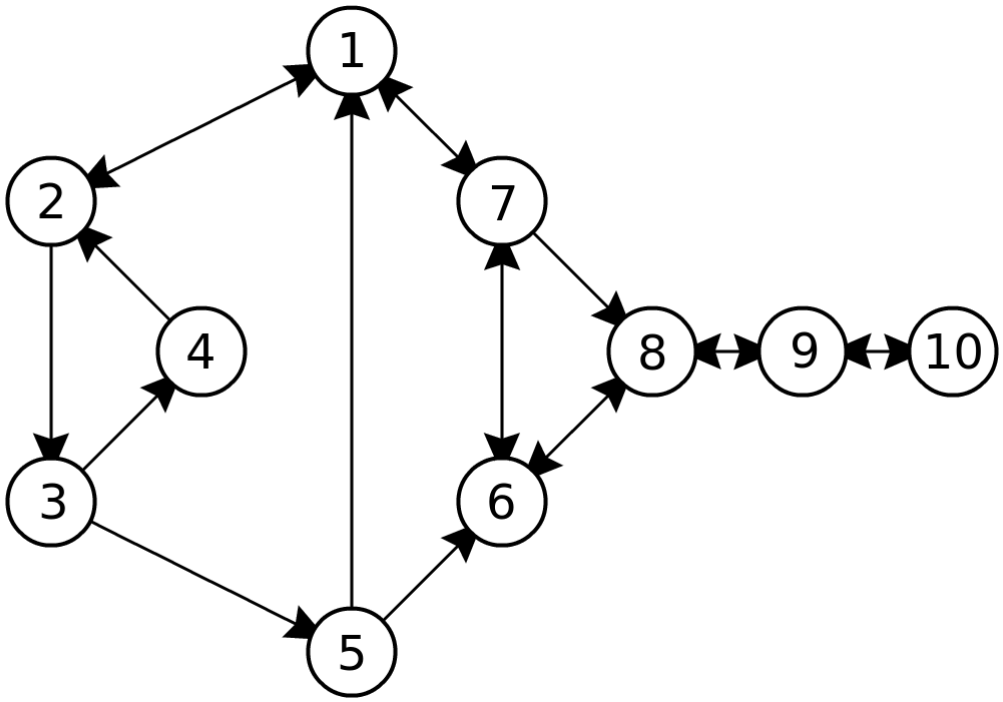

Figura 6.1: www.linux.ime.usp.br/~katague/mac499/teste

A tabela seguinte mostra a classificação de ambos os métodos para as páginas representadas pela figura 6.1. (Para $\alpha = \xi = 0,85$)

	PageRank	HITS Authority	HITS Hub
1ª posição	8	6	7
2ª posição	9	1	5
3ª posição	6	8	6
4ª posição	7	7	8
5ª posição	1	9	2
6ª posição	2	2	9
7ª posição	10	3	1
8ª posição	3	10	3
9ª posição	4	4	10
10ª posição	5	5	4

As tabelas seguintes mostram a classificação de ambos os métodos para o dataset que representa a busca por "computational complexity". (Para $\alpha = \xi = 0,85$)

	PageRank
1ª posição	eccc.uni-trier.de/eccc
2ª posição	www.fortnow.com/lance/complog
3ª posição	computationalcomplexity.org
4ª posição	www.eatcs.org
5ª posição	www.c3.lanl.gov/~percus/volume.html
6ª posição	www.liacs.nl/~beatcs
7ª posição	www.springer.de
8ª posição	link.springer-ny.com/link/service/journals/00037
9ª posição	www.gillespiefox.com
10ª posição	www.uiuc.edu

	HITS Authority ranking
1ª posição	eccc.uni-trier.de/eccc
2ª posição	www.acm.org
3ª posição	www.combinatorics.org
4ª posição	dimacs.rutgers.edu
5ª posição	link.springer-ny.com/link/service/journals/00037
6ª posição	computer.org
7ª posição	www.eatcs.org
8ª posição	eccc.uni-trier.de/eccc/info/people.html
9ª posição	cs.utep.edu/longpre/complexity.html
10ª posição	citeseer.nj.nec.com/cs

	HITS Hub ranking
1ª posição	eccc.uni-trier.de/eccc/info/people.html
2ª posição	www.tcs.hut.fi/~orponen/bookmarks.html
3ª posição	www.cs.rutgers.edu/~chvatal
4ª posição	www.dcc.unicamp.br/fkm/comptheory.html
5ª posição	www.sztaki.hu/~ivanyos/mhotlist.html
6ª posição	haegar.informatik.uni-wuerzburg.de/.../vollmer/tcs-bookmarks.html
7ª posição	math.uni-heidelberg.de/logic/bb/bblinks.html
8ª posição	math.uni-heidelberg.de/logic/e_wwwlinks.html
9ª posição	math.uni-heidelberg.de/logic/wwwlinks.html
10ª posição	www.fortnow.com/lance/complog

As tabelas anteriores mostram que existe uma divergência razoável na ordem dos resultados de classificação dos dois algoritmos. Ao procurar por complexidade computacional, a maioria dos resultados que obtivemos foram relevantes. Vale lembrar que os *links* nas tabelas referenciam alguns domínios que podem estar obsoletos pelo fato do grafo do dataset ter sido construído há alguns anos.

6.5 Conclusão

Conceber um método de classificação de conteúdo não é fácil, pois muitos requisitos devem ser levados em conta. Nestes dois métodos, por exemplo, em nenhum momento é levado em consideração o conteúdo do código-fonte das páginas que não sejam os *links*. Muito embora isso seja trabalho do método de busca e não do método de classificação, o trabalho conjunto das partes pode servir pra trazer resultados cada vez mais assertivos diante do que se procura.

Uma das dificuldades destes métodos quando utilizados em mecanismos de busca é a grandeza dos grafos com que trabalham. Estes grafos requerem uma abordagem especial nos quesitos algorítmico e armazenamento, pois não é viável guardar nem mesmo a matriz L, esparsa, quando o número de vértices é muito grande.

Fazer a distinção de qual algoritmo é melhor não é uma tarefa objetiva. Talvez para alguns termos específicos, o *PageRank* mostre um resultado melhor, e o HITS seja melhor para outros. É importante ressaltar que este é o método de classificação no seu estado puro, ao contrário dos buscadores Google e Teoma, que se utilizam de outros artifícios para refinar a ordenação do resultado de suas buscas.

7 Apêndice

Esta seção é dedicada a prover ao leitor algumas definições e explicações matemáticas com as quais ele não pode estar habituado.

7.1 Matricial

Neste trabalho, definimos um vetor como sendo uma matriz $n \times 1$, o vetor-coluna.

Definição. *Uma matriz $A_{n\times n}$ é dita* ***estocástica*** *quando todas as suas entradas são não-negativas e em cada linha a soma de suas entradas é igual à 1.*

Definição. *Uma matriz $A_{n\times n}$ é dita* ***irredutível*** *se e somente se o grafo direcionado que ela representa é fortemente conexo. Em outras palavras, se para cada par de índices (i, j) existe uma sequência de entradas em A tal que $a_{ik_1}a_{k_1k_2}...a_{k_tj} \neq 0$. Equivalentemente, A é irredutível se para qualquer matriz de permutação P,*

$$P'AP \neq \begin{pmatrix} X & Y \\ 0 & Z \end{pmatrix}, \text{ onde } X \text{ e } Z \text{ são quadradas}$$

Definição. *Uma matriz $A_{n\times n}$ é dita* ***periódica*** *se, para algum inteiro $k \geq 2$, $A^k = A$. Caso não exista tal k, a matriz é dita* ***aperiódica****.*

Definição. *Uma matriz $A_{n\times n}$ é dita* ***semi-definida positiva*** *se, para qualquer vetor $v_{n\times 1}$,*

$$v'Av \geqslant 0$$

7.2 Processos Estocásticos

Definição. *Um* ***processo estocástico*** *é um conjunto de variáveis aleatórias $\{X_t\}_{t=0}^{\infty}$ que contêm um espaço de estados $\{S_1, S_2, ..., S_n\}$ em comum. O parâmetro t é geralmente associado ao tempo e X_t representa o estado do processo no tempo t. No contexto do PageRank consideraremos o tempo discreto.*

Definição. *Uma* ***cadeia de Markov*** *é um processo estocástico que satisfaz a propriedade de Markov*

$$P(X_{t+1} = S_j | X_t = S_{i_t}, X_{t-1} = S_{i_{t-1}}, \ldots, X_0 = S_{i_0}) = P(X_{t+1} = S_j | X_t = S_{i_t})$$

para cada $t = 0, 1, 2 \ldots$. *A notação* $P(E|F)$ *denota probabilidade condicional do evento* E *ocorrer dado o acontecimento do evento* F.

A ***propriedade de Markov*** *garante que o processo não possui memória. As transições de estado ao longo do tempo dependem apenas do estado atual, sendo sua história irrelevante. O processo do usuário aleatório na web é uma cadeia de Markov.*

Definição. *A* ***probabilidade de transição*** $p_{ij}(t) = P(X_t = S_j | X_{t-1} = S_i)$ *é a probabilidade de mudança do estado* S_i *para o estado* S_j *no tempo* t.

Definição. *Numa cadeia de Markov, um estado* S_j *é* ***acessível*** *de um estado* S_i, *ou* $S_i \to S_j$, *quando num instante o estado atual é* S_i, *e após uma quantidade finita de transições é possível estar no estado* S_j. *Matematicamente,* $P(X_{n+k} = S_j | X_k = S_i) = p_{ij}^{(n)} > 0$

Definição. *Numa cadeia de Markov, um estado* S_i *se* ***comunica*** *com um estado* S_j *quando* $S_i \leftrightarrow S_j$.

Definição. *Uma cadeia de Markov é dita* ***irredutível*** *quando a matriz de transição* P *que representa a cadeia é irredutível.*

Definição. *Um* ***vetor de distribuição de probabilidade*** *(ou vetor de probabilidade), é definido como um vetor linha não negativo* $p' = (p_1, p_2, ..., p_n)$ *onde* $\sum_k p_k = 1$

Definição. *Um vetor de probabilidade* ***estacionário*** *para uma cadeia de Markov, cuja matriz de transição é* P, *é um vetor* π' *tal que* $\pi' = \pi' P$

7.3 Álgebra Linear

Definição. *Um* ***autovetor*** *de uma matriz* $A_{n \times n}$ *é um vetor não nulo que, quando multiplicado por* A, *resulta num vetor que preserva a direção do vetor original, diferindo apenas de um escalar. Matematicamente,* v *é um autovetor da matriz* A *se:*

$$Av = \lambda v$$

O escalar λ *é denominado o* ***autovalor*** *associado à* v.

Teorema. ***(Perron–Frobenius)*** *Uma matriz real* $A_{n \times n}$ *com todas as entradas positivas possui um único autovalor dominante positivo, sendo seu autovetor associado composto estritamente por entradas positivas.*

Referências

[1] T. Berners-Lee, R. Fielding e L. Masinter, *Uniform resource identifier (uri): Generic syntax*, http://tools.ietf.org/html/rfc3986, janeiro 2005.

[2] Sergey Brin, Rajeev Motwani, Lawrence Page e Terry Winograd, *The pagerank citation ranking: Bringing order to the web*, Technical report, Stanford University, 1998.

[3] _____, What can you do with a web in your pocket?, *IEEE Data Engineering Bulletin* **21** (1998), no. 2, 37–47.

[4] Sergey Brin e Lawrence Page, The anatomy of a large-scale hypertextual web search engine, *Proceedings of the Seventh International World Wide Web Conference*, vol. 30, April 1998, http://infolab.stanford.edu/~backrub/google.html, pp. 107–117.

[5] Amy N. Langville e Carl D. Meyer, *Google's pagerank and beyond: The science of search engines rankings*, Princeton University Press, 2006.

[6] Mariana Pereira de Melo, *Ordenação das páginas do Google - PageRank*, Master's thesis, Instituto de Matemática e Estatística - Universidade de São Paulo, maio 2009.

[7] Sheldon M. Ross, *Probability models for computer science*, Harcourt Academic Press, 2002.