

POLYTECHNIC UNIVERSITY
Department of Computer and Information Science

PageRank

K. Ming Leung

Abstract: The basic ideas behind Google's PageRank algorithm is discussed.

Directory

- [Table of Contents](#)
- [Begin Article](#)

Copyright © 2000 mleung@poly.edu
Last Revision Date: December 2, 2004

Table of Contents

1. Introduction
2. The PageRank Concept
3. Markov Chain Transitional Matrix for PageRank
4. Computing PageRanks
5. An Illustrative Example
6. Moler's surfer Program

1. Introduction

Within the past few years, Google[1] has become by far the most utilized search engine worldwide. A decisive factor, besides high performance and ease of use, is the superior quality of search results compared to other search engines. This quality of search results is substantially based on a sophisticated method, called PageRank[2], to rank web documents.

The PageRank algorithm was developed by Google's founders, Larry Page and Sergey Brin, when they were graduate students at Stanford University. PageRank is determined entirely by the link structure of the World Wide Web. It is recomputed about once a month and does not involve the actual content of any Web pages or individual queries. Then, for any particular query, Google finds the pages on the Web that match that query and lists those pages in the order of their PageRank.

There is no doubt that within the past years many changes, adjustments and modifications regarding the ranking methods of Google have taken place. Some details of the algorithm are well-kept com-

pany secrets. We will discuss here one particular version of it in the hope that the basic ideas are still valid today.

2. The PageRank Concept

Since the early stages of the world wide web, search engines have developed different methods to rank web pages. Until today, the occurrence of a search phrase within a document is one major factor within ranking techniques of virtually any search engine. The occurrence of a search phrase can thereby be weighted by the length of a document (ranking by keyword density) or by its accentuation within a document by HTML tags.

For the purpose of better search results and especially to make search engines resistant against automatically generated web pages based upon the analysis of content specific ranking criteria (doorway pages), the concept of link popularity was developed. Following this concept, the number of inbound links for a document measures its general importance. Hence, a web page is generally more important, if many other web pages link to it. The concept of link popularity

often avoids good rankings for pages which are only created to deceive search engines and which don't have any significance within the web, but numerous webmasters elude it by creating masses of inbound links for doorway pages from just as insignificant other web pages.

Contrary to the concept of link popularity, PageRank is not simply based upon the total number of inbound links. The basic approach of PageRank is that a document is in fact considered the more important the more other documents link to it, but those inbound links do not count equally. First of all, a document ranks high in terms of PageRank, if other high ranking documents link to it.

So, within the PageRank concept, the rank of a document is given by the rank of those documents which link to it. Their rank again is given by the rank of documents which link to them. Hence, the PageRank of a document is always determined recursively by the PageRank of other documents. In the end, PageRank is based on the linking structure of the whole web. Although this approach seems to be very broad and complex, Page and Brin were able to put it into practice by a relatively trivial algorithm.

3. Markov Chain Transitional Matrix for PageRank

Imagine surfing the Web, going from page to page by randomly choosing an outgoing link from one page to get to the next. This can lead to dead ends at pages with no outgoing links, or cycles around cliques of interconnected pages. So, a certain fraction of the time, simply choose a random page from the Web. This theoretical random walk is known as a Markov chain or Markov process. The limiting probability that an infinitely dedicated random surfer visits any particular page is its PageRank. A page has high rank if other pages with high rank link to it.

Let W be the set of Web pages that can be reached by following a chain of hyperlinks starting at some root page and let n be the number of pages in W . For Google, the set W actually varies with time, but by the end of 2002, n was over 3 billion. Let G be the n -by- n connectivity matrix of a portion of the Web, that is $g_{ij} = 1$ if there is a hyperlink to page i from page j and zero otherwise. The matrix G can be huge, but it is very sparse. Its j th column shows the links on the j th page. The number of nonzeros in G is the total number of

hyperlinks in W . Let r_i and c_j be the row and column sums of G .

$$r_i = \sum_j g_{ij} \quad c_j = \sum_i g_{ij}.$$

The quantities r_j and c_j are the in-degree and out-degree of the j th page. Let p be the probability that the random walk follows a link. A typical value is $p = 0.85$. Then $1 - p$ is the probability that an arbitrary page is chosen. Let \mathbf{A} be the n -by- n matrix whose elements are

$$a_{ij} = p \frac{g_{ij}}{c_j} + \delta,$$

where $\delta = (1 - p)/n$.

Notice that \mathbf{A} comes from scaling the connectivity matrix by its column sums. The j th column is the probability of jumping from the j th page to the other pages on the Web. Most of the elements of \mathbf{A} are equal to δ , the probability of jumping from one page to another without following a link. If $n = 3 \times 10^9$ and $p = 0.85$, then $\delta = 5 \times 10^{-11}$. The matrix \mathbf{A} is the transition probability matrix of the Markov chain. Its elements are all strictly between zero and one.

In particular, its column sums are all equal to one since

$$\sum_i a_{ij} = \sum_i \left[p \frac{g_{ij}}{c_j} + \delta \right] = \frac{p}{c_j} \sum_i g_{ij} + n \frac{1-p}{n} = 1.$$

An important result known as the Perron-Frobenius Theorem applies to such matrices. It concludes that a nonzero solution of the equation

$$\mathbf{x} = \mathbf{A}\mathbf{x}$$

exists and is unique to within a scaling factor. If this scaling factor is chosen so that

$$\sum_i x_i = 1$$

then \mathbf{x} is uniquely determined and is the state vector of the Markov chain. This vector is Google's PageRank. The elements of \mathbf{x} are all positive and less than one.

4. Computing PageRanks

The vector \mathbf{x} is the solution to the singular, homogeneous linear system

$$(\mathbf{I} - \mathbf{A})\mathbf{x} = 0$$

For modest n , an easy way to compute \mathbf{x} in Matlab is to start with some approximate solution, such as the PageRanks from the previous month, or

```
 $\mathbf{x} = \text{ones}(n,1)/n$ 
```

Then simply repeat the assignment statement

```
 $\mathbf{x} = \mathbf{A}*\mathbf{x}$ 
```

until successive vectors agree to within a specified tolerance. This is known as the power method and is about the only possible approach for very large n . In practice, the matrices \mathbf{G} and \mathbf{A} are never actually formed. One step of the power method would be done by one pass over a database of Web pages, updating weighted reference counts generated by the hyperlinks between pages.

The best way to compute PageRank in Matlab is to take advantage of the particular structure of the Markov matrix. The equation

$$\mathbf{x} = \mathbf{A}\mathbf{x}$$

can be written

$$\mathbf{x} = (p\mathbf{GD} + \delta\mathbf{e}\mathbf{e}^T)\mathbf{x}$$

where \mathbf{e} is the n -vector of all ones and \mathbf{D} is the diagonal matrix formed from the reciprocals of the outdegrees, so that

$$d_{jj} = \frac{1}{c_j}$$

We want to have

$$\mathbf{e}^T \mathbf{x} = 1$$

so the equation becomes

$$(\mathbf{I} - p\mathbf{GD})\mathbf{x} = \delta\mathbf{e}$$

As long as p is strictly less than one, the coefficient matrix $\mathbf{I} - p\mathbf{GD}$ is nonsingular and these equations can be solved for \mathbf{x} . This approach

preserves the sparsity of \mathbf{G} , but it breaks down as $p \rightarrow 1$ and $\delta \rightarrow 0$. Once \mathbf{G} has been generated, we need to scale it by its column sums,

```
c = sum(G)
```

It has been proposed that future versions of Matlab allow the expression

```
G./c
```

to divide each column of \mathbf{G} by the corresponding element of c . Until this is available, it is best to use the `spdiags` function to create a sparse diagonal matrix,

```
D = spdiags(1./c',0,n,n)
```

The sparse matrix product $\mathbf{G}^*\mathbf{D}$ will then be computed efficiently. The statements

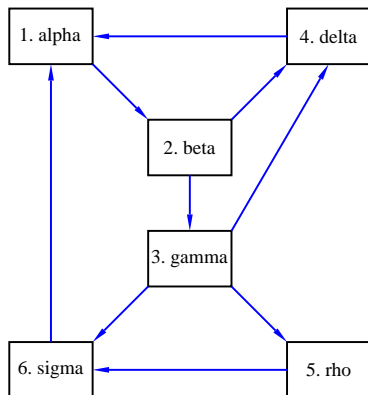
```
p = .85  
delta = (1-p)/n  
e = ones(n,1)  
I = speye(n,n)  
x = (I - p*G*D)\(delta*e)
```

compute PageRank by solving the sparse linear system with Gaussian elimination.

5. An Illustrative Example

The figure here shows the graph for an example involving only $n = 6$ instead of $n = 3 \times 10^9$ web pages. Pages on the Web are identified by strings known as uniform resource locators, or URLs. Most URLs begin with `http` because they use the hypertext transfer protocol. In Matlab we can store the URLs as an array of strings in a cell array. This example involves a 6-by-1 cell array.

```
U = {'http://www.alpha.com'  
'http://www.beta.com'  
'http://www.gamma.com'  
'http://www.delta.com'  
'http://www.rho.com'  
'http://www.sigma.com'}
```



Two different kinds of indexing into cell arrays are possible. Parentheses denote subarrays, including individual cells, and curly braces denote the contents of the cells. If k is a scalar, then $U(k)$ is a 1-by-1 cell array consisting of the k th cell in U , while U_k is the string in that cell. Thus $U(1)$ is a single cell and $U1$ is the string

'http://www.alpha.com'. Think of mail boxes with addresses on a city street. B(502) is the box at number 502, while B502 is the mail in that box. We can generate the connectivity matrix by specifying the pairs of indices (i,j) of the nonzero elements. Because there is a link to beta.com from alpha.com, the (2,1) element of G is nonzero. The nine connections are described by

```
i = [ 2 3 4 4 5 6 1 6 1]
```

```
j = [ 1 2 2 3 3 3 4 5 6]
```

A sparse matrix is stored in a data structure that requires memory only for the nonzero elements and their indices. This is hardly necessary for a 6-by-6 matrix with only 27 zero entries, but it becomes crucially important for larger problems. The statements

```
n = 6
```

```
G = sparse(i,j,1,n,n);
```

```
full(G)
```

generate the sparse representation of an n-by-n matrix with ones in the positions specified by the vectors i and j and display its full representation.

```
0 0 0 1 0 1
1 0 0 0 0 0
0 1 0 0 0 0
0 1 1 0 0 0
0 0 1 0 0 0
0 0 1 0 1 0
```

The statement

```
c = full(sum(G))
```

computes the column sums

```
c =
1 2 3 1 1 1
```

The diagonal matrices I and D are computed as before

```
I = speye(n,n)
D = spdiags(1./c',0,n,n)
```

The statement

```
x = (I - p*G*D)\(delta*e)
```

then solves the sparse linear system to produce

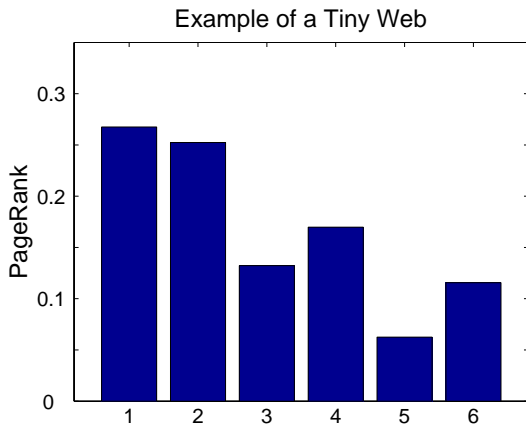
```
x =  
0.2675  
0.2524  
0.1323  
0.1697  
0.0625  
0.1156
```

For this tiny example, the smallest element of the Markov transition matrix is $\delta = .15 = 6 = .0250$.

```
A = p*G*D + delta
```

```
A =  
0.0250 0.0250 0.0250 0.8750 0.0250 0.8750  
0.8750 0.0250 0.0250 0.0250 0.0250 0.0250  
0.0250 0.4500 0.0250 0.0250 0.0250 0.0250  
0.0250 0.4500 0.3083 0.0250 0.0250 0.0250  
0.0250 0.0250 0.3083 0.0250 0.0250 0.0250  
0.0250 0.0250 0.3083 0.0250 0.8750 0.0250
```


Notice that the column sums of A are all equal to one. The bar graph of x is shown in the following figure.



If the URLs are sorted in PageRank order and listed along with their in- and out-degrees, the result is

	PageRank	in	out	url
1	0.2675	2	1	http://www.alpha.com
2	0.2524	1	2	http://www.beta.com
4	0.1697	2	1	http://www.delta.com
3	0.1323	1	3	http://www.gamma.com
6	0.1156	2	1	http://www.sigma.com
5	0.0625	1	1	http://www.rho.com

We see that `alpha` has a higher PageRank than `delta` or `sigma`, even though they all have the same number of links, and that `beta` is ranked second because it basks in `alpha`'s glory. A random surfer will visit `alpha` almost 27% of the time and `rho` just about 6% of the time.

6. Moler's `surfer` Program

Moler's collection of NCM programs includes `surfer.m`. A statement like

```
[U,G] = surfer('http://www.xxx.zzz',n)
```

starts at a specified URL and tries to surf the Web until it has visited `n` pages. If successful, it returns an `n`-by-1 cell array of URLs and an

n-by-n sparse connectivity matrix. The function uses `urldread`, which was introduced in Matlab 6.5, along with underlying Java utilities to access the Web. Surfing the Web automatically is a dangerous undertaking and this function must be used with care. Some URLs contain typographical errors and illegal characters. There is a list of URLs to avoid that includes `.gif` files and Web sites known to cause difficulties. Most importantly, **surfer** can get completely bogged down trying to read a page from a site that appears to be responding, but that never delivers the complete page. When this happens, it may be necessary to have the computer's operating system ruthlessly terminate Matlab. With these precautions in mind, you can use `surfer` to generate your own PageRank examples.

The statement

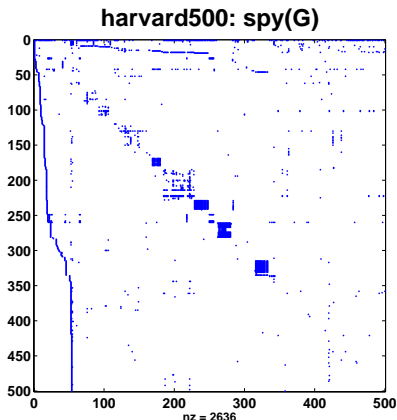
```
[U,G] = surfer('http://www.harvard.edu',500)
```

accesses the home page of Harvard University and generates a 500-by-500 test case. The graph generated in August, 2003, is available in the NCM directory. The statements

```
load harvard500
```

`spy(G)`

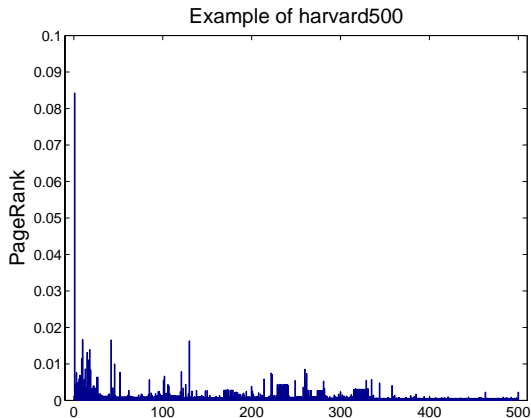
produce a spy plot that shows the nonzero structure of the connectivity matrix.



The statement
`pagerank(U,G)`

computes page ranks, produces a bar graph of the ranks, and prints the most highly ranked URLs in PageRank order. For the **harvard500** data, the dozen most highly ranked pages are

	PageRank	in	out	url
1	0.0823	195	26	http://www.harvard.edu
10	0.0161	21	18	http://www.hbs.edu
42	0.0161	42	0	http://search.harvard.edu:8765/custom/query
130	0.0160	24	12	http://www.med.harvard.edu
18	0.0135	45	46	http://www.gse.harvard.edu
15	0.0129	16	49	http://www.hms.harvard.edu
9	0.0112	21	27	http://www.ksg.harvard.edu
17	0.0109	13	6	http://www.hsph.harvard.edu
46	0.0097	18	21	http://www.gocrimson.com
13	0.0084	9	1	http://www.hsdm.med.harvard.edu
260	0.0083	26	1	http://search.harvard.edu:8765/query.html
19	0.0081	23	21	http://www.radcliffe.edu



The URL where the search began, `www.harvard.edu`, dominates. Like most universities, Harvard is organized into various colleges and institutes, including the Kennedy School of Government, the Harvard Medical School, the Harvard Business School, and the Radcliffe Institute. You can see that the home pages of these schools have high

PageRank. With a different sample, such as the one generated by Google itself, the ranks would be different.

References

- [1] Google is a trademark of Google inc., Mountain View, CA, USA.
[3](#)
- [2] PageRank is a trademark of Google inc., Mountain View, CA, USA. It is protected by US Patent 6,285,999. [3](#)
- [3] Some of the materials here are adopted from C. Moler, *Numerical Computing with Matlab* at the *Mathworks site*.