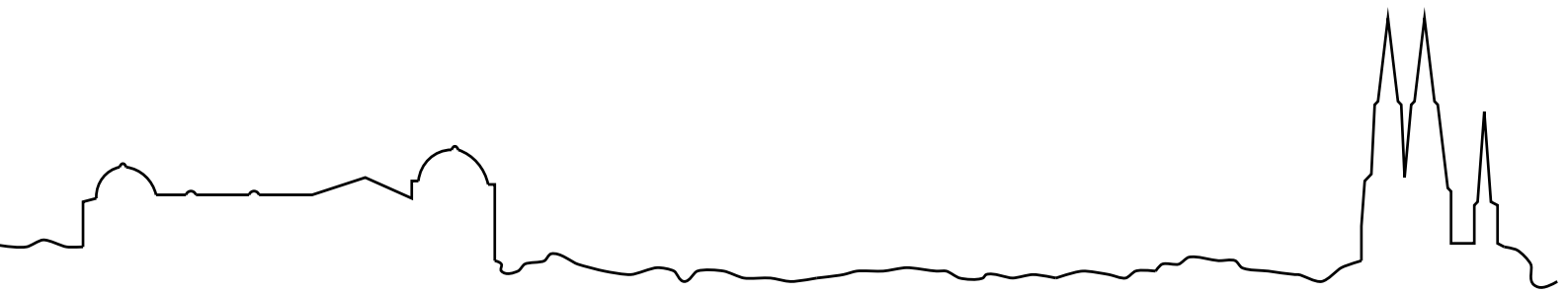
# Investigating Google's PageRank algorithm

by

Erik Andersson
eran3133@student.uu.se

Per-Anders Ekström
peek5081@student.uu.se

Report in Scientific Computing, advanced course - Spring 2004




UPPSALA UNIVERSITET
Inst. för informationsteknologi
Avd. för teknisk databehandling

UPPSALA UNIVERSITY
Information Technology
Dept. of Scientific Computing

# 1 Introduction

Search engines are huge power factors on the Web, guiding people to information and services. Google[1] is the most successfull search engine in recent years, mostly due to its very comprehensive and accurate search results. When Google was an early research project at Stanford, several papers were written describing the underlying algorithms [2] [3]. The dominant algorithm was called PageRank and is still the key for providing accurate rankings for search results.

Google uses the Power method to compute PageRank. For the whole Internet and larger domains this is probably the only possible method (principally due to the high memory-requirements of other methods). In the Power method a number (50-100) of matrix vector multiplications are performed.

For smaller domains, other methods than the Power method would be interesting to investigate. One good candidate is the Arnoldi method which has higher memory requirements but converges after less iterations.

To efficiently handle these large-scale computations we need to implement the algorithms using a parallel system. Some sort of load balancing might be needed to get good performance for the parallelization.

A Web-crawler needs to be implemented to gather realistic test data.

In this review we investigate these methods.

# 2 Review of PageRank

In this following section we present the basic ideas of PageRank. We also describe various problems for calculating PageRank and how to resolve them.

## 2.1 PageRank explained

The Internet can be seen as a large graph, where the Web pages themselves represent nodes, and their links (direct connection to other Web pages) can be seen as the edges of the graph. The links (edges) are directed; i.e. a link only points one way, although there is nothing stopping the other page from pointing back. This interpretation of the Web opens many doors when it comes to creating algorithms for deciphering and ranking the world's Web-pages.

The PageRank algorithm is at the heart of the Google search engine. It is this algorithm that in essence decides how important a specific page is and therefore how high it will show up in a search result.

The underlying idea for the PageRank algorithm is the following: *a page is important, if other important pages link to it.* This idea can be seen as a way of calculating the importance of pages by voting for them. Each link is viewed as a vote - a de facto

[1] http://www.google.com

recommendation for the importance of a page - whatever reasons the page has for linking to a specific page. The PageRank-algorithm can, with this interpretation, be seen as the counter of an online ballot, where pages vote for the importance of others, and this result is then tallied by PageRank and is reflected in the search results.

However, not all votes are equally important. A vote from a page with low importance (i.e. it has few *inlinks*[2]) should be worth far less than a vote from an important page (with thousands of inlinks). Also, each vote's importance is divided by the number of different votes a page casts, i.e. with a single *outlink*[3] all the weight is put towards the sole linked page, but if 100 outlinks are present, they all get a 1/100th of the total weight.

For $n$ pages $P_i, i = 1, 2, \ldots, n$ the corresponding PageRank is set to $r_i, i = 1, 2 \ldots, n$. The mathematical formulation for the recursively defined PageRank are presented in equation (1):

$$r_i = \sum_{j \in L_i} r_j / N_j, \qquad i = 1, 2, \ldots, n. \tag{1}$$

where $r_i$ is the PageRank of page $P_i$, $N_j$ is the number of outlinks from page $P_j$ and $L_i$ are the pages that link to page $P_i$.

Since this is a recursive formula an implementation needs to be iterative and will require several iterations before stabilizing to an acceptable solution. Equation (1) can be solved in an iterative fashion using algorithm (2.1):

**Algorithm 2.1** PageRank

$r_i^{(0)}, \quad i = 1, 2 \ldots, n.$ arbitrary nonzero starting value
**for** $k = 0, 1, \ldots$ **do**
  $r_i^{(k+1)} = \sum_{j \in L_i} \frac{r_j^{(k)}}{N_j}, \quad i = 1, 2 \ldots, n.$
  **if** $\|\mathbf{r}^{(k)} - \mathbf{r}^{(k+1)}\|_1 < tolerance$ **then**
    **break**
  **end if**
**end for**

You start with an arbitrarily guessed vector $r$ (e.g. a vector of ones, all divided with number of pages present), that describes the initial PageRank value $r_i$ for all pages $P_i$. Then you iterate the recursive formula until two consecutively iterated PageRank vectors are similar enough.

[2] An inlink is a link that points to the current page from another page.
[3] An outlink is a link that points out from the current page to another page.

## 2.2 Matrix model

By defining a matrix

$$Q_{ij} := \begin{cases} 1/N_i & \text{if } P_i \text{ links to } P_j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

the PageRank problem can be seen as a matrix-problem. The directed graph in Figure 1 exemplifies a very small isolated part of the Web with only 6 Web-pages, $P_1, P_2, \ldots, P_6$.

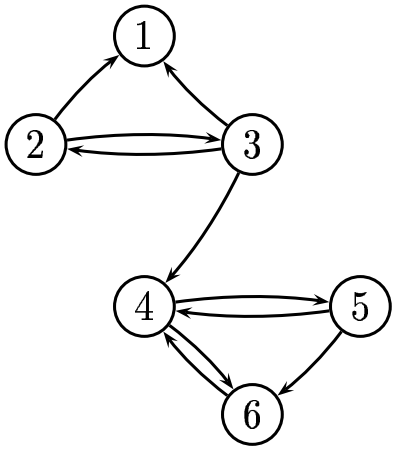



Figure 1: Small isolated Web site of 6 pages $P_1, P_2, \ldots, P_6$

In the matrix-formulation, this link structure will be written as:

$$\mathbf{Q} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \tag{3}$$

Here $Q_{ij}$ describes that there is a link from page $P_i$ to page $P_j$, and these are all divided by $N_i$ (which is the number of outlinks on page $P_i$).

The iteratively calculated PageRank $\mathbf{r}$ could then be written as:

$$\mathbf{r}_{(k+1)}^T = \mathbf{r}_{(k)}^T \mathbf{Q} \qquad , k = 0, 1, \ldots \tag{4}$$

i.e. the Power method.

## 2.3 Random walker

To better explain and visualize the problems and concepts of the PageRank-algorithm in an intuitive fashion, a random walker model of the web can be used. This random walker

(or surfer) starts from a random page, and then selects one of the outlinks from the page in a random fashion. The PageRank (importance) of a specific page can now be viewed as the asymptotic probability that the surfer is present at the page. This is possible, as the surfer is more likely to randomly wander to pages with many votes (lots of inlinks), giving him a large probabibility of ending up in such pages.

### 2.3.1 Stuck at a page

The random walker described above will run into difficulties on his trek around the web. As he randomly "wanders" through the link structure he might reach a page that has no outlinks - forever confining him to this page. For the small Web shown in figure 1 this will happen if the random walker goes to page $P_1$. The corresponding link-matrix has a row of zeros at every page without outlinks. How can this problem be solved?

The method used in the PageRank-algorithm is the following:
Replace all zeros with $1/n$ in all the zero-rows, where $n$ is the dimension of the matrix.

In our matrix formulation, this can be written as:

$$\hat{\mathbf{Q}} = \mathbf{Q} + \frac{1}{n}\mathbf{d}\mathbf{e}^T \tag{5}$$

where $\mathbf{e}$ is a column-vector of ones, and $\mathbf{d}$ is a column-vector that describe which rows in the matrix $\mathbf{Q}$ that are all zero, it's defined as

$$d_i := \begin{cases} 1 & \text{if } N_i = 0 \\ 0 & \text{otherwise} \end{cases}, i = 1, 2, \ldots, n. \tag{6}$$

For our example matrix this addition would be:

$$\hat{\mathbf{Q}} = \mathbf{Q} + \frac{1}{n}\mathbf{d}\mathbf{e}^T = \mathbf{Q} + \frac{1}{6}\begin{pmatrix}1\\0\\0\\0\\0\\0\end{pmatrix}\begin{pmatrix}1 & 1 & 1 & 1 & 1 & 1\end{pmatrix} =$$

$$= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} + \begin{pmatrix} \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

With the creation of matrix $\hat{\mathbf{Q}}$, we have a row-stochastic matrix, i.e. a matrix where all rows sums up to 1.

### 2.3.2 Stuck in a subgraph

There is still one possible pitfall for our random walker, he can wander into a subsection of the complete graph that does not link to any outside pages, locking him into a small part of the web. For the small Web shown in Figure 1 this will happen if the walker comes down to the lower part of the structure. If he ends up in this section, there are no possibilities for him to return to the upper part. In the link-matrix described above this corresponds to an reducible matrix.

This means that if he gets to the enclosed subsection he will randomly wander inside this specific subgraph, and the asymptotic probability that he will be in one of these pages will increase with each random step. We therefore want the matrix to be irreducible, making sure he can not get stuck in a subgraph.

The method used in PageRank to guarantee irreducibility is something called "teleportation", the ability to jump, with a small probability, from any page in the linkstructure to any other page. This can mathematically be described as:

$$\hat{\hat{\mathbf{Q}}} = \alpha\hat{\mathbf{Q}} + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T \tag{7}$$

where $\mathbf{e}$ is a column-vector of ones, and $\alpha$ is a dampening factor (i.e. the "teleportation" probability factor). For our example matrix and an $\alpha$ set to 0.85 this addition would be:

$$\begin{aligned}
\hat{\hat{\mathbf{Q}}} = \alpha\hat{\mathbf{Q}} + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T = \\
= 0.85\begin{pmatrix} \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} + (1-0.85)\frac{1}{6}\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} = \\
= \frac{17}{20}\begin{pmatrix} \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} + \frac{1}{40}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} = \\
= \begin{pmatrix} 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 11/12 & 1/60 & 11/12 & 1/60 & 1/60 & 1/60 \\ 19/60 & 19/60 & 1/60 & 19/60 & 1/60 & 1/60 \\ 1/60 & 1/60 & 1/60 & 1/60 & 7/15 & 7/15 \\ 1/60 & 1/60 & 1/60 & 7/15 & 7/15 & 1/60 \\ 1/60 & 1/60 & 1/60 & 11/12 & 1/60 & 1/60 \end{pmatrix}
\end{aligned}$$

With the creation of matrix $\hat{\hat{\mathbf{Q}}}$, we have an *irreducible matrix*[4].

[4] http://mathworld.wolfram.com/IrreducibleMatrix.html

When adding $(1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T$ there is an equal chance of jumping to all pages. Instead of $\mathbf{e}^T$ we can use a weighted vector, having different probabilities for certain pages - this give us power to bias the end-result for our own needs.

### 2.4 The selection of $\alpha$

It may be shown [4] that if our matrix $\hat{\mathbf{Q}}^T$ has eigenvalues: $\{1, \lambda_2, \lambda_3, \ldots\}$ our new matrix $\hat{\hat{\mathbf{Q}}}^T$ will have the eigenvalues: $\{1, \alpha\lambda_2, \alpha\lambda_3, \ldots\}$.

The value $\alpha$ therefore heavily influences our calculations. It can be shown that the Power method approximately converges at the rate of $C \mid \lambda_2/\lambda_1 \mid^{(m)}$, and as we created our final matrix $\hat{\mathbf{Q}}^T$, we scaled down all eigenvalues (as shown above) except the largest one with the factor $\alpha$.

- A small $\alpha$ ($\approx 0.7$) would therefore mean that the Power method converges quickly. This also means that our final result would poorly describe the underlying link structure as we allow the teleportation to heavily influence the result.
- A large $\alpha$ ($\approx 0.9$) on the other hand means that the Power method converges slowly, but the answer better describes the properties of the real underlying link structure.

As a good compromise, Google uses an $\alpha$ of 0.85 [2].

### 2.5 Practical calculations of PageRank

$\hat{\hat{\mathbf{Q}}}$ is an irreducible row-stochastic matrix. According to Perron-Frobenius Theory [4], an irreducible column-stochastic matrix has 1 as the largest eigenvalue and its corresponding right eigenvector has only non-negative elements. That is the reason we do a left hand multiplication. We now have everything we need to compute PageRank, and the final formula becomes:

$$\hat{\hat{\mathbf{Q}}}^T\mathbf{r} = \mathbf{r} \tag{8}$$

The form of the problem written in equation (8) is the classic definition of the eigenvalue/vector-problem, and the goal of finding the importance of all pages transforms into the problem of finding the eigenvector corresponding to the largest eigenvalue of 1.

As written above, the matrices used in the PageRank-calculations are immense, but it can be shown that we do not have to create the full matrix $\hat{\hat{\mathbf{Q}}}^T$, nor the somewhat full matrix $\hat{\mathbf{Q}}^T$ explicitly to correctly calculate PageRank. We can instead directly use our very sparse link matrix $\mathbf{Q}$, which was initially created to describe the link structure together with two more sparse matrices, as in equation (9).

$$\mathbf{r} = \hat{\hat{\mathbf{Q}}}^T\mathbf{r} = \alpha\hat{\mathbf{Q}}^T\mathbf{r} + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T\mathbf{r} = \alpha\mathbf{Q}^T\mathbf{r} + \alpha\frac{1}{n}\mathbf{e}\mathbf{d}^T\mathbf{r} + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T\mathbf{r} \tag{9}$$

Internet Mathematics Vol. 1, No. 3: 335-380

# Deeper Inside PageRank

Amy N. Langville and Carl D. Meyer

**Abstract.** This paper serves as a companion or extension to the "Inside PageRank" paper by Bianchini et al. [Bianchini et al. 03]. It is a comprehensive survey of all issues associated with PageRank, covering the basic PageRank model, available and recommended solution methods, storage issues, existence, uniqueness, and convergence properties, possible alterations to the basic model, suggested alternatives to the traditional solution methods, sensitivity and conditioning, and finally the updating problem. We introduce a few new results, provide an extensive reference list, and speculate about exciting areas of future research.

## 1. Introduction

Many of today's search engines use a two-step process to retrieve pages related to a user's query. In the first step, traditional text processing is done to find all documents using the query terms, or related to the query terms by semantic meaning. This can be done by a look-up into an inverted file, with a vector space method, or with a query expander that uses a thesaurus. With the massive size of the web, this first step can result in thousands of retrieved pages related to the query. To make this list manageable for a user, many search engines sort this list by some ranking criterion. One popular way to create this ranking is to exploit the additional information inherent in the web due to its hyperlinking structure. Thus, link analysis has become the means to ranking. One successful and well-publicized link-based ranking system is PageRank, the ranking system used by the Google search engine. Actually, for pages related to a query, an IR (Information Retrieval) score is combined with a PR (PageRank) score to deter-


1542-7951/04 $0.50 per page

mine an overall score, which is then used to rank the retrieved pages [Blachman 03]. This paper focuses solely on the PR score.

We begin the paper with a review of the most basic PageRank model for determining the importance of a web page. This basic model, so simple and elegant, works well, but part of the model's beauty and attraction lies in its seemingly endless capacity for "tinkering." Some such tinkerings have been proposed and tested. In this paper, we explore these previously suggested tinkerings to the basic PageRank model and add a few more suggestions and connections of our own. For example, why has the PageRank convex combination scaling parameter traditionally been set to .85? One answer, presented in Section 5.1, concerns convergence to the solution. However, we provide another answer to this question in Section 7 by considering the sensitivity of the problem. Another area of fiddling is the uniform matrix $\mathbf{E}$ added to the hyperlinking Markov matrix $\mathbf{P}$. What other alternatives to this uniform matrix exist? In Section 6.3, we present the common answer, followed by an analysis of our alternative answer. We also delve deeper into the PageRank model, discussing convergence in Section 5.5.1; sensitivity, stability, and conditioning in Section 7; and updating in Section 8. The numerous alterations to and intricacies of the basic PageRank model presented in this paper give an appreciation of the model's beauty and usefulness, and hopefully, will inspire future and greater improvements.

## 2. The Scene in 1998

The year 1998 was a busy year for link analysis models. On the East Coast, a young scientist named Jon Kleinberg, an assistant professor in his second year at Cornell University, was working on a web search engine project called HITS. His algorithm used the hyperlink structure of the web to improve search engine results, an innovative idea at the time, as most search engines used only textual content to return relevant documents. He presented his work [Kleinberg 99], begun a year earlier at IBM, in January 1998 at the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms held in San Francisco, California. Very nearby, at Stanford University, two PhD candidates were working late nights on a similar project called PageRank. Sergey Brin and Larry Page, both computer science students, had been collaborating on their web search engine since 1995. By 1998, things were really starting to accelerate for these two scientists. They were using their dorm rooms as offices for the fledgling business, which later became the giant Google. By August 1998, both Brin and Page took a leave of absence from Stanford in order to focus on their growing business. In a public presentation at the Seventh International World Wide Web conference (WWW98) in Brisbane,

Australia, their paper "The PageRank Citation Ranking: Bringing Order to the Web" [Brin et al. 98b] made small ripples in the information science community that quickly turned into waves. The connections between the two models are striking (see [Langville and Meyer 03]) and it's hard to say whether HITS influenced PageRank, or vice versa, or whether both developed independently. Nevertheless, since that eventful year, PageRank has emerged as the dominant link analysis model, partly due to its query-independence, its virtual immunity to spamming, and Google's huge business success. Kleinberg was already making a name for himself as an innovative academic, and unlike Brin and Page, did not try to develop HITS into a company. However, later entrepreneurs did; the search engine Teoma uses an extension of the HITS algorithm as the basis of its underlying technology [Sherman 02]. As a side note, Google kept Brin and Page busy and wealthy enough to remain on leave from Stanford. This paper picks up after their well-cited original 1998 paper and explores the numerous suggestions that have been made to the basic PageRank model, thus, taking the reader deeper inside PageRank. We note that this paper describes methods invented by Brin and Page, which were later implemented into their search engine Google. Of course, it is impossible to surmise the details of Google's implementation since the publicly disseminated details of the 1998 papers [Brin et al. 98a, Brin and Page 98, Brin et al. 98b]. Nevertheless, we do know that PageRank remains "the heart of [Google's] software ... and continues to provide the basis for all of [their] web search tools," as cited directly from the Google web page, http://www.google.com/technology/index.html.

## 3. The Basic PageRank Model

The original Brin and Page model for PageRank uses the hyperlink structure of the web to build a Markov chain with a primitive[1] transition probability matrix $\mathbf{P}$. The irreducibility of the chain guarantees that the long-run stationary vector $\boldsymbol{\pi}^T$, known as the PageRank vector, exists. It is well-known that the power method applied to a primitive matrix will converge to this stationary vector. Further, the convergence rate of the power method is determined by the magnitude of the subdominant eigenvalue of the transition rate matrix [Stewart 94].

[1] A matrix is *irreducible* if its graph shows that every node is reachable from every other node. A nonnegative, irreducible matrix is *primitive* if it has only one eigenvalue on its spectral circle. An irreducible Markov chain with a primitive transition matrix is called an aperiodic chain. Frobenius discovered a simple test for primitivity: the matrix $\mathbf{A} \geq 0$ is primitive if and only if $\mathbf{A}^m > 0$ for some $m > 0$ [Meyer 00]. This test is useful in determining whether the power method applied to a matrix will converge.

### 3.1. The Markov Model of the Web

We begin by showing how Brin and Page, the founders of the PageRank model, force the transition probability matrix, which is built from the hyperlink structure of the web, to be stochastic and primitive. Consider the hyperlink structure of the web as a directed graph. The nodes of this digraph represent web pages and the directed arcs represent hyperlinks. For example, consider the small document collection consisting of six web pages linked as in Figure 1.

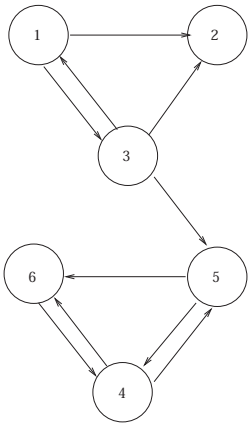



**Figure 1**. Directed graph representing web of six pages

The Markov model represents this graph with a square matrix $\mathbf{P}$ whose element $p_{ij}$ is the probability of moving from state $i$ (page $i$) to state $j$ (page $j$) in one time-step. For example, assume that, starting from any node (web page), it is equally likely to follow any of the outgoing links to arrive at another node. Thus,

$$\mathbf{P} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}.$$

Any suitable probability distribution may be used across the rows. For example, if web usage logs show that a random surfer accessing page 1 is twice as likely to jump to page 2 as he or she is to jump to page 3, then the first row of $\mathbf{P}$, denoted $\mathbf{p}_1^T$, becomes

$$\mathbf{p}_1^T = \begin{pmatrix} 0 & 2/3 & 1/3 & 0 & 0 & 0 \end{pmatrix}.$$

(Similarly, column $i$ of $\mathbf{P}$ is denoted $\mathbf{p}_i$.) Another weighting scheme is proposed in [Baeza-Yates and Davis 04]. One problem with solely using the web's hyperlink structure to build the Markov matrix is apparent. Some rows of the matrix, such as row 2 in our example above, contain all zeroes. Thus, $\mathbf{P}$ is not stochastic. This occurs whenever a node contains no outlinks; many such nodes exist on the web. Such nodes are called dangling nodes. One remedy is to replace all zero rows, $\mathbf{0}^T$, with $\frac{1}{n}\mathbf{e}^T$, where $\mathbf{e}^T$ is the row vector of all ones and $n$ is the order of the matrix. The revised transition probability matrix called $\bar{\mathbf{P}}$ is

$$\bar{\mathbf{P}} = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/3 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

(We note that the uniform vector $\frac{1}{n}\mathbf{e}^T$ can be replaced with a general probability vector $\mathbf{v}^T > 0$. See Section 6.2. for more details about this *personalization vector* $\mathbf{v}^T$.) However, this adjustment alone is not enough to insure the existence of the stationary vector of the chain, i.e., the PageRank vector. Were the chain irreducible, the PageRank vector is guaranteed to exist. By its very nature, with probability 1, the web unaltered creates a reducible Markov chain. (In terms of graph theory, the web graphs are nonbipartite and not necessarily strongly connected.) Thus, one more adjustment, to make $\mathbf{P}$ irreducible, is implemented. The revised stochastic and irreducible matrix $\bar{\bar{\mathbf{P}}}$ is

$$\bar{\bar{\mathbf{P}}} = \alpha\bar{\mathbf{P}} + (1-\alpha)\mathbf{e}\mathbf{e}^T/n = \begin{pmatrix} 1/60 & 7/15 & 7/15 & 1/60 & 1/60 & 1/60 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 19/60 & 19/60 & 1/60 & 1/60 & 19/60 & 1/60 \\ 1/60 & 1/60 & 1/60 & 1/60 & 7/15 & 7/15 \\ 1/60 & 1/60 & 1/60 & 7/15 & 1/60 & 7/15 \\ 1/60 & 1/60 & 1/60 & 11/12 & 1/60 & 1/60 \end{pmatrix}$$

where $0 \leq \alpha \leq 1$ and $\mathbf{E} = \frac{1}{n}\mathbf{e}^T$. This convex combination of the stochastic matrix $\bar{\mathbf{P}}$ and a stochastic perturbation matrix $\mathbf{E}$ insures that $\bar{\bar{\mathbf{P}}}$ is both stochastic and irreducible. Every node is now directly connected to every other node, making the chain irreducible by definition. Although the probability of transitioning may be very small in some cases, it is always nonzero. The irreducibility adjustment also insures that $\bar{\bar{\mathbf{P}}}$ is primitive, which implies that the power method will converge to the stationary PageRank vector $\boldsymbol{\pi}^T$.

## 4. Storage Issues

The size of the Markov matrix makes storage issues nontrivial. In this section, we provide a brief discussion of more detailed storage issues for implementation. The 1998 paper by Brin and Page [Brin and Page 98] and more recent papers by Google engineers [Barroso et al. 03, Ghemawat et al. 03] provide detailed discussions of the many storage schemes used by the Google search engine for all parts of its information retrieval system. The excellent survey paper by Arasu et al. [Arasu et al. 01] also provides a section on storage schemes needed by a web search engine. Since this paper is mathematically oriented, we focus only on the storage of the mathematical components, the matrices and vectors, used in the PageRank part of the Google system.

For subsets of the web, the transition matrix $\mathbf{P}$ (or its graph) may or may not fit in main memory. For small subsets of the web, when $\mathbf{P}$ fits in main memory, computation of the PageRank vector can be implemented in the usual fashion. However, when the $\mathbf{P}$ matrix does not fit in main memory, researchers must be more creative in their storage and implementation of the essential components of the PageRank algorithm. When a large matrix exceeds a machine's memory, researchers usually try one of two things: they compress the data needed so that the compressed representation fits in main memory and then creatively implement a modified version of PageRank on this compressed representation, or they keep the data in its uncompressed form and develop I/O-efficient implementations of the computations that must take place on the large, uncompressed data.

You may skip this

For modest web graphs for which the transition matrix $\mathbf{P}$ can be stored in main memory, compression of the data is not essential, however, some storage techniques should still be employed to reduce the work involved at each iteration. For example, the $\mathbf{P}$ matrix is decomposed into the product of the inverse of the diagonal matrix $\mathbf{D}$ holding outdegrees of the nodes and the adjacency matrix $\mathbf{G}$ of 0s and 1s is useful in saving storage and reducing work at each power iteration. The decomposition $\mathbf{P} = \mathbf{D}^{-1}\mathbf{G}$ is used to reduce the number of multiplications required in each $\mathbf{x}^T\mathbf{P}$ vector-matrix multiplication needed by the power method. Without the $\mathbf{P} = \mathbf{D}^{-1}\mathbf{G}$ decomposition, this requires $nnz(\mathbf{P})$ multiplications and $nnz(\mathbf{P})$ additions, where $nnz(\mathbf{P})$ is the number of nonzeroes in $\mathbf{P}$. Using the vector $diag(\mathbf{D}^{-1})$, $\mathbf{x}^T\mathbf{P}$ can be accomplished as $\mathbf{x}^T\mathbf{D}^{-1}\mathbf{G} = (\mathbf{x}^T).*(diag(\mathbf{D}^{-1}))\mathbf{G}$, where $.*$ represents component-wise multiplication of the elements in the two vectors. The first part, $(\mathbf{x}^T).*(diag(\mathbf{D}^{-1}))$ requires $n$ multiplications. Since $\mathbf{G}$ is an adjacency matrix, $(\mathbf{x}^T).*(diag(\mathbf{D}^{-1}))\mathbf{G}$ now requires an additional $nnz(\mathbf{P})$ additions for a total savings of $nnz(\mathbf{P}) - n$ multiplications. In addition, for large matrices, compact storage schemes

You may skip this

[Barrett et al. 94], such as compressed row storage or compressed column storage, are often used. Of course, each compressed format, while saving some storage, requires a bit more overhead for matrix operations.

Rather than storing the full matrix or a compressed version of the matrix, web-sized implementations of the PageRank model store the $\mathbf{P}$ or $\mathbf{G}$ matrix in an adjacency list of the columns of the matrix [Raghavan and Garcia-Molina 01a]. In order to compute the PageRank vector, the PageRank power method (defined in Section 5.1) requires vector-matrix multiplications of $\mathbf{x}^{(k-1)T}\mathbf{P}$ at each iteration $k$. Therefore, quick access to the columns of the matrix $\mathbf{P}$ (or $\mathbf{G}$) is essential to algorithm speed. Column $i$ contains the inlink information for page $i$, which, for the PageRank system of ranking web pages, is more important than the outlink information contained in the rows of $\mathbf{P}$ or $\mathbf{G}$. For the tiny six-node web from Section 3, an adjacency list representation of the columns of $\mathbf{G}$ is:

| Node | Inlinks from |
|---|---|
| 1 | 3 |
| 2 | 1, 3 |
| 3 | 1 |
| 4 | 5, 6 |
| 5 | 3, 4 |
| 6 | 4, 5 |

Exercise 2.24 of Cleve Moler's recent book [Moler 04] gives one possible implementation of the power method applied to an adjacency list, along with sample MATLAB code. When the adjacency list does not fit in main memory, references [Raghavan and Garcia-Molina 01a, Raghavan and Garcia-Molina 03] suggest methods for compressing the data. Some references [Chen et al. 02a, Haveliwala 99] take the other approach and suggest I/O-efficient implementations of PageRank. Since the PageRank vector itself is large and completely dense, containing over 4.3 billion pages, and must be consulted in order to process each user query, Haveliwala [Haveliwala 02a] has suggested a technique to compress the PageRank vector. This encoding of the PageRank vector hopes to keep the ranking information cached in main memory, thus speeding query processing.

Because of their potential and promise, we briefly discuss two methods for compressing the information in an adjacency list, the gap technique [Bharat et al. 98] and the reference encoding technique [Raghavan and Garcia-Molina 01b, Raghavan and Garcia-Molina 03]. The gap method exploits the locality of hyperlinked pages. The source and destination pages for a hyperlink are often close to each other lexicographically. A page labeled 100 often has inlinks from pages nearby such as pages 112, 113, 116, and 117 rather than pages 117,924 and 4,931,010). Based on this locality principle, the information in an adjacency

You may skip this

list for page 100 is stored as follows:

| Node | Inlinks from |
|---|---|
| 100 | 112 0 2 0 |

Storing the gaps between pages compresses storage because these gaps are usually nice, small integers.

The reference encoding technique for graph compression exploits the similarity between web pages. If pages $x$ and $y$ have similar adjacency lists, it is possible to compress the adjacency list of $y$ by representing it in terms of the adjacency list of $x$, in which case $x$ is called a reference page for $y$. Pages within the same domain might often share common outlinks, making the reference encoding technique attractive. Consider the example in Figure 2, taken from [Raghavan and Garcia-Molina 03]. The binary reference vector, which has the same size as the

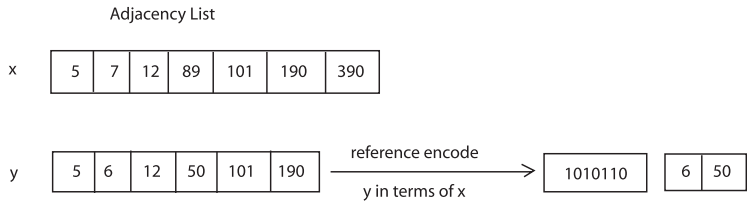



**Figure 2.** Reference encoding example

adjacency list of $x$, contains a 1 in the $i$th position if the corresponding adjacency list entry $i$ is shared by $x$ and $y$. The second vector in the reference encoding is a list of all entries in the adjacency list of $y$ that are not found in the adjacency list of its reference $x$. Reference encoding provides a nice means of compressing the data in an adjacency list, however, for each page one needs to determine which page should serve as the reference page. This is not an easy decision, but heuristics are suggested in [Raghavan and Garcia-Molina 01b]. Both the gap method and the reference encoding method are used, along with other compression techniques, to impressively compress the information in a standard web graph. These techniques are freely available in the graph compression tool WebGraph, which is produced by Paolo Boldi and Sebastiano Vigna [Boldi and Vigna 03, Boldi and Vigna 04].

The final storage issue we discuss concerns dangling nodes. The pages of the web can be classified as either dangling nodes or nondangling nodes. Recall that dangling nodes are web pages that contain no outlinks. All other pages, having at least one outlink, are called nondangling nodes. Dangling nodes exist in many forms. For example, a page of data, a page with a postscript graph, a page with JPEG pictures, a PDF document, a page that has been fetched by a

crawler but not yet explored—these are all examples of possible dangling nodes. As the research community moves more and more material online in the form of PDF and postscript files of preprints, talks, slides, and technical reports, the proportion of dangling nodes is growing. In fact, for some subsets of the web, dangling nodes make up 80 percent of the collection's pages.

The presence of these dangling nodes can cause philosophical, storage, and computational issues for the PageRank problem. We address the storage issue now and save the philosophical and computational issues associated with dangling nodes for the next section. Recall that Google founders Brin and Page suggested replacing $\mathbf{0}^T$ rows of the sparse hyperlink matrix $\mathbf{P}$ with dense vectors (the uniform vector $\frac{1}{n}\mathbf{e}^T$ or the more general $\mathbf{v}^T$ vector) to create the stochastic matrix $\bar{\mathbf{P}}$. Of course, if this suggestion was to be implemented explicitly, storage requirements would increase dramatically. Instead, the stochasticity fix can be modeled implicitly with the construction of one vector $\mathbf{a}$. Element $a_i = 1$ if row $i$ of $\mathbf{P}$ corresponds to a dangling node, and 0, otherwise. Then $\bar{\mathbf{P}}$ (and also $\bar{\bar{\mathbf{P}}}$) can be written as a rank-one update of $\mathbf{P}$.

$$\bar{\mathbf{P}} = \mathbf{P} + \mathbf{a}\mathbf{v}^T, \text{ and therefore, } \bar{\bar{\mathbf{P}}} = \alpha\,\bar{\mathbf{P}} + (1-\alpha)\,\mathbf{e}\mathbf{v}^T = \alpha\,\mathbf{P} + (\alpha\,\mathbf{a} + (1-\alpha)\,\mathbf{e})\mathbf{v}^T.$$

## 5. Solution Methods for Solving the PageRank Problem

Regardless of the method for filling in and storing the entries of $\bar{\bar{\mathbf{P}}}$, PageRank is determined by computing the stationary solution $\boldsymbol{\pi}^T$ of the Markov chain. The row vector $\boldsymbol{\pi}^T$ can be found by solving either the eigenvector problem

$$\boldsymbol{\pi}^T\bar{\bar{\mathbf{P}}} = \boldsymbol{\pi}^T,$$

or by solving the homogeneous linear system

$$\boldsymbol{\pi}^T(\mathbf{I} - \bar{\bar{\mathbf{P}}}) = \mathbf{0}^T,$$

where $\mathbf{I}$ is the identity matrix. Both formulations are subject to an additional equation, the normalization equation $\boldsymbol{\pi}^T\mathbf{e} = 1$, where $\mathbf{e}$ is the column vector of all 1s. The normalization equation insures that $\boldsymbol{\pi}^T$ is a probability vector. The $i$th element of $\boldsymbol{\pi}^T$, $\pi_i$, is the PageRank of page $i$. Stewart's book, *An Introduction to the Numerical Solution of Markov Chains* [Stewart 94], contains an excellent presentation of the various methods of solving the Markov chain problem.

### 5.1. The Power Method

Traditionally, computing the PageRank vector has been viewed as an eigenvector problem, $\boldsymbol{\pi}^T\bar{\bar{\mathbf{P}}} = \boldsymbol{\pi}^T$, and the notoriously slow power method has been the method of choice. There are several good reasons for using the power method. First, consider iterates of the power method applied to $\bar{\bar{\mathbf{P}}}$ (a completely dense matrix, were it to be formed explicitly). Note that $\mathbf{E} = \mathbf{e}\mathbf{v}^T$. For any starting vector $\mathbf{x}^{(0)T}$ (generally, $\mathbf{x}^{(0)T} = \mathbf{e}^T/n$),

$$\begin{aligned}\mathbf{x}^{(k)T} &= \mathbf{x}^{(k-1)T}\bar{\bar{\mathbf{P}}} = \alpha\mathbf{x}^{(k-1)T}\bar{\mathbf{P}} + (1-\alpha)\mathbf{x}^{(k-1)T}\mathbf{e}\mathbf{v}^T \\ &= \alpha\mathbf{x}^{(k-1)T}\bar{\mathbf{P}} + (1-\alpha)\mathbf{v}^T \\ &= \alpha\mathbf{x}^{(k-1)T}\mathbf{P} + (\alpha\mathbf{x}^{(k-1)T}\mathbf{a} + (1-\alpha))\mathbf{v}^T, \end{aligned} \tag{5.1}$$

since $\mathbf{x}^{(k-1)T}$ is a probability vector, and thus, $\mathbf{x}^{(k-1)T}\mathbf{e} = 1$. Written in this way, it becomes clear that the power method applied to $\bar{\bar{\mathbf{P}}}$ can be implemented with vector-matrix multiplications on the extremely sparse $\mathbf{P}$, and $\bar{\bar{\mathbf{P}}}$ and $\bar{\mathbf{P}}$ are never formed or stored. A matrix-free method such as the power method is required due to the size of the matrices and vectors involved (Google's index is currently 4.3 billion pages). Fortunately, since $\mathbf{P}$ is sparse, each vector-matrix multiplication required by the power method can be computed in $nnz(\mathbf{P})$ flops, where $nnz(\mathbf{P})$ is the number of nonzeroes in $\mathbf{P}$. And since the average number of nonzeroes per row in $\mathbf{P}$ is 3-10, $O(nnz(\mathbf{P})) \approx O(n)$. Furthermore, at each iteration, the power method only requires the storage of one vector, the current iterate, whereas other accelerated matrix-free methods, such as restarted GMRES or BiCGStab, require storage of at least several vectors, depending on the size of the subspace chosen. Finally, the power method on Brin and Page's $\bar{\mathbf{P}}$ matrix converges quickly. Brin and Page report success using only 50 to 100 power iterations [Brin et al. 98b].

We return to the issue of dangling nodes now, this time discussing their philosophical complications. In one of their early papers [Brin et al. 98a], Brin and Page report that they "often remove dangling nodes during the computation of PageRank, then add them back in after the PageRanks have converged." From this vague statement it is hard to say exactly how Brin and Page were computing PageRank. But, we are certain that the removal of dangling nodes is not a fair procedure. Some dangling nodes should receive high PageRank. For example, a very authoritative PDF file could have many inlinks from respected sources, and thus, should receive a high PageRank. Simply removing the dangling nodes biases the PageRank vector unjustly. In fact, doing the opposite and incorporating dangling nodes adds little computational effort (see Equation (5.1)), and further, can have a beneficial effect as it can lead to more efficient and accurate computation of PageRank. (See [Lee et al. 03] and the next section.)