School of Mathematical and Statistical Sciences
Arizona State University

Student: Lee Reeves

Other students: Binod Pant, Ronnie Ramirez

Course: APM 505

Program: Mathematics MA

Instructor: Malena Espanol

Date: Fall 2019

Google PageRank Explained via Power Iteration

How Google used linear algebra to bring order to the Web, help people find great websites, and earn billions of dollars

Binod Pant, Ronnie Ramirez, Lee Reeves

December 5, 2019

History

In 1996, Larry Page and Sergei Brin, then PhD students at Stanford University, began working on a search engine called BackRub, running on Stanford's servers. In 1997, they renamed the search engine Google and obtained the domain google.com, and in 1998 they founded the company Google Inc. (Google 2015) By 2004, at their IPO, the company was valued at $27 billion.

Before they became billionaires, Brin and Page, along with professors Rajeev Motwani and Terry Winograd, described the secret sauce behind Google, the PageRank algorithm, in the paper *The PageRank Citation Ranking: Bringing Order to the Web* (Page et al. 1999). PageRank has proven to be immensely valuable, but surprisingly it is a rather simple application of linear algebra. In this paper, we describe the PageRank algorithm as an application of the method of power iteration.

Intuition

Imagine a "random surfer" who clicks on successive links at random. How often this random surfer arrives on a particular page depends on several factors, including how many links lead to the page, how often the random surfer arrives on pages containing those links, and how many links are on those pages.

The qualities of a page the random surfer visits often are also qualities that make for a good ranking. Links from long lists of web pages don't count for much, because the probability of following any one of those links is low. Links from little known pages also don't count for much, because those pages are rarely visited. But links from popular pages that don't link to many other pages greatly increase the probability the random surfer will visit the linked page, and can be thought of as recommendations.

This suggests "how often the random surfer arrives on a page" would make a good ranking, and that's the idea behind PageRank.

Mathematically, this model corresponds to a random walk on the directed graph of the web, and PageRank could be defined as the stationary distribution (or standing probability distribution) of that random walk.

But this simple model has some problems. What happens if the random surfer reaches a page with no outgoing links, or a group of pages that link to each other, but contain no links that lead out of the group? In this simple model, they'll get stuck. But in reality, rather than getting stuck, our random surfer might jump directly to some other page, ignoring links. In fact, there might be a small probability the random surfer would make a direct jump at any time, and that is the behavior modeled by the PageRank algorithm.

That's an intuitive description of the PageRank algorithm, as described in Page et al. 1999, which we will now translate into an application of power iteration:

PageRank as Power Iteration

Let ℓ_j be the number of outgoing links from page j (counting duplicates only once).

We define the link matrix L, whose entries L_{ij} give the probability our random surfer will follow a link from page j to page i, by

$$L_{ij} := \begin{cases} 1/\ell_j, & \text{if page j links to page i} \\ 0, & \text{otherwise} \end{cases}$$

Because each column contains the probabilities of all possible outcomes when following a link from page j, each column must sum to 1, This is clearly true because page j links to ℓ_j pages, so the sum over all destination pages is $\sum_{i=1}^{n} L_{ij} = \ell_j * \frac{1}{\ell_j} + (n - \ell_j) * 0 = 1$. Thus L is a column-stochastic matrix.

We also define d as the probability of making a direct jump, and a jump vector e, whose k^{th} component is the probability a direct jump will lead to page k.

$$e_k := \text{Probability a direct jump ends on page } k$$

With this definition, direct jumps do not care which page they start from. Also, because this vector contains the probabilities of all possible outcomes when making a direct jump, its components must all be nonnegative and must sum to 1 (which limits our choice of e).

Then we define a transition matrix T, combining the effects of following links and making direct jumps, by

$$T_{ij} := \begin{cases} e_i, & \text{if } \ell_j = 0 \text{ (i.e. page j has no outgoing links)} \\ (1-d)L_{ij} + de_i, & \text{otherwise} \end{cases}$$

This first case corresponds to leaving a page with no outgoing links, from which the random surfer always makes a direct jump. And the second case combines following links with probability $(1 - d)$ and making direct jumps with probability d. And once again, because each column contains the probabilities of all possible outcomes for the next page after page

j, every entry must be nonnegative and each column must (and does) sum to 1, and T is a column-stochastic matrix, because if $\ell_j = 0$, then:

$$\sum_{i=1}^{n} T_{ij} = \sum_{i=1}^{n} e_i = 1,$$

and if $\ell_j \neq 0$, then

$$\sum_{i=1}^{n} T_{ij} = \sum_{i=1}^{n} (1-d)L_{ij} + de_i = (1-d)\sum_{i=1}^{n} L_{ij} + d\sum_{i=1}^{n} e_i = (1-d) + d = 1.$$

Finally, we define a state vector R, whose k^{th} component gives the probability our random surfer is currently on page k:

$$R_k := \text{Probability the current page is page } k$$

and once again, because this represents the probabilities of all possible current pages, the components of this vector must all be nonnegative and must sum to 1.

Then the effect of one time step, in which our random surfer clicks on one link, is a simple matrix multiplication:

$$R^{(t+1)} = TR^{(t)}$$

and we can find the PageRank vector R, which is the stationary (or standing probability) distribution, from an initial guess $R^{(0)}$ (again chosen so the components are nonnegative and sum to one), using the method of power iteration:

$$R := \lim_{t\to\infty} T^t R^{(0)}.$$

which preserves the initial invariant, $\sum_{k=1}^{n} R = \sum_{k=1}^{n} R^{(t)} = 1$ for all t. We continue until the change between iterations is less than a chosen tolerance:

$$\text{Stop when } \left\| R^{(t+1)} - R^{(t)} \right\|_1 < \text{desired tolerance.}$$

A Simple Example

To illustrate the behavior of the state vector R, we start with an overly simple example with $d = 0$ (normally we should not use $d = 0$ because the resulting system might not converge).

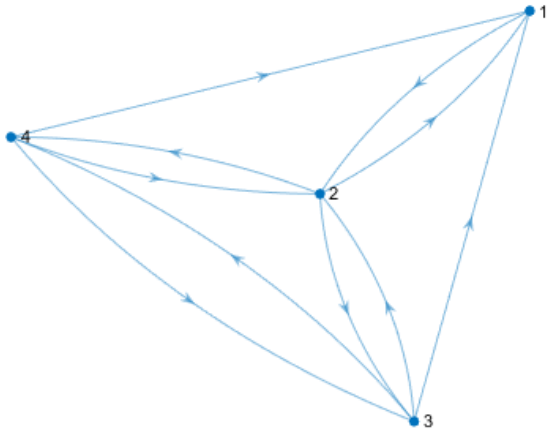

Figure 1: Directed graph for Example 1

t	$R_1^{(t)}$	$R_2^{(t)}$	$R_3^{(t)}$	$R_4^{(t)}$
0	1	0	0	0
1	0	1	0	0
2	0.3333	0	0.3333	0.3333
3	0.2222	0.5556	0.1111	0.1111
4	0.2593	0.2963	0.2222	0.2222
5	0.2469	0.4074	0.1728	0.1728
6	0.2510	0.3621	0.1934	0.1934
7	0.2497	0.3800	0.1852	0.1852
8	0.2501	0.3731	0.1884	0.1884
9	0.2500	0.3757	0.1872	0.1872

Table 1: Iterations of Example 1

We have four pages, each of which links to all others, except for page 1, which only links to page 2. A directed graph for this network is shown in Figure 1. This corresponds to the transition matrix

$$T = L = \begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 1 & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{3} & 0 & \frac{1}{3} \\ 0 & \frac{1}{3} & \frac{1}{3} & 0 \end{bmatrix}$$

We start from page 1, i.e., we are certain the random surfer is on page 1, so $R^{(0)} = (1, 0, 0, 0)^T$. With a tolerance of 0.01, power iteration converges after 9 iterations, with the results shown in Table 1.

After 1 iteration, we have

$$R^{(1)} = TR^{(0)} = (0, 1, 0, 0)^T$$

and we're certain the random surfer is on page 2, having followed the only link on page 1.

After 2 iterations, we have

$$R^{(2)} = TR^{(0)} = (0.3333, 0, 0.3333, 0.3333)^T.$$

Now we're unsure where the random surfer is, and the state vector is a vector of probabilities, with probability $\frac{1}{3}$ each that the random surfer is on page 1, 3, or 4.

And so on, until after 9 iterations, when PageRank converges, we have roughly 37.5% chance that the random surfer is on page 2, 25% chance of page 1, and 18.7% chance each of pages 3 or 4. This seems sensible, because pages 1 and 2 have the most incoming links, and page 1 links only to page 2.

Convergence

The Perron–Frobenius theorem guarantees that, with $d > 0$ and a suitable choice of E (discussed below), T has a real positive eigenvalue r, any other eigenvalue of T must be strictly smaller than r in absolute value, and there exists an eigenvector v corresponding to the eigenvalue r such that all components of v are nonnegative, which we may choose so that $\|v\| = 1$.

Then, because T is column stochastic (all columns sum to one) with only nonnegative entries,

$$\begin{aligned}
\|Tv\|_1 &= \left\| \sum_{j=1}^{n} T_j v_j \right\|_1 && \text{(expanding Tv as a linear combination of columns of T)} \\
&= \sum_{i=1}^{n} \left| \sum_{j=1}^{n} T_{ij} v_j \right| && \text{(by the definition of the 1-norm)} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} |T_{ij} v_j| && \text{(because all entries of T and v are nonnegative real numbers)} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} T_{ij} |v_j| && \text{(because all entries of T are nonnegative real numbers)} \\
&= \sum_{j=1}^{n} \sum_{i=1}^{n} T_{ij} |v_j| && \text{(exchanging the summations)} \\
&= \sum_{j=1}^{n} |v_j| \sum_{i=1}^{n} T_{ij} && \text{(by the distributive property)} \\
&= \sum_{j=1}^{n} |v_j| && \text{(because T is column stochastic)} \\
&= \|v\|_1 && \text{(by the definition of the 1-norm).}
\end{aligned}$$

Therefore

$$\|v\|_1 = \|Tv\|_1 = \|rv\|_1 = r \|v\|_1 ,$$

which implies $r = 1$.

So the transition matrix T has dominant eigenvalue 1, and all other eigenvalues are strictly less than 1 in absolute value.

This means power iteration will converge, as long as the initial guess is not orthogonal to the eigenvector corresponding to the eigenvalue 1. Fortunately, this is extremely unlikely when n is large.

Therefore convergence is almost certain as k goes to infinity, but the rate of convergence is $\lambda_2/1 = \lambda_2$, and we do not know what λ_2 is. In other words, fast convergence is not guaranteed; but Page et. al. reported that the PageRank algorithm converged in just 52 iterations on their database of 322 million pages, and conjectured that the scaling factor was roughly linear in $\log n$, as shown in Figure 2.

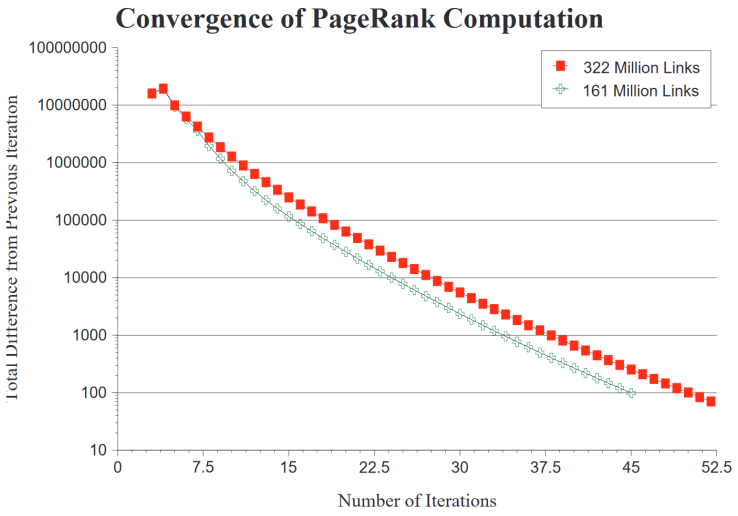

Figure 2: Convergence of PageRank Computation, Page et al. 1999.

A Suitable Choice of e

Earlier we mentioned that the Perron–Frobenius theorem, which guarantees the existence of the eigenvector we've defined as the PageRank, relies on a suitable choice of the direct jump vector e. What is "a suitable choice"?

There are two possibilities. First, if e is strictly positive (every component greater than zero), then T is a positive matrix, and the proof above is valid. This is satisfied, for example, if every component of e is simply $1/n$.

Second, the proof above is also valid if T is irreducible and has period 1, the meaning of which is too technical for this short paper (full details are available from Wikipedia 2019). Fortunately, the theory of Markov Chains supplies a simpler interpretation, which we will simply state without proof.

T can be interpreted as the transition matrix of a Markov Chain, and T is irreducible and has period 1, if the corresponding Markov Chain is irreducible, which means that every page can eventually be reached from any page, and aperiodic, which means that, for every page, the greatest common denominator of the length of all paths from that page to itself is one.

As a practical matter, T is almost certain to be aperiodic if it is irreducible and represents a link structure as complex as the Web.

But there are many pages and groups of pages on the web with no incoming links, and even

if a search engine does not find such pages initially, pages may become disconnected when other pages change.

If such disconnected pages are represented in T, we must ensure that our choice of E results in a digraph (whose nodes are pages and whose edges correspond to nonzero components of T) that is strongly connected.

Why Power Iteration?

Trefethen and Bau tell us that "On its own, power iteration is of limited use, for several reasons. First, it can find only the eigenvector corresponding to the largest eigenvalue. Second, the convergence is linear, reducing the error only by a constant factor $\approx |\lambda_2/\lambda_1|$ at each iteration. Finally, the quality of this factor depends on having a largest eigenvalue that is significantly larger than the others. If the largest two eigenvalues are close in magnitude, the convergence will be very slow." (trefethenNumericalLinearAlgebra1997).

All these concerns are true, but not applicable to PageRank. We are only interested in the eigenvector corresponding to the largest eigenvalue, 1, and although PageRank takes many iterations to converge by a single digit, matrix multiplication ($O(n^2)$) is much faster than solving the system $(A-\mu I)w = v^{k-1}$ used in inverse iteration and Rayleigh quotient iteration ($O(n^3)$).

This is particularly true for sparse matrices, and the link matrix L will be sparse. Even if e (and therefore T) is not sparse, we can use the equivalent calculation

$$R^{(t+1)} = (1-d)LR^{(t)} + de$$

to update the PageRank with a sparse matrix multiplication. So we should briefly discuss sparse matrices.

Sparse Matrices

The term *sparse matrix* may refer either to a matrix in which most of the elements are zero, or to the data structure used to store such a matrix. These data structures have more than one entry per nonzero element, but do not store nonzero elements at all.

Two of the most common sparse matrix formats are compressed sparse column format (used by MATLAB for matrices declared with the `sparse` command) and compressed sparse row format, also called Yale format, which are almost identical.

The compressed sparse column format represents a matrix M with three one-dimensional arrays:

A contains the non-zero elements of M in column-major order (down the first column, then down the second column, and so on).

JA contains the row index of each element in **A**.

IA starts with zero, then contains, for each column, the index in **A** where the column begins. This means $\mathbf{IA}[i] - \mathbf{IA}[i-1]$ is the number of nonzero elements in row $i-1$.

For example, the matrix

$$\begin{bmatrix} 0 & 3 & 0 & 0 \\ 4 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 7 & 0 & 0 & 6 \end{bmatrix}$$

is stored (if indexes start from 1) as:

```
A = [4 7 3 1 6]; JA = [2 4 1 3 4]; IA = [0 2 3 4 5]
```

The compressed sparse row, or Yale, format is identical, except **A** is in row-major order, **JA** contains column indexes, and **IA** contains the index in **A** of the first element of each column.

For real applications of PageRank, we must use sparse matrices. Simply storing a full matrix for 344 million pages in IEEE double-precision would require about 100 terabytes, while storing a sparse matrix with an average of 10 links per page would require less than 100 gigabytes. And each step of power iteration is faster with sparse matrices by a similar factor because the nonzero elements are simply ignored.

Examples

We created an example network with 20 pages (many more would be difficult to see in a single graph) and calculated the PageRank for this network. Figure 3 shows the network as a directed graph, and the resulting PageRank in parentheses next to each node.

The results seem reasonable. Nodes near the edges of the graph (placed there by Matlab because they have few links) have low page rank, and node 14, which has the most incoming links, also has the highest PageRank.

Then, to illustrate the effect of rank sinks, we fully connected nodes 8, 9, and 10, and removed the links leading out of this group, creating a rank sink among three of the (formerly) low ranked pages.

Figure 4 shows the resulting PageRank when $d = 0$. The rank sink has captured 100% of the PageRank, leaving other pages with zero. Then in Figure 5, we used $d = 0.3$, which greatly reduced the amount of PageRank captured by the rank sink. But the rank sink still ranks higher than the same pages in the original network, suggesting that removing outgoing links might have been an effective SEO strategy.

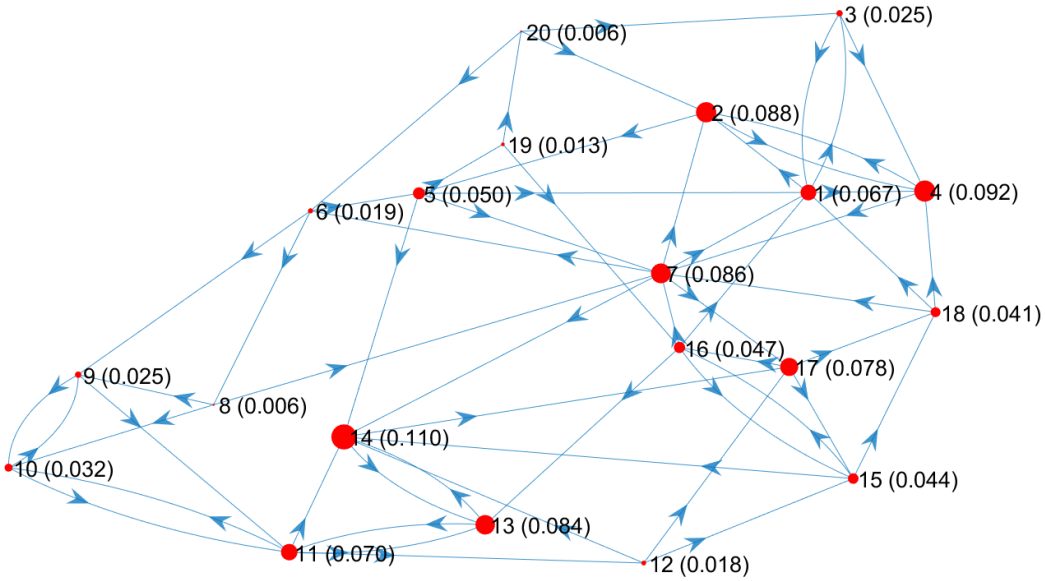

Figure 3: An example network with 20 pages. Nodes represent pages, arrows represent links, and numbers in parentheses are the resulting PageRank.

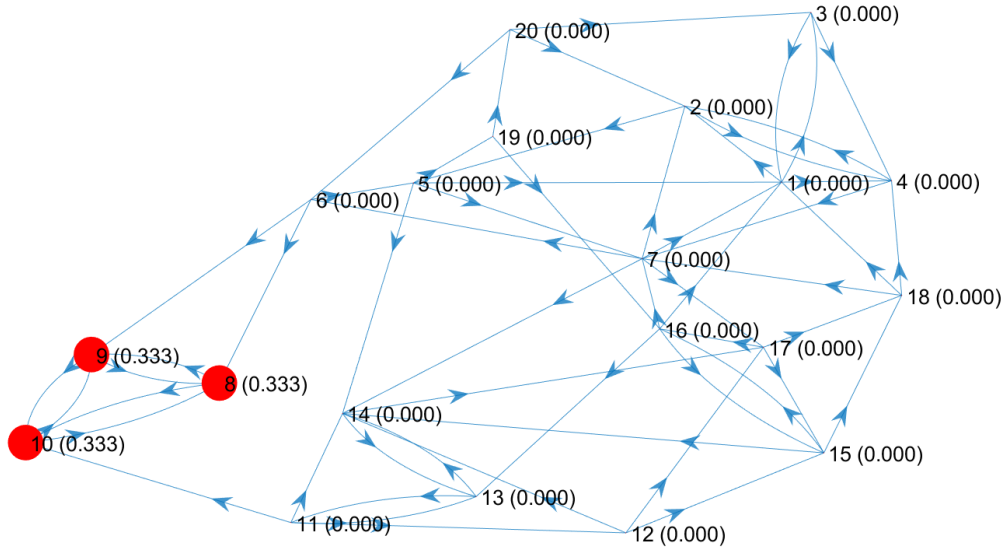

Figure 4: A similar network, but with a rank sink in pages 8, 9, and 10, which are now fully connected and have no links leading to any other pages. For this graph, we calculated the PageRank with d=0, showing that if we do not include direct jumps in the model, a rank sink captures all PageRank.

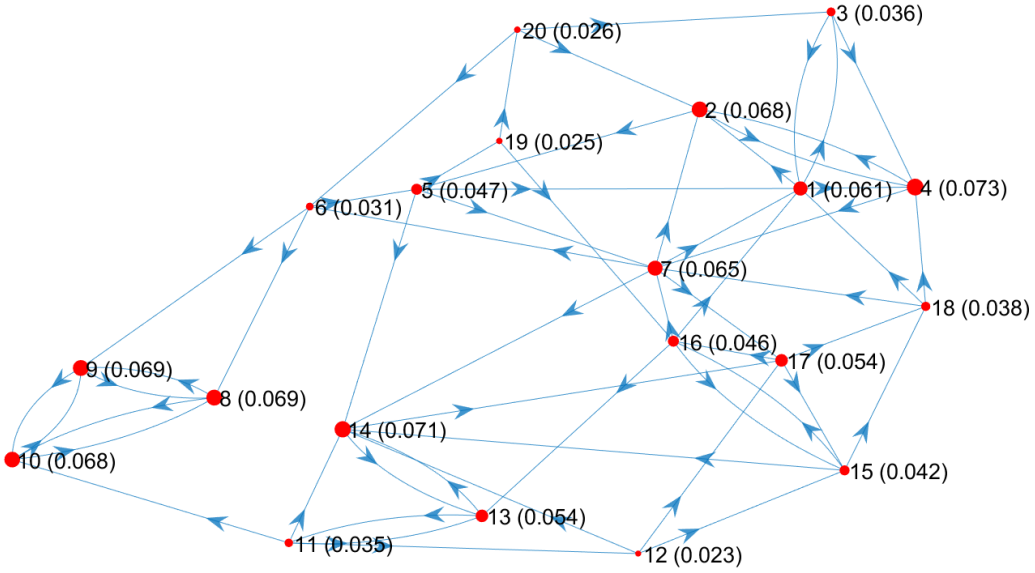

Figure 5: In this graph, the network is the same as the previous figure. The rank sink still exists, but we calculated PageRank with d=0.3, showing that the direct jumps reduce the amount of PageRank captured by rank sinks.

References

Google (June 2015). *Our History in Detail – Google*. URL: `https://web.archive.org/web/20150623193037/https://www.google.com/about/company/history/` (visited on 11/15/2019).

Page, Lawrence et al. (Nov. 1999). *The PageRank Citation Ranking: Bringing Order to the Web*. Techreport. URL: `http://ilpubs.stanford.edu:8090/422/` (visited on 11/15/2019).

Wikipedia (Nov. 2019). "Perron–Frobenius Theorem". en. In: *Wikipedia*. Page Version ID: 925304670. URL: `https://en.wikipedia.org/w/index.php?title=Perron%5C%E2%5C%80%5C%93Frobenius_theorem%5C&oldid=925304670` (visited on 11/15/2019).