Ranking Web Pages

Tim Chartier and Anne Greenbaum

Department of Mathematics @

Winter 2008

Google

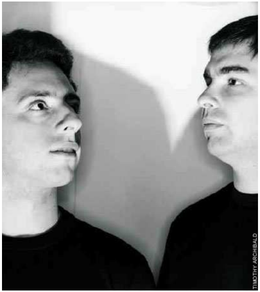
- Have a question? Looking for an old friend? Need a reference for a paper? A popular and often effective form of information acquisition is submitting queries to Google.com.
- In fact, in January 2003 just over 1,200 searches would have been conducted in the past second.
- “Google” is a play on the word “googol,” the number 10^{100}, reflecting the company’s goal of organizing all information on the World Wide Web.

Google-ing Math

- Suppose that we submit the query `mathematics` to Google.
- **Why** is the page we see at the top of the list deemed the “best” page related to the query?
- The web page listed first by Google is deemed, loosely speaking, the best web page related to the query.
- How is this page given such distinction?

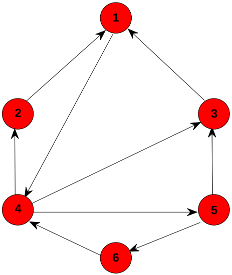
The PageRank algorithm

- Developed by Google's founders, Larry Page and Sergey Brin, who were graduate students at Stanford University when the foundational ideas of Google developed.
- Google ranks webpages according to the percentage of time one would end up at each web on a random walk through the web.



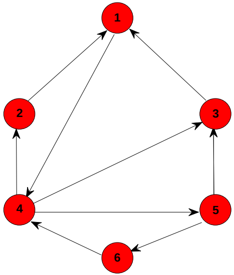
Return to Monte Carlo

Let's return to Monte Carlo simulation to mathematically model such a random walk through a web network.



Surf over mini–web

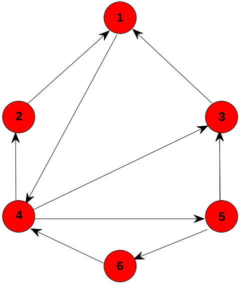
- Assume we start at web page 1.
- We will assume that at each stage the surfer will randomly follow one of the links on the page. The surfer can choose any link with equal probability.



Adjacency matrix

- We will represent the structure of a network with a matrix.
- The *adjacency matrix* for the network below is:

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



Your Turn

- From the course webpage found at:
  http://www.math.washington.edu/~greenbau
  download `googleSim1.m`.
- Find the PageRank of the system.
- Experiment with altering networks and viewing the results.
- How many iterates do you need to distinguish the rankings of the various webpages?

What about Google?

- Google indexes billions of webpages.
- How is PageRank found by Google?

Getting Stochastic

- Form a stochastic matrix M from our adjacency matrix.
- That is, element m_{ij} gives the probability of a surfer visit webpage j from webpage i, which implies

$$m_{ij} = g_{ij} / \sum_j g_{ij}.$$

- Therefore for

$$G = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \quad M = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Sparse is Nice

- Note that G is sparse.
- Recall the size of n. The sparsity of G will be an asset in manipulating it on a computer.
- In particular, only the nonzero entries, along with column and row information, are stored for large sparse matrices.
- Because the average out-degree of pages on the web is about seven [Kleinberg et al. 1999], this saves a factor on the order of half a billion in storage space and since n is growing over time while the average number of links on each page appear to remain about constant, the savings will only increase over time.

And?

- Now, let's start our random walk at state 1.
- What is the probability that we land at web page i after one step?
- While trivial to compute, we can also find this with our transition matrix.
- First, we represent our initial state by the vector

$$\mathbf{v} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- Simply compute $\mathbf{v}M = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \end{pmatrix}$.

Two step

- Now, let's take another step.
- Compute $\mathbf{v}_2 = \mathbf{v}_1 M = \begin{pmatrix} 0 & 0.33 & 0.33 & 0 & 0.33 & 0 \end{pmatrix}$
- **Your Turn**

  Find $\mathbf{v}_3$.

Two step

- Now, let's take another step.
- Compute $\mathbf{v}_2 = \mathbf{v}_1 M = \begin{pmatrix} 0 & 0.33 & 0.33 & 0 & 0.33 & 0 \end{pmatrix}$
- **Your Turn**

  Find $\mathbf{v}_3$.

  <u>Answer</u> $\mathbf{v}_3 = \mathbf{v}_2 M = \begin{pmatrix} 0.67 & 0 & 0.17 & 0 & 0 & 0.17 \end{pmatrix}$

Lotsa steps

- An important observation should be made about the matrix-vector multiplication. In particular,

$$
\begin{aligned}
\mathbf{v}_4 &= \mathbf{v}_3 M \\
&= (\mathbf{v}_2 M) M \\
&= \mathbf{v}_2 M^2 \\
&= (\mathbf{v}_1 M) M^2 \\
&= \mathbf{v}_1 M^3 \\
&= (\mathbf{v} M) M^3 \\
&= \mathbf{v} M^4.
\end{aligned}
$$

Lotsa steps

- An important observation should be made about the matrix-vector multiplication. In particular,

$$
\begin{aligned}
\mathbf{v}_4 &= \mathbf{v}_3 M \\
&= (\mathbf{v}_2 M)M \\
&= \mathbf{v}_2 M^2 \\
&= (\mathbf{v}_1 M)M^2 \\
&= \mathbf{v}_1 M^3 \\
&= (\mathbf{v}M)M^3 \\
&= \mathbf{v}M^4.
\end{aligned}
$$

- Therefore, we can easily find say $\mathbf{v}_{100}$. Compute $\mathbf{v}M^{100} = \begin{pmatrix} 0.263 & 0.105 & 0.158 & 0.316 & 0.105 & 0.053 \end{pmatrix}$

Marathon of steps

- **Your Turn** Find $\mathbf{v}_{500}$.
- **Your Turn** Find $\mathbf{v}_{700}$.
- What do you notice?

Marathon of steps

- **Your Turn** Find $\mathbf{v}_{500}$.
- **Your Turn** Find $\mathbf{v}_{700}$.
- What do you notice?
- To three decimal places,

$$\begin{aligned} \mathbf{v}M^{700} &= \mathbf{v}M^{500} = \mathbf{v}M^{100} \\ &= \begin{pmatrix} 0.263 & 0.105 & 0.158 & 0.316 & 0.105 & 0.053 \end{pmatrix} \end{aligned}$$

Google's Eigenvectors

- A non-negative vector that satisfies $\mathbf{v}M = \mathbf{v}$ is called a *steady-state vector* of the Markov process (where $\mathbf{v}$ is normalized such that $\sum \mathbf{v}_i = 1$, which results in a vector of probabilities).
- For us, it is important to note that this is a left-eigenvector of the matrix M. That is, $\mathbf{v}M = \mathbf{v}$.
- Did you notice that we were just using the Power Method to find $\mathbf{v}$?
- However, notice that we didn't need, at least in that example, to normalize the vector at each step.

PageRank is a vector!

- Google defines the PageRank of page i to be $\mathbf{v}_i$.
- Therefore, the largest element of $\mathbf{v}$ corresponds to the page with the highest PageRank, the second largest to the page with the second highest PageRank, and so on.
- The limiting frequency that an infinitely dedicated random surfer visits any particular page is that page's PageRank.

MATLAB's search

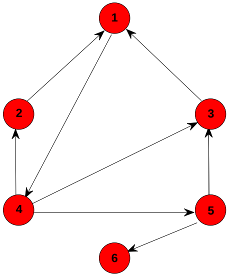
- The following will implement the Power Method to find the PageRank vector.

```
iterates = 0;
while max(abs(vNew-v)) > .001
   v = vNew;
   vNew = v*M;
   iterates = iterates + 1;
end
```

- The full code can be found also on the course webpage as `googlePower.m`.

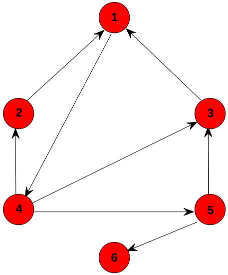
Catching another wave

- Let's surf again.
- Adapt `googlePower.m` for the network below.



Dangling node

- Note, web page 6 is what is called a *dangling node* with no outlinks. What web pages have this behavior?
- What problem did you see with our current model?
- Ideas to fix it? Let's see if we can come up with the one of Brin and Page that lies deep within Google's algorithm.



Revised random surfing

The rules to our Monte Carlo “game” are now:

- Again, restrict ourselves only to *indexed* web pages.
- Assume that for $p = 0.85$ or 85% of the time a surfer follows a link that is available on the current web page that the surfer is visiting. The other 15% of the time the surfer randomly visits (with equal probability) any web page available in the network.

It's element-ary

- Let r_i denote the row sum of row i.
- Therefore, the transition matrix M has elements

$$m_{ij} = \begin{cases} p\left(\frac{g_{ij}}{r_i}\right) + \frac{1-p}{n}, & r_i \neq \mathbf{0} \\ \frac{1}{n}, & r_i = \mathbf{0}. \end{cases}$$

- Again, $p = 0.85$.
- This model creates a transition matrix that is often called the *Google matrix*.

Forming the Transition Matrix

- Therefore for our problem the adjacency matrix is:

$$G = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

- The first row of the transition matrix M is

$$\left(.15/6 \quad .15/6 \quad .15/6 \quad .85 + .15/6 \quad .15/6 \quad .15/6\right).$$

Forming the Transition Matrix, cont.

- Again,

$$G = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

- Therefore, the Google matrix is

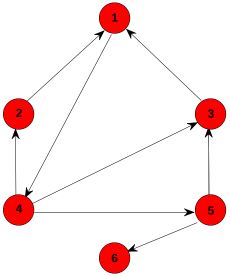
$$M = \begin{pmatrix} 0.0250 & 0.0250 & 0.0250 & 0.8750 & 0.0250 & 0.0250 \\ 0.8750 & 0.0250 & 0.0250 & 0.0250 & 0.0250 & 0.0250 \\ 0.8750 & 0.0250 & 0.0250 & 0.0250 & 0.0250 & 0.0250 \\ 0.0250 & 0.3083 & 0.3083 & 0.0250 & 0.3083 & 0.0250 \\ 0.0250 & 0.0250 & 0.4500 & 0.0250 & 0.0250 & 0.4500 \\ 0.1667 & 0.1667 & 0.1667 & 0.1667 & 0.1667 & 0.1667 \end{pmatrix}$$

Rankings

- If you enter M into `googlePower.m` the algorithm will converge to:

$$\begin{pmatrix} 0.2680 & 0.1117 & 0.1594 & 0.2644 & 0.1117 & 0.0846 \end{pmatrix}$$

- Therefore, page 1 has the best ranking followed by page 4. Compare this to the network.



Existence and uniqueness

- If this vector is not unique, which one would you choose? Would you bid between companies for which one to choose?
- The following theorem [Lax 1997] guarantees the uniqueness of the steady-state vector and that it will have positive entries:

> **Theorem** (Perron) Every real square matrix P who entries are all positive has a unique eigenvector with all positive entries, its corresponding eigenvalue has multiplicity one, and it is the dominant eigenvalue, in that every other eigenvalue has strictly smaller magnitude.

Stochastic matrices

- Recall that the rows of M sum to 1. Therefore, $M\mathbf{1} = \mathbf{1}$, where $\mathbf{1}$ is the column vector of all ones. That is, $\mathbf{1}$ is a right eigenvector of M associated with the eigenvalue 1, most notably for our purposes having all positive entries.
- Perron's Theorem ensures that $\mathbf{1}$ is the unique right eigenvector with all positive entries, and hence its eigenvalue must be the dominant one.
- The right and left eigenvalues of a matrix are the same, therefore 1 is the dominant left eigenvalue as well. So, there exists a unique steady-state vector $\mathbf{v}$ that satisfies $\mathbf{v}M = \mathbf{v}$. Normalizing this eigenvector so that $\sum \mathbf{v}_i = 1$ gives a steady-state vector.

Your Turn!

- It is time to experiment and play. Indeed, we will become Google search engines (simple ones) ourselves.
- To search from a homepage, you will type a statement like: `[U,G] = surfer('http://www.xxx.zzz',n)`.
- This starts at the given URL and tries to surf the Web until it has visited n pages. That is, an n by n matrix is formed.
- Note, surfing can cause problems and as such you may even have to terminate MATLAB. Yet, nonetheless we can create our own PageRank example.

Google-time!

- Download `surfer.m`, `pagerank.m` and `pagerankpow.m` from the course webpage. Note, this version of the Power Method only uses G and does not use the transition matrix M.
- These codes were written by Cleve Moler although `pagerank.m` is an adapted version of Cleve's codes.
- Let's begin with www.washington.edu as our starting URL. Let's only visit 20 pages. Therefore type:

```
[U,G] = surfer('http://www.washington.edu',20);
```

  - U = a cell array of n strings, the URLs of the nodes.
  - G = an n-by-n sparse matrix with $G(i,j) = 1$ if node j is linked to node i.
- Type: `pagerank` and the pagerank will be computed.
- **Note**: The program hangs sometimes and requires breaking from the program (CTRL-C) or shutting down MATLAB altogether.