Webgraph structure and PageRank
Two More Datasets Available

- TheFind.com
  - Large set of products (~6GB compressed)
  - For each product
    - Attributes
    - Related products

- Craigslist
  - About 3 weeks of data (~7.5GB compressed)
    - Text of posts, plus category metadata
    - e.g., match buyers and sellers
How big is the Web?

- Technically, infinite
- Much duplication (30-40%)
- Best estimate of “unique” static HTML pages comes from search engine claims
  - Google = 8 billion(?), Yahoo = 20 billion

What is the structure of the Web? How is it organized?
Web as a Graph

I teach a class on Networks.

Networks Course: We have a class blog

Networks Class Blog: This blog post is about Microsoft

Microsoft Home Page
In early days of the Web links were navigational.
Today many links are transactional.
Directed graphs

- Two types of directed graphs:
  - DAG – directed acyclic graph:
    - Has no cycles: if u can reach v, then v can not reach u
  - Strongly connected:
    - Any node can reach any node via a directed path

- Any directed graph can be expressed in terms of these two types
Strongly connected component (SCC) is a set of nodes $S$ so that:

- Every pair of nodes in $S$ can reach each other
- There is no larger set containing $S$ with this property
Graph structure of the Web

- Take a large snapshot of the web and try to understand how it’s SCCs “fit” as a DAG.

- **Computational issues:**
  - Say want to find SCC containing specific node v?
  - Observation:
    - Out(v) ... nodes that can be reachable from v (BFS out)
    - SCC containing v:
      - = Out(v, G) ∩ In(v, G)
      - = Out(v, G) ∩ Out(v, \( \bar{G} \))
    where \( \bar{G} \) is G with directions of all edge flipped
Graph structure of the Web

- There is a giant SCC
- Broder et al., 2000:
  - Giant weakly connected component: 90% of the nodes
250 million webpages, 1.5 billion links [Altavista]
Diameter of the Web

- Diameter (average directed shortest path length) is 19 (in 1999)
Diameter of the Web

- Average distance:
  75% of time there is no directed path from start to finish page
  - Follow in-links (directed): 16.12
  - Follow out-links (directed): 16.18
  - Undirected: 6.83

- Diameter of SCC (directed):
  - At least 28
Degree distribution on the Web

[Broder et al., '00]
Degrees in real networks

- Take real network plot a histogram of $p_k$ vs. $k$
- Plot the same data on log-log axis:

\[ p_k = \beta k^{-\alpha} \]

\[ \log p_k = \log \beta - \alpha \log k \]
Exponential tail vs. Power-law tail

Exponential: $Y \sim e^{-X}$

Power law: $Y \sim X^{-2}$
Power law degree exponents

- Power law degree exponent is typically $2 < \alpha < 3$
  - Web graph [Broder et al. 00]:
    - $\alpha_{in} = 2.1$, $\alpha_{out} = 2.4$
  - Autonomous systems [Faloutsos et al. 99]:
    - $\alpha = 2.4$
  - Actor collaborations [Barabasi-Albert 00]:
    - $\alpha = 2.3$
  - Citations to papers [Redner 98]:
    - $\alpha \approx 3$
  - Online social networks [Leskovec et al. 07]:
    - $\alpha \approx 2$
Power-law network

Random network
(Erdos-Renyi random graph)

Degree distribution is Binomial

Scale-free (power-law) network

Degree distribution is Power-law

Function is scale free if:
\[ f(ax) = c f(x) \]
Web pages are not equally “important”
  - www.joe-schmoe.com vs www.stanford.edu

Since there is big diversity in the connectivity of the webgraph we can rank pages by the link structure
Links as votes

- First try:
  - Page is more important if it has more links
    - In-coming links? Out-going links?
- Think of in-links as votes:
  - [www.stanford.edu](http://www.stanford.edu) has 23,400 inlinks
  - [www.joe-schmoe.com](http://www.joe-schmoe.com) has 1 inlink
- Are all in-links are equal?
  - Links from important pages count more
  - Recursive question!
Simple recursive formulation

- Each link’s vote is proportional to the importance of its source page
- If page $P$ with importance $x$ has $n$ out-links, each link gets $x/n$ votes
- Page $P$’s own importance is the sum of the votes on its in-links
The web in 1839

\[ y = y/2 + a/2 \]
\[ a = y/2 + m \]
\[ m = a/2 \]
Solving the flow equations

- 3 equations, 3 unknowns, no constants
  - No unique solution
  - All solutions equivalent modulo scale factor
- Additional constraint forces uniqueness
  - y+a+m = 1
  - y = 2/5, a = 2/5, m = 1/5
- Gaussian elimination method works for small examples, but we need a better method for large web-size graphs
Matrix formulation

- Matrix $M$ has one row and one column for each web page
- Suppose page $j$ has $n$ out-links
  - If $j \rightarrow i$, then $M_{ij} = 1/n$
  - else $M_{ij} = 0$
- $M$ is a column stochastic matrix
  - Columns sum to 1
- Suppose $r$ is a vector with one entry per web page
  - $r_i$ is the importance score of page $i$
  - Call it the rank vector
  - $|r| = 1$
Suppose page $j$ links to 3 pages, including $i$.
The flow equations can be written as:

\[ r = Mr \]

So the rank vector is an eigenvector of the stochastic web matrix.

- In fact, its first or principal eigenvector, with corresponding eigenvalue 1.
Example

\[ y = y/2 + a/2 \]
\[ a = y/2 + m \]
\[ m = a/2 \]

\[
\begin{array}{ccc}
Y! & A & MS \\
Y! & \frac{1}{2} & \frac{1}{2} & 0 \\
A & \frac{1}{2} & 0 & 1 \\
MS & 0 & \frac{1}{2} & 0 \\
\end{array}
\]

\[ r = Mr \]
Power Iteration method

- Simple iterative scheme (aka relaxation)
- Suppose there are N web pages
- Initialize: $r^0 = [1/N,\ldots,1/N]^T$
- Iterate: $r^{k+1} = Mr^k$
- Stop when $|r^{k+1} - r^k|_1 < \varepsilon$
  - $|x|_1 = \sum_{1 \leq i \leq N} |x_i|$ is the $L_1$ norm
  - Can use any other vector norm e.g., Euclidean
Power Iteration Example

- Power iteration:
  - Set $r_i = 1/n$
  - $r_i = \sum_j M_{ij} r_j$
  - And iterate

- Example:

  \[
  \begin{array}{cccccc}
  y & 1/3 & 1/3 & 5/12 & 3/8 & 2/5 \\
  a = & 1/3 & 1/2 & 1/3 & 11/24 & \ldots & 2/5 \\
  m & 1/3 & 1/6 & 1/4 & 1/6 & 1/5 \\
  \end{array}
  \]
Random Walk Interpretation

- Imagine a random web surfer
  - At any time $t$, surfer is on some page $P$
  - At time $t+1$, the surfer follows an outlink from $P$ uniformly at random
  - Ends up on some page $Q$ linked from $P$
  - Process repeats indefinitely
- Let $\mathbf{p}(t)$ be a vector whose $i^{th}$ component is the probability that the surfer is at page $i$ at time $t$
  - $\mathbf{p}(t)$ is a probability distribution on pages
The stationary distribution

- Where is the surfer at time $t+1$?
  - Follows a link uniformly at random
  - $p(t+1) = Mp(t)$
- Suppose the random walk reaches a state such that $p(t+1) = Mp(t) = p(t)$
  - Then $p(t)$ is called a stationary distribution for the random walk
- Our rank vector $r$ satisfies $r = Mr$
  - So it is a stationary distribution for the random surfer
Existence and Uniqueness

A central result from the theory of random walks (aka Markov processes):

For graphs that satisfy certain conditions, the stationary distribution is unique and eventually will be reached no matter what the initial probability distribution at time $t = 0$. 
Some pages are “dead ends” (have no out-links)
  - Such pages cause importance to leak out

Spider traps (all out links are within the group)
  - Eventually spider traps absorb all importance
Spider traps

- A group of pages is a spider trap if there are no links from within the group to outside the group
  - Random surfer gets trapped
- Spider traps violate the conditions needed for the random walk theorem
Spider traps

- **Power iteration:**
  - Set $r_i=1$
  - $r_i = \sum_j M_{ij} r_j$
  - And iterate

- **Example:**

<table>
<thead>
<tr>
<th></th>
<th>Y!</th>
<th>A</th>
<th>MS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y!</td>
<td>1/2</td>
<td>1/2</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>1/2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MS</td>
<td>0</td>
<td>1/2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>y</th>
<th>1</th>
<th>1</th>
<th>3/4</th>
<th>5/8</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>1/2</td>
<td>1/2</td>
<td>3/8</td>
<td>...</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
<td>3/2</td>
<td>7/4</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Solution: Random teleports

- The Google solution for spider traps
- At each time step, the random surfer has two options:
  - With probability $\beta$, follow a link at random
  - With probability $1-\beta$, jump to some page uniformly at random
- Common values for $\beta$ are in the range 0.8 to 0.9
- Surfer will teleport out of spider trap within a few time steps
Random teleports ($\beta = 0.8$)

$$
\begin{align*}
\begin{array}{ccc}
\text{Yahoo} & \text{Amazon} & \text{M'soft} \\
0.2 \times \frac{1}{3} & 0.8 \times \frac{1}{2} & 0.2 \times \frac{1}{3} \\
\frac{1}{2} & 0.8 \times \frac{1}{2} & 0.2 \times \frac{1}{3} \\
0.8 \times \frac{1}{2} & 0.2 \times \frac{1}{3} & 0.2 \times \frac{1}{3}
\end{array}
\end{align*}
$$

$$
\begin{align*}
\begin{array}{ccc}
y & a & m \\
1/2 & 1/2 & 0 \\
1/2 & 1/2 & 0 \\
0 & 1/2 & 1
\end{array}
\end{align*}
$$

$$
\begin{align*}
\begin{array}{ccc}
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3
\end{array}
\end{align*}
$$

$$
\begin{align*}
\begin{array}{ccc}
y & a & m \\
7/15 & 7/15 & 1/15 \\
7/15 & 1/15 & 1/15 \\
1/15 & 7/15 & 13/15
\end{array}
\end{align*}
$$
Random teleports ($\beta = 0.8$)

\[
\begin{pmatrix}
0.8 & 1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 & + 0.2 \\
1/3 & 1/3 & 1/3 & 1/3 \\
\end{pmatrix}
\]

\[
\begin{align*}
y & 7/15 & 7/15 & 1/15 \\
a & 7/15 & 1/15 & 1/15 \\
m & 1/15 & 7/15 & 13/15 \\
\end{align*}
\]

\[
\begin{align*}
y & 1 & 1.00 & 0.84 & 0.776 & 7/11 \\
a & = & 1 & 0.60 & 0.60 & 0.536 & \ldots & 5/11 \\
m & 1 & 1.40 & 1.56 & 1.688 & 21/11 \\
\end{align*}
\]
Dead ends

- Power iteration:
  - Set $r_i=1$
  - $r_i = \sum_j M_{ij} r_j$
  - And iterate

- Example:

<table>
<thead>
<tr>
<th></th>
<th>Y!</th>
<th>A</th>
<th>MS</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>1/2</td>
<td>1/2</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1/2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>1/2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>y</th>
<th>1</th>
<th>1</th>
<th>3/4</th>
<th>5/8</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>1/2</td>
<td>1/2</td>
<td>3/8</td>
<td>...</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
<td>1/2</td>
<td>1/4</td>
<td>1/4</td>
<td>0</td>
</tr>
</tbody>
</table>
Dealing with dead-ends

- Teleport
  - Follow random teleport links with probability 1.0 from dead-ends
  - Adjust matrix accordingly
- Prune and propagate
  - Preprocess the graph to eliminate dead-ends
  - Might require multiple passes
  - Compute page rank on reduced graph
  - Approximate values for deadends by propagating values from reduced graph
Suppose there are N pages

- Consider a page j, with set of outlinks O(j)
- We have $M_{ij} = 1/|O(j)|$ when $j \rightarrow i$ and $M_{ij} = 0$ otherwise

The random teleport is equivalent to

- adding a teleport link from j to every other page with probability $(1-\beta)/N$
- reducing the probability of following each outlink from $1/|O(j)|$ to $\beta/|O(j)|$
- Equivalent: tax each page a fraction $(1-\beta)$ of its score and redistribute evenly
Page Rank

- Construct the N x N matrix $A$ as follows
  - $A_{ij} = \beta M_{ij} + (1-\beta)/N$
- Verify that $A$ is a stochastic matrix
- The page rank vector $r$ is the principal eigenvector of this matrix
  - satisfying $r = Ar$
- Equivalently, $r$ is the stationary distribution of the random walk with teleports
Key step is matrix-vector multiplication
- $r_{\text{new}} = Ar_{\text{old}}$

Easy if we have enough main memory to hold $A$, $r_{\text{old}}$, $r_{\text{new}}$

Say $N = 1$ billion pages
- We need 4 bytes for each entry (say)
- 2 billion entries for vectors, approx 8GB
- Matrix $A$ has $N^2$ entries
  - $10^{18}$ is a large number!
r = Ar, where

\[ A_{ij} = \beta M_{ij} + (1 - \beta)/N \]

\[ r_i = \sum_{1 \leq j \leq N} A_{ij} r_j \]

\[ r_i = \sum_{1 \leq j \leq N} \left[ \beta M_{ij} + (1 - \beta)/N \right] r_j \]

\[ = \beta \sum_{1 \leq j \leq N} M_{ij} r_j + (1 - \beta)/N \sum_{1 \leq j \leq N} r_j \]

\[ = \beta \sum_{1 \leq j \leq N} M_{ij} r_j + (1 - \beta)/N, \text{ since } |r| = 1 \]

\[ r = \beta Mr + [(1 - \beta)/N]_N \]

where \([x]_N\) is an N-vector with all entries \(x\)
We can rearrange the page rank equation:

- \( r = \beta M \cdot r + \left(\frac{1-\beta}{N}\right)_N \)
- \( \left(\frac{1-\beta}{N}\right)_N \) is an N-vector with all entries \((1-\beta)/N\)
- \( M \) is a sparse matrix!
  - 10 links per node, approx 10N entries
- So in each iteration, we need to:
  - Compute \( r^{\text{new}} = \beta M \cdot r^{\text{old}} \)
  - Add a constant value \((1-\beta)/N\) to each entry in \( r^{\text{new}} \)
Sparse matrix encoding

- Encode sparse matrix using only nonzero entries
  - Space proportional roughly to number of links
  - say 10N, or 4*10*1 billion = 40GB
  - still won’t fit in memory, but will fit on disk

<table>
<thead>
<tr>
<th>source node</th>
<th>degree</th>
<th>destination nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>1, 5, 7</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>17, 64, 113, 117, 245</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>13, 23</td>
</tr>
</tbody>
</table>
Basic Algorithm

- Assume we have enough RAM to fit $r^{\text{new}}$, plus some working memory
  - Store $r^{\text{old}}$ and matrix $M$ on disk

Basic Algorithm:
- **Initialize:** $r^{\text{old}} = [1/N]_N$
- **Iterate:**
  - **Update:** Perform a sequential scan of $M$ and $r^{\text{old}}$ to update $r^{\text{new}}$
  - Write out $r^{\text{new}}$ to disk as $r^{\text{old}}$ for next iteration
  - Every few iterations, compute $|r^{\text{new}} - r^{\text{old}}|$ and stop if it is below threshold
    - Need to read in both vectors into memory
Initialize all entries of $r^{new}$ to $(1-\beta)/N$

For each page $p$ (out-degree $n$):
Read into memory: $p$, $n$, $dest_1, \ldots, dest_n$, $r^{old}(p)$
for $j = 1..n$:
\[ r^{new}(dest_j) += \beta r^{old}(p)/n \]
In each iteration, we have to:

- Read \( r^{\text{old}} \) and \( M \)
- Write \( r^{\text{new}} \) back to disk
- IO Cost = \( 2|r| + |M| \)

What if we had enough memory to fit both \( r^{\text{new}} \) and \( r^{\text{old}} \)?

What if we could not even fit \( r^{\text{new}} \) in memory?

- 10 billion pages
Block-based update algorithm

<table>
<thead>
<tr>
<th>r\text{new}</th>
<th>src</th>
<th>degree</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0, 1, 3, 5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0, 5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3, 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r\text{old}</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Analysis of Block Update

- Similar to nested-loop join in databases
  - Break $r^{new}$ into $k$ blocks that fit in memory
  - Scan $M$ and $r^{old}$ once for each block
- $k$ scans of $M$ and $r^{old}$
  - $k(|M| + |r|) + |r| = k|M| + (k+1)|r|$
- Can we do better?
- Hint: $M$ is much bigger than $r$ (approx 10-20x), so we must avoid reading it $k$ times per iteration
### Block-Stripe Update Algorithm

The table shows the relationships between source (src), degree, and destination.

<table>
<thead>
<tr>
<th>src</th>
<th>degree</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0, 1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

The notation `r_{new}` and `r_{old}` indicate the new and old rank values, respectively.
Block-Stripe Analysis

- Break $\mathbf{M}$ into stripes
  - Each stripe contains only destination nodes in the corresponding block of $\mathbf{r}^{\text{new}}$
- Some additional overhead per stripe
  - But usually worth it
- Cost per iteration
  - $|\mathbf{M}|(1+\varepsilon) + (k+1)|\mathbf{r}|$