CS345
Data Mining

Link Analysis Algorithms
Page Rank

Anand Rajaraman, Jeffrey D. Ullman
Link Analysis Algorithms

- Page Rank
- Hubs and Authorities
- Topic-Specific Page Rank
- Spam Detection Algorithms
- Other interesting topics we won’t cover
  - Detecting duplicates and mirrors
  - Mining for communities
Ranking web pages

- Web pages are not equally “important”
  - www.joe-schmoe.com vs www.stanford.edu

- Inlinks as votes
  - www.stanford.edu has 23,400 inlinks
  - www.joe-schmoe.com has 1 inlink

- Are all inlinks equal?
  - 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 $n$ outlinks, each link gets $x/n$ votes.
- Page $P$’s own importance is the sum of the votes on its inlinks.
Simple “flow” model

The web in 1839

\[
y = \frac{y}{2} + \frac{a}{2}\\
a = \frac{y}{2} + m\\
m = \frac{a}{2}
\]
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 graphs
Matrix formulation

- Matrix $M$ has one row and one column for each web page.
- Suppose page $j$ has $n$ outlinks:
  - If $j \neq i$, then $M_{ij} = \frac{1}{n}$
  - 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
  - $|r| = 1$
Example

Suppose page \( j \) links to 3 pages, including \( i \)

\[
M \quad r \quad = \quad r
\]

\[
\begin{array}{c}
i \\
j \\
i
\end{array}
\]

\[
\text{1/3}
\]
Eigenvector formulation

☐ The flow equations can be written

\[ r = Mr \]

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

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

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

\[
\begin{bmatrix}
y \\ a \\ m
\end{bmatrix}
= \begin{bmatrix}
1/2 & 1/2 & 0 \\
1/2 & 0 & 1 \\
0 & 1/2 & 0
\end{bmatrix}
\begin{bmatrix}
y \\ a \\ m
\end{bmatrix}
\]

\[ r = Mr \]
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

\[
\begin{align*}
\text{Yahoo} & \quad \text{Amazon} & \quad \text{M’soft} \\
\end{align*}
\]

\[
\begin{pmatrix}
y \\ a \\ m
\end{pmatrix}
= \begin{pmatrix}
1/2 & 1/2 & 0 \\
1/2 & 0 & 1 \\
0 & 1/2 & 0
\end{pmatrix}
\]

\[
\begin{pmatrix}
y \\ a \\ m
\end{pmatrix}
= \begin{pmatrix}
1/3 & 1/3 & 5/12 & 3/8 & 2/5 \\
1/3 & 1/2 & 1/3 & 11/24 & \ldots & 2/5 \\
1/3 & 1/6 & 1/4 & 1/6 & 1/5
\end{pmatrix}
\]
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 $p(t)$ be a vector whose $i^{th}$ component is the probability that the surfer is at page $i$ at time $t$
  - $p(t)$ 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 $t = 0$. 
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
Microsoft becomes a spider trap

\[
\begin{array}{c|cccc}
  & y & a & m \\
\hline
 y & 1/2 & 1/2 & 0 \\
a & 1/2 & 0 & 0 \\
m & 0 & 1/2 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
 y & 1 & 1 & 3/4 & 5/8 & 0 \\
a & 1 & 1/2 & 1/2 & 3/8 & \ldots & 0 \\
m & 1 & 3/2 & 7/4 & 2 & 3
\end{array}
\]
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 ($\beta = 0.8$)
Random teleports \((\beta = 0.8)\)

\[
\begin{array}{cccc}
1/2 & 1/2 & 0 \\
1/2 & 0 & 0 \\
0 & 1/2 & 1 \\
\end{array}
\]+\[0.2
\begin{array}{ccc}
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
\end{array}
\]

<table>
<thead>
<tr>
<th></th>
<th>7/15</th>
<th>7/15</th>
<th>1/15</th>
</tr>
</thead>
</table>
y | 7/15 | 1/15 | 1/15 |
a | 1/15 | 7/15 | 13/15 |
m | 1/15 | 7/15 | 13/15 |

\[
y = m
\]

\[
a = \begin{array}{ccc}
1 & 0.60 & 0.60 \\
1 & 1.40 & 1.56 \\
1 & 1.40 & 1.56 \\
\end{array}
\]

\[
ym = \begin{array}{ccc}
1 & 0.60 & 0.536 \\
1 & 1.40 & 1.688 \\
1 & 1.40 & 1.688 \\
\end{array}
\]

\[
\text{...}
\]

\[
\text{...}
\]

\[
\text{...}
\]
Matrix formulation

- Suppose there are $N$ pages
  - Consider a page $j$, with set of outlinks $O(j)$
  - We have $M_{ij} = 1/|O(j)|$ when $j \neq i$ 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 $N \times N$ 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
Dead ends

- Pages with no outlinks are “dead ends” for the random surfer
  - Nowhere to go on next step
Microsoft becomes a dead end

Non-stochastic!
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
Computing page rank

- Key step is matrix-vector multiplication
  \[ r^{\text{new}} = A r^{\text{old}} \]

- 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!
Rearranging the equation

\[ \mathbf{r} = \mathbf{Ar}, \text{ where} \]

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

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

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

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

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

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

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

\[ r = \beta M r + \left( \frac{(1-\beta)}{N} \right)_N \]

\( \left( \frac{(1-\beta)}{N} \right)_N \) 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 \( r^{new} = \beta M r^{old} \)
- Add a constant value \((1-\beta)/N\) to each entry in \( r^{new} \)
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
**Update step**

Initialize all entries of $r^{\text{new}}$ to $(1-\beta)/N$

For each page $p$ (out-degree $n$):

- Read into memory: $p$, $n$, dest$_1$,\ldots,dest$_n$, $r^{\text{old}}(p)$
- for $j = 1..n$:
  
  $$r^{\text{new}}(\text{dest}_j) += \beta*r^{\text{old}}(p)/n$$

<table>
<thead>
<tr>
<th>src</th>
<th>degree</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>1, 5, 6</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>17, 64, 113, 117</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>13, 23</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Analysis

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

Diagram:

```
  r^new
  0
  1
  2
  3
  4
  5

  r^old
  0
  1
  2
  3
  4
  5
```
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

<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>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>src (old)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>src (new)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
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
  - $|\mathbf{M}|(1+\varepsilon) + (k+1)|\mathbf{r}|$
Next

- Topic-Specific Page Rank
- Hubs and Authorities
- Spam Detection