Link Analysis

CSE 454 Advanced Internet Systems
University of Washington
Ranking Search Results

• TF / IDF or BM25
• Tag Information
  – Title, headers
• Font Size / Capitalization
• Anchor Text on Other Pages
• Classifier Predictions
  – Spam, Adult, Review, Celebrity, …

• Link Analysis
  – HITS – (Hubs and Authorities)
  – PageRank
Pagerank Intuition

Think of Web as a big graph.

Suppose surfer keeps randomly clicking on the links. Importance of a page = probability of being on the page

Derive transition matrix from adjacency matrix

Suppose ∃ N forward links from page P
Then the probability that surfer clicks on any one is 1/N
Matrix Representation

Let $M$ be an $N \times N$ matrix

$$m_{uv} = \frac{1}{N_v} \text{ if page } v \text{ has a link to page } u$$

$$m_{uv} = 0 \quad \text{if there is no link from } v \text{ to } u$$

Let $R_0$ be the initial rank vector

Let $R_i$ be the $N \times 1$ rank vector for $i^{th}$ iteration

Then

$$R_{i+1} = M \times R_i$$
Problem: Page Sinks.

- Sink = node (or set of nodes) with no out-edges.
- Why is this a problem?
Solution to Sink Nodes

Let:

\((1-c)\) = chance of random transition from a sink.

\(N\) = the number of pages

\[ K = \begin{pmatrix}
\cdots & \cdots & 1/N & \cdots \\
\cdots & \cdots & \cdots & \cdots \\
\end{pmatrix} \]

\[ M^* = cM + (1-c)K \]

\[ R_{i+1} = M^* \times R_i \]
Computing PageRank - Example

\[ M = \begin{pmatrix}
A & B & C & D \\
A & 0 & 0 & 0 & \frac{1}{2} \\
B & 0 & 0 & 0 & \frac{1}{2} \\
C & 1 & 1 & 0 & 0 \\
D & 0 & 0 & 1 & 0 \\
\end{pmatrix} \]

\[ M^* = \begin{pmatrix}
0.05 & 0.05 & 0.05 & 0.45 \\
0.05 & 0.05 & 0.05 & 0.45 \\
0.85 & 0.85 & 0.05 & 0.05 \\
0.05 & 0.05 & 0.85 & 0.05 \\
\end{pmatrix} \]

\[ R_0 = \begin{pmatrix}
\frac{1}{4} \\
\frac{1}{4} \\
\frac{1}{4} \\
\frac{1}{4} \\
\end{pmatrix} \]

\[ R_{30} = \begin{pmatrix}
0.176 \\
0.176 \\
0.332 \\
0.316 \\
\end{pmatrix} \]
Ooops

• What About Sparsity?

\[ M^* = \begin{pmatrix}
0.05 & 0.05 & 0.05 & 0.45 \\
0.05 & 0.05 & 0.05 & 0.45 \\
0.85 & 0.85 & 0.05 & 0.05 \\
0.05 & 0.05 & 0.85 & 0.05 \\
\end{pmatrix} \]

\[ M^* = cM + (1-c)K \]

\[ K = \begin{pmatrix}
... \\
\cdots 1/N \cdots \\
... \\
\end{pmatrix} \]
Authority and Hub Pages (1)

• A page is a good authority
  (with respect to a given query)
  if it is pointed to by many good hubs
  (with respect to the query).

• A page is a good hub page
  (with respect to a given query)
  if it points to many good authorities
  (for the query).

• Good authorities & hubs reinforce
Authority and Hub Pages (2)

Authorities and hubs for a query *tend* to form a bipartite subgraph of the web graph.

(A page can be a good authority *and* a good hub)
Linear Algebraic Interpretation

• **PageRank = principle eigenvector of** $M^*$
  – in limit

• **HITS = principle eigenvector of** $M^* \times (M^*)^T$
  – Where $[ ]^T$ denotes transpose $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}^T = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$

• **Stability**
  Small changes to graph $\rightarrow$ small changes to weights.
  – Can prove PageRank is stable
  – And HITS isn’t
Stability Analysis (Empirical)

- **Make 5 subsets by deleting 30% randomly**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>12</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>52</td>
<td>20</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>171</td>
<td>119</td>
<td>99</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>135</td>
<td>56</td>
<td>40</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>179</td>
<td>159</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>316</td>
<td>141</td>
<td>170</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>257</td>
<td>107</td>
<td>72</td>
</tr>
<tr>
<td>10</td>
<td>13</td>
<td>170</td>
<td>80</td>
<td>69</td>
</tr>
</tbody>
</table>

- **PageRank much more stable**
Practicality

• **Challenges**
  - M no longer sparse (don’t represent explicitly!)
  - Data too big for memory (be sneaky about disk usage)

• **Stanford Version of Google**:
  - 24 million documents in crawl
  - 147GB documents
  - 259 million links
  - Computing pagerank “few hours” on single 1997 workstation

• **But How?**
  - Next discussion from Haveliwala paper…
Efficient Computation: Preprocess

- **Remove ‘dangling’ nodes**
  - Pages w/ no children

- **Then repeat process**
  - Since now more danglers

- **Stanford WebBase**
  - 25 M pages
  - 81 M URLs in the link graph
  - After two prune iterations: 19 M nodes
Representing ‘Links’ Table

- **Stored on disk in binary format**

<table>
<thead>
<tr>
<th>Source node (32 bit integer)</th>
<th>Outdegree (16 bit int)</th>
<th>Destination nodes (32 bit integers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>12, 26, 58, 94</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5, 56, 69</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1, 9, 10, 36, 78</td>
</tr>
</tbody>
</table>

- **Size for Stanford WebBase: 1.01 GB**
  - Assumed to exceed main memory
  - (But source & dest assumed to fit)
Algorithm 1

∀s Source[s] = 1/N

while residual > τ {  
  ∀d Dest[d] = 0
  while not Links.eof() {  
    Links.read(source, n, dest_1, … dest_n)  
    for j = 1… n  
      Dest[dest_j] = Dest[dest_j] + Source[source]/n  
  }
  ∀d Dest[d] = (1-c) * Dest[d] + c/N  /* dampening c = 1/N */
  residual = || Source – Dest ||  /* recompute every few iterations */
  Source = Dest
}

∀ dest node
dest links (sparse) × source node

∀ source node

Analysis

• If memory can hold both source & dest
  – IO cost per iteration is $|\text{Links}|$
  – Fine for a crawl of 24 M pages
  – But web > 8 B pages in 2005 [Google]
  – Increase from 320 M pages in 1997 [NEC study]

• If memory only big enough to hold just dest…?
  – Sort \textit{Links} on source field
  – Read \textit{Source} sequentially during rank propagation step
  – Write \textit{Dest} to disk to serve as \textit{Source} for next iteration
  – IO cost per iteration is $|\text{Source}| + |\text{Dest}| + |\text{Links}|$

• But What if memory can’t even hold dest?
  – Random access pattern will make working set $= |\text{Dest}|$
  – Thrash!!!

…..?????
Block-Based Algorithm

- **Partition** *Dest* into B blocks of D pages each
  - If memory = P physical pages
  - D < P-2 since need input buffers for Source & Links

- **Partition (sorted)** *Links* into B files
  - Linksᵢ only has *some* of the dest nodes for each source
    
    Specifically, Linksᵢ only has dest nodes such that
    
    - DDᵢ <= dest < DDᵢ₊₁
    
    - Where DD = number of 32 bit integers that fit in D pages
## Partitioned Link File

<table>
<thead>
<tr>
<th>Source node (32 bit int)</th>
<th>Outdegr (16 bit)</th>
<th>Num out (16 bit)</th>
<th>Destination nodes (32 bit integer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>2</td>
<td>12, 26</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>1, 9, 10</td>
</tr>
</tbody>
</table>

- **Buckets 0-31**

<table>
<thead>
<tr>
<th>Source node (32 bit int)</th>
<th>Outdegr (16 bit)</th>
<th>Num out (16 bit)</th>
<th>Destination nodes (32 bit integer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>1</td>
<td>58</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>56</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>36</td>
</tr>
</tbody>
</table>

- **Buckets 32-63**

<table>
<thead>
<tr>
<th>Source node (32 bit int)</th>
<th>Outdegr (16 bit)</th>
<th>Num out (16 bit)</th>
<th>Destination nodes (32 bit integer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>1</td>
<td>94</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>69</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>78</td>
</tr>
</tbody>
</table>

- **Buckets 64-95**
Analysis of Block Algorithm

• **IO Cost per iteration** =
  - $B^*|Source| + |Dest| + |Links|*(1+e)
  - $e$ is factor by which Links increased in size
    • Typically 0.1-0.3
    • Depends on number of blocks

• **Algorithm ~ nested-loops join**
Comparing the Algorithms
Comparing the Algorithms

![Graph comparing the algorithms for different physical memory sizes. The x-axis represents physical memory size (256 MB, 64 MB, 32 MB), and the y-axis represents minutes per iteration on a log scale. The graph shows performance comparisons between Naïve (1 Block), 2 Blocks, and 4 Blocks configurations.](image-url)
Adding PageRank to a Search Engine

• Weighted sum of importance + similarity with query
• Score(q, d)
  \[ \text{Score}(q, d) = w \times \text{sim}(q, p) + (1-w) \times \text{R}(p), \text{ if sim}(q, p) > 0 \]
  \[ = 0, \text{ otherwise} \]
• Where
  – \( 0 < w < 1 \)
  – sim(q, p), R(p) must be normalized to \([0, 1]\).
Summary of Key Points

• PageRank Iterative Algorithm
• Sink Pages
• Efficiency of computation – Memory!
  – Don’t represent M* explicitly.
  – Minimize IO Cost.
  – Break arrays into Blocks.
  – Single precision numbers ok.
• Number of iterations of PageRank.
• Weighting of PageRank vs. doc similarity.