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

Matrix Representation

Let \( M \) be an \( N \times N \) matrix

\[
M_{uv} = \frac{1}{N_v} \quad \text{if page } v \text{ has a link to page } u
\]

\[
M_{uv} = 0 \quad \text{if there is no link from } v \text{ to } u
\]

Let \( R_0 \) be the initial rank vector.

Let \( R_i \) be the \( N \times 1 \) rank vector for \( i \)th iteration.

Then \( R_i = M \times R_{i-1} \)

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

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

\[
K = \begin{bmatrix}
... & ... & ... & 1 / N & ... \\
... & ... & ... & ... & ...
\end{bmatrix}
\]

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

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

\[
M = \begin{pmatrix}
A & B & C & D \\
0 & 0 & 0 & 0.5 \\
0 & 0 & 0 & 0.5 \\
1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0
\end{pmatrix}
\]

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

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

\[
K = \begin{pmatrix}
\ldots & \ldots & \ldots & \ldots \\
\ldots & \frac{1}{N} & \ldots & \ldots
\end{pmatrix}
\]

Adding PageRank to a SearchEngine

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

Authority and Hub Pages

- 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 (cont)

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 \( M^* \cdot (M^*)^T \)
  – Where \([ J]^T\) denotes transpose

- 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

<table>
<thead>
<tr>
<th>Subset</th>
<th>Source</th>
<th>Outdegree</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>13</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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, dest1, … destn)
    for j = 1… n 
  }
  ∀d Dest[d] = (1-c) * Dest[d] + c/N
  residual = ||Source – Dest|| */recompute every few iterations */
  Source = Dest
}
Analysis

• If memory can hold both source & dest
  – IO cost per iteration is | 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 Links on source field
  – Read Source sequentially during rank propagation step
  – IO cost per iteration is | Source | + | Dest | + | Links |

• But What if memory can’t even hold dest?
  – Random access pattern will make working set = | 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
  – Links, only has some of the dest nodes for each source
    Specifically, Links, only has dest nodes such that
    • DD*i <= dest < DD*(i+1)
    • Where DD = number of 32 bit integers that fit in D pages

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

Partitioned Link File

<table>
<thead>
<tr>
<th>Source node (32 bit)</th>
<th>Outdeg (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>5</td>
<td>1, 9, 10</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Buckets 0-31

| 0                   | 4              | 1              | 58                                |
| 1                   | 3              | 5              | 56                                |
| 2                   | 5              | 3              | 36                                |

Buckets 32-63

| 0                   | 4              | 1              | 94                                |
| 1                   | 3              | 1              | 69                                |
| 2                   | 5              | 1              | 78                                |

Buckets 64-95

Comparing the Algorithms

Comparison of Block-Based Algorithm

• Physical Memory vs. I/O Rate
  – 256 MB
  – 64 MB
  – 32 MB

Comparison of Block-Based Algorithm

• Physical Memory vs. I/O Rate
  – 256 MB
  – 64 MB
  – 32 MB
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.