**Course Overview**

- Info Extraction
- Link Analysis
- Datamining
- P2P
- Security
- Ecommerce
- Web Services
- Semantic Web

**Systems Foundation: Networking & Clusters**

**Case Studies:** Nutch, Google, Altavista

**Datamining**

- Information Retrieval
- Precision vs Recall
- Inverted Indices

**Synchronization & Monitors**

**Crawler Architecture**

**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.

**Stability**

- Stability
  small changes to graph → small changes to weights.

- Conclusion
  HITS is not stable.
  But PageRank is quite stable!

  Details in a few slides

**Ranking Search Results**

- TF / IDF Calculation
- Tag Information
  – Title, headers
- Font Size / Capitalization
- Anchor Text on Other Pages
- 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 \( \exists N \) 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
\[
m_{uv} = \frac{1}{N} \quad \text{if page v has a link to page u}
m_{uv} = 0 \quad \text{if there is no link from v to u}
\]
Let \( R_0 \) be the initial rank vector
Let \( R_i \) be the \( N \times 1 \) rank vector for \( i \)th iteration
Then \[ R_i = M \times R_{i-1} \]

Problem: PageRank Sinks.
- Sinks = Sets of Nodes with no out-edges.
- Why is this a problem?

Solution to Sink Nodes
Let \((1-c)\) denote the chance of a random transition out of a sink.
\( N \) = the number of pages
\[
K = \begin{bmatrix}
    \cdots & 1/N & \cdots \\
    \cdots & 1/N & \cdots \\
    \cdots & 1/N & \cdots \\
\end{bmatrix}
\]
\[
M^* = cM + (1-c)K
\]
\[ R_i = M^* \times R_{i-1} \]

Computing PageRank - Example
\[
M = \begin{bmatrix}
    A & B & C & D \\
    0 & 0 & 0 & \frac{1}{2} \\
    0 & 0 & 0 & \frac{1}{2} \\
    1 & 1 & 0 & 0 \\
    0 & 0 & 1 & 0 \\
\end{bmatrix}
\]
\[
M^* = \begin{bmatrix}
    0.05 & 0.05 & 0.05 & 0.45 \\
    0.05 & 0.05 & 0.05 & 0.45 \\
    0.85 & 0.85 & 0.05 & 0.05 \\
    0.05 & 0.05 & 0.85 & 0.05 \\
\end{bmatrix}
\]
\[
R_0 = \begin{bmatrix}
    \frac{1}{4} \\
    \frac{1}{4} \\
    \frac{1}{4} \\
    \frac{1}{4} \\
\end{bmatrix}
\]
\[
R_{30} = \begin{bmatrix}
    0.176 \\
    0.176 \\
    0.332 \\
    0.316 \\
\end{bmatrix}
\]

Worked Example
Note: assume \( 1/N = 0.15 \)
Instead on \( R_0 \) = uniform \( 1/N \)
let \( R_0 = 1 \) everywhere
Navigational Effects (matrix M)

Random Jumps Give Extra 0.15
Page A: 0.85 (from Page C) + 0.15 (random) = 1
Page B: 0.425 (from Page A) + 0.15 (random) = 0.575
Page C: 0.85 (from Page D) + 0.85 (from Page B) + 0.425 (from Page A) + 0.15 (random) = 2.275
Page D: receives nothing but 0.15 (random) = 0.15

Round 2
Page A: 2.275*0.85 (from Page C) + 0.15 (random) = 2.08375
Page B: 1*0.85/2 (from Page A) + 0.15 (random) = 0.575
Page C: 0.15*0.85 (from D) + 0.575*0.85 (from B) + 1*0.85/2 (from Page A) + 0.15 (random) = 1.19125
Page D: receives nothing but random 0.15 = 0.15

Example of calculation (4)
After 20 iterations, we get
Page A: 1.490
Page B: 0.783
Page C: 1.577
Page D: 0.15

Example - Conclusions
• Page C has highest importance in page graph!
  – Page A has the next highest:
• Convergence requires
  – Many iterations
  – Is it guaranteed??

Linear Algebraic Interpretation
• PageRank = principle eigenvector of $M^*$
  – in limit
• HITS = principle eigenvector of $M^* \times (M^*)^T$
  – Where $[ \ ]^T$ denotes transpose
  \[
  \begin{bmatrix}
  1 & 2 \\
  3 & 4
  \end{bmatrix}
  \end{bmatrix}^T =
  \begin{bmatrix}
  1 & 3 \\
  2 & 4
  \end{bmatrix}
\]
• Can prove PageRank is stable
• And HITS isn’t
Stability Analysis

- Make 5 subsets by deleting 30% randomly

|   |   |   |   |   |  
|---|---|---|---|---|---
| 1 | 1 | 3 | 1 | 1 | 1 |
| 2 | 2 | 5 | 3 | 3 | 2 |
| 3 | 3 | 12 | 6 | 6 | 3 |
| 4 | 4 | 52 | 20 | 23 | 4 |
| 5 | 5 | 171 | 119 | 99 | 5 |
| 6 | 6 | 135 | 56 | 40 | 8 |
| 7 | 7 | 179 | 159 | 100 | 7 |
| 8 | 8 | 316 | 141 | 170 | 6 |
| 9 | 9 | 257 | 107 | 72 | 9 |
| 10 | 10 | 170 | 80 | 69 | 18 |

- 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</th>
<th>Outdegree</th>
<th>Destination nodes</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

Defining PageRank

Let u be a web page,

- \( F_u \) = set of pages u points (forward) to,
- \( B_u \) = set of pages that point to u (i.e. from behind),
- \( N_u = |F_u| \) be the number pages in \( F_u \).

The rank (importance) of page u ... (first cut):

\[
R(u) = \sum_{v \in B_u} \left( \frac{R(v)}{N_v} \right)
\]

Compute Iteratively:

\[
R_i(u) = \sum_{v \in B_u} \left( \frac{R_{i-1}(v)}{N_v} \right)
\]

Algorithm 1

\[
\forall s \text{ Source}[s] = 1/N \\
\text{while residual} > \tau \{ \\
\quad \forall d \text{ Dest}[d] = 0 \\
\text{while not} \text{ Links.eof()} \{ \\
\quad \text{Links.read(source, n, dest1, … dest_n)} \\
\quad \text{for j = 1…n} \\
\quad \text{Dest[destj] = Dest[destj] + Source[source]/n} \\
\quad \} \\
\quad \forall d \text{ Dest}[d] = (1-c) \ast \text{Dest}[d] + c/N \quad \text{/* dampening c = 1/N */} \\
\text{residual} = |\text{Source} - \text{Dest}| \quad \text{/* recompute every few iterations */} \\
\} \\
\text{Source} = \text{Dest}
\]
**Analysis**

- **If memory is big enough to hold 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 is big enough to hold just Dest**
  - Sort Links on source field
  - Read Source sequentially during rank propagation step
  - Write Dest to disk to serve as Source for next iteration
  - IO cost per iteration is $|Source| + |Dest| + |Links|

- **If memory can’t 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
  - $D < P$ since need input buffers for Source & Links

- **Partition Links into B files**
  - Links, only has some of the dest nodes for each source
  - Links, only has dest nodes such that
    - $DD\times i < dest < DD\times(i+1)$
    - 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>Outdeg (16 bit)</th>
<th>Num out (16 bit)</th>
<th>Destination nodes (32 bit int)</th>
<th>Buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 4 2</td>
<td>12, 26</td>
<td>5</td>
<td></td>
<td>0-31</td>
</tr>
<tr>
<td>1 3 1</td>
<td>5</td>
<td>1, 9, 10</td>
<td></td>
<td>32-63</td>
</tr>
<tr>
<td>0 4 1</td>
<td>58</td>
<td>1</td>
<td></td>
<td>64-95</td>
</tr>
<tr>
<td>1 3 1</td>
<td>56</td>
<td>36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 5 1</td>
<td>94</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 3 1</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 5 1</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Comparing the Algorithms**

- Weighted sum of importance+similarity with query
- Score(q, d)
  
  $$\text{Score}(q, d) = w \times \text{sim}(q, p) + (1-w) \times R(p), \quad \text{if sim}(q, p) > 0$$
  
  $$= 0, \text{otherwise}$$

- Where
  
  - $0 < w < 1$
  - $\text{sim}(q, p), R(p)$ must be normalized to $[0, 1]$. 

---

**Analysis of Block Algorithm**

- IO Cost per iteration = $B \times |Source| + |Dest| + |Links|\times(1+e)$
  - $e$ is factor by which Links increased in size
  - Typically 0.1-0.3
  - Depends on number of blocks
  - Algorithm = nested-loops join

---

**Adding PageRank to a SearchEngine**

- Weighted sum of importance+similarity with query
- Score(q, d)
  
  $$\text{Score}(q, d) = w \times \text{sim}(q, p) + (1-w) \times R(p), \quad \text{if sim}(q, p) > 0$$
  
  $$= 0, \text{otherwise}$$

- Where
  
  - $0 < w < 1$
  - $\text{sim}(q, p), R(p)$ must be normalized to $[0, 1]$. 

---
Summary of Key Points

- **PageRank Iterative Algorithm**
- **Rank Sinks**
- **Efficiency of computation – Memory!**
  - Single precision Numbers.
  - Don’t represent M* explicitly.
  - Break arrays into Blocks.
  - Minimize IO Cost.
- **Number of iterations of PageRank.**
- **Weighting of PageRank vs. doc similarity.**