PageRank

Oren Etzioni

(Based on slides by Dan Weld)
Outline

Challenge: rank pages based on the Web’s link structure & fold that into ranking function.

• Hubs & Authorities
• Page Rank Intuition
• PageRank algorithm & Example
• Optimizations
• Google
Authority and Hub Pages (1)

The basic idea:

• 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** and good **hubs** reinforce each other.
Hubs & Authorities

• Idea due to Jon Kleinberg, Cornell, ’98
• Intuition:
  – Web contains “authoritative” pages on topics
  – Web contains “hubs” → lists of authoritative pages
  – Pages can have a degree of authority and a degree of “hub-ishness”
• Algorithm for computing this is HITS (“hypertext induced topic selection “)
Authority and Hub Pages (2)

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

![Diagram showing bipartite graph with hubs and authorities connected]

- A web page can be a good authority and a good hub.
HITS vs. PageRank

• Executed at query time, not at indexing time
• Computes two scores per document (hub and authority)
• Processed on a small subset of ‘relevant’ documents, not all documents
• Not commonly used by search engines
Stability

Stability = small changes to graph result in only small changes to resulting weights.

- Empirical & theoretical studies…

- Conclusion
  - HITS is not stable.
  - PageRank solves this problem!
PageRank Intuition

• What academic papers are important?
  – ones that are widely cited!

• What Web pages are “important”?

• Are all incoming links “created equal”?
  – What is the analogy to TFxIDF?
Another Pagerank Intuition

• Think of Web as a big graph.
• Suppose surfer keeps randomly clicking on the links.
  – each click moves her to a different page
• Importance of a page = probability of being on the page
  – “when the music stops”
Fundamental PageRank Equation

• \( \text{Rank}(p) = \) “importance” of page \( p \)
• \( N_u = \) out degree of page \( u \)
• link \( (u,v) \) confers \( \text{Rank}(u)/ N_u \) units of rank to page \( v \)
• \( B_v = \) set of pages points to page \( v \)

\[
\forall v, \text{Rank}_{i+1}(v) = \sum_{u \in B_v} \text{Rank}_i(u) / N_u
\]
Preprocessing

• Remove ‘dangling’ nodes
  – Pages w/ no children

• Then repeat process
  – Since now more danglers
  – will iterating this removing all the nodes?

• Stanford WebBase
  – 25 M pages
  – 81 M URLs in the link graph
  – After two prune iterations: 19 M nodes
Iterative ("fixpoint") Computation

- For N pages, initialize the rank of all pages to 1/N
- Iteratively compute the PageRank equation
- The process can be expressed as "eigenvector" calculation over a matrix
Matrix Representation

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

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

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

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

and $R_0$ be the initial rank vector.

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

\[
\begin{bmatrix}
A & B & C & D
\end{bmatrix}
\begin{bmatrix}
A \\
B \\
C \\
D
\end{bmatrix}
= 
\begin{bmatrix}
1/4 \\
1/4 \\
1/4 \\
1/4
\end{bmatrix}
\]
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.

\[ N = \text{the number of pages} \]

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

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

\[ R_i = M^* \times R_{i-1} \]
Computing PageRank – Example, N=4, c = 0.8

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

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

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

\[ R_{30} = \begin{pmatrix}
0.176 \\
0.176 \\
0.332 \\
0.316
\end{pmatrix} \]
Brin & Page ’98 Formulation

- Page A has Pages T1..Tn pointing to it.
- $d = 0.85$ is a dampening factor.
- $C(A) =$ out degree of page A.
- PageRank of A is $PR(A)$

\[
PR(A) = (1 - d) + d(PR(T1)C(T1) + \ldots + PR(Tn)/C(Tn))
\]
Another Example (Details)

let $R_0 = 1$ everywhere
Navigational Effects

Page A
1

1*0.85/2

Page B
1

Page C
1

1*0.85/2

Page D
1

1*0.85

1*0.85

1*0.85

1*0.85
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 the highest PageRank, and page A has the next highest: page C has a highest importance in this page graph!

• More iterations lead to convergence of PageRanks.
Practicality

• Challenges
  – M no longer sparse (don’t represent explicitly!)
  – Data too big for memory (optimize 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…
Representing ‘Links’ Table

- Stored on disk in binary format

<table>
<thead>
<tr>
<th>Source node (32 bit int)</th>
<th>Outdegree (16 bit int)</th>
<th>Destination nodes (32 bit int)</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
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] = c * Dest[d] + (1-c)/N  /* dampening */
residual = ||Source – Dest||  /* recompute every few iterations*/
Source = Dest
}
Analysis of Algorithm 1

- 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 $\sim 800$ M pages in 2/99 [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-2 since need input buffers for Source & Links

- Partition *Links* into B files
  - Links\(_i\) only has some of the dest nodes for each source
  - Links\(_i\) only has dest nodes such that
    - DD\(_i\) <= dest < DD\(_i\)(i+1)
    - Where DD = number of 32 bit integers that fit in D pages

![Diagram of source node, dest node, and Links (sparse)](image)
## 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 int)</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 int)</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 int)</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 =
  – $B \times |Source| + |Dest| + |Links| \times (1+e)$
  – $e$ is factor by which Links increased in size
    • Typically 0.1-0.3
    • Depends on number of blocks

• Algorithm ~ nested-loops join
Comparing the Algorithms

![Bar chart comparing algorithms with different memory sizes.](image)
Adding PageRank to a SearchEngine

- Weighted sum of importance + similarity with query

\[
\text{Score}(q, d) = w \times \text{sim}(q, p) + (1-w) \times R(p), \quad \text{if } \text{sim}(q, p) > 0
\]

= 0, 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.
Ideas emphasized by Google ‘98

• Introduced PageRank
• Focus on anchortext
• Proximity is a factor in the ranking function
• Font is a factor (bold, word size, etc.)

No analysis of which factor was most influential!