Link Analysis

CSE 454 Advanced Internet Systems
University of Washington

Ranking Search Results

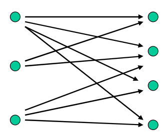
- **TF / IDF Calculation**
- **Tag Information**
  – Title, headers
- **Font Size / Capitalization**
- **Anchor Text on Other Pages**
- **Link Analysis**
  – HITS – (Hubs and Authorities)
  – PageRank

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.



hubs **authorities**

- 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

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×N matrix

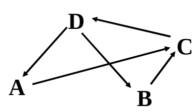
$m_{uv} = 1/N_v$ if page v has a link to page u

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

Let R_0 be the initial rank vector

Let R_i be the N×1 rank vector for i^{th} iteration

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



M

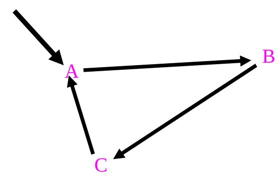
$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \left(\begin{array}{cccc} 0 & 0 & 0 & ½ \\ 0 & 0 & 0 & ½ \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right) \end{array}$$

$\mathbf{R_0}$

$$\begin{pmatrix} ¼ \\ ¼ \\ ¼ \\ ¼ \end{pmatrix}$$

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

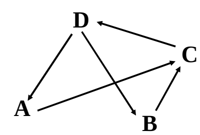
$$K = \begin{pmatrix} & \cdots & \\ \cdots & {}^1/_N & \cdots \\ & \cdots & \end{pmatrix}$$

$M^* = cM + (1-c)K$

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

Computing PageRank - Example

$$M = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \left(\begin{array}{cccc} 0 & 0 & 0 & ½ \\ 0 & 0 & 0 & ½ \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right) \end{array}$$



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

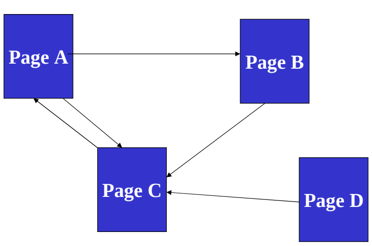
$\mathbf{R_0}$ $\begin{pmatrix} ¼ \\ ¼ \\ ¼ \\ ¼ \end{pmatrix}$ $\mathbf{R_{30}}$ $\begin{pmatrix} 0.176 \\ 0.176 \\ 0.332 \\ 0.316 \end{pmatrix}$

Another Example (Details)

Note: assume 1/N = 0.15

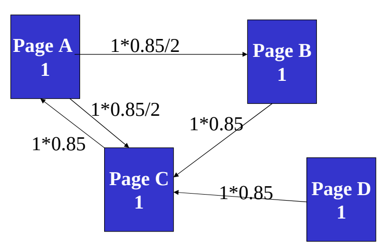
Instead on R_0 = uniform 1/N

let R_0 = 1 everywhere



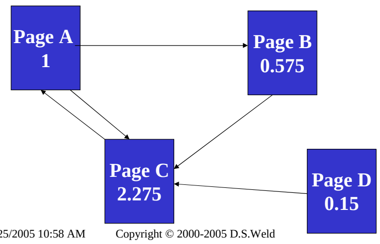
Slightly different formulation

Navigational Effects (matrix M)



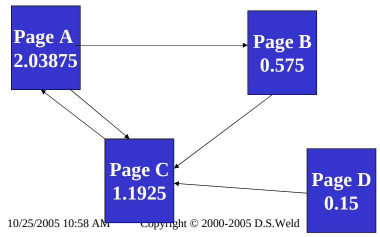
Random Jumps Give Extra 0.15

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



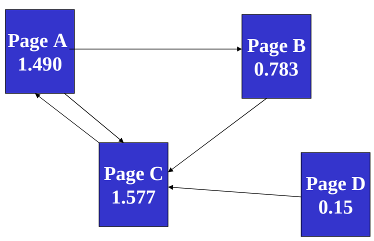
Round 2

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



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 $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$
- **Can prove PageRank is stable**
- **And HITS isn't**

Stability Analysis (Empirical)

- **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	10	179	159	100	7
8	8	316	141	170	6
9	9	257	107	72	9
10	13	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**

Source node (32 bit integer)	Outdegree (16 bit int)	Destination nodes (32 bit integers)
0	4	12, 26, 58, 94
1	3	5, 56, 69
2	5	1, 9, 10, 36, 78

- **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 = $|F_u|$ = the out-degree of u.

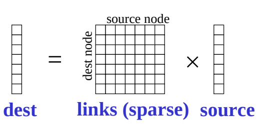
The rank (importance) of page u … *(first cut):*

$$R(u) = \sum_{v \in B_u} (R(v) / N_v)$$

Compute Iteratively:

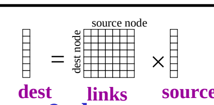
$$R_i(u) = \sum_{v \in B_u} (R_{i-1}(v) / N_v)$$

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
}

Analysis

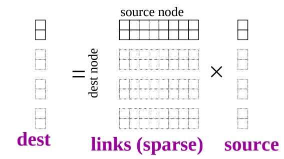


- **If memory can hold both source & dest**
  – IO cost per iteration is | *Links*|
  – Fine for a crawl of 24 M pages
  – But web > 8 B pages in 2005 [Google]
  – Increase from 320 M pages in 1997 [NEC study]
- **If memory only big enough to hold just dest…?**
  – Sort *Links* on source field
  – Read *Source* sequentially during rank propagation step
  – Write *Dest* to disk to serve as *Source* for next iteration
  – IO cost per iteration is | *Source*| + | *Dest*| + | *Links*|
- **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_i only has some of the dest nodes for each source
  - Links_i only has dest nodes such that
    - DD*i <= dest < DD*(i+1)
    - Where DD = number of 32 bit integers that fit in D pages



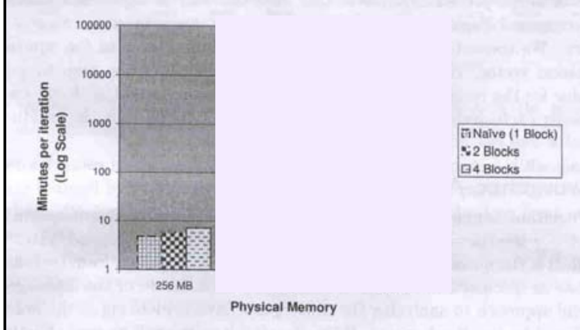
Partitioned Link File

Source node (32 bit int)	Outdegr (16 bit)	Num out (16 bit)	Destination nodes (32 bit integer)	
0	4	2	12, 26	Buckets 0-31
1	3	1	5	
2	5	3	1, 9, 10	
0	4	1	58	Buckets 32-63
1	3	1	56	
2	5	1	36	
0	4	1	94	Buckets 64-95
1	3	1	69	
2	5	1	78	

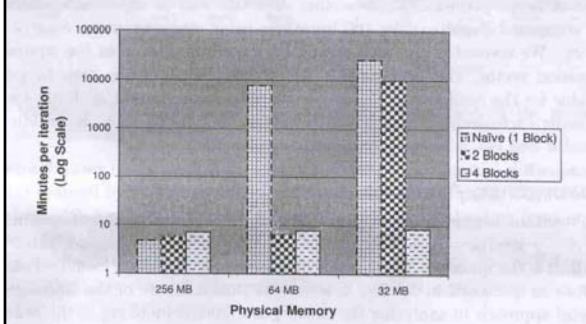
Analysis of Block Algorithm

- **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



Adding PageRank to a SearchEngine

- Weighted sum of importance+similarity with query
- Score(q, d)
  - **= w∗sim(q, p) + (1-w) ∗ R(p),** **if sim(q, p) > 0**
  - **= 0, 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.**