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

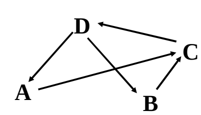
Matrix Representation

Let M be an N×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

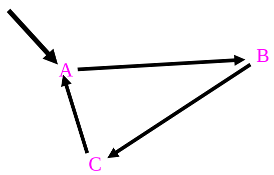
	A	B	C	D
A	0	0	0	½
B	0	0	0	½
C	1	1	0	0
D	0	0	1	0

R_0

¼
¼
¼
¼

Problem: Page Sinks.

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



Solution to Sink Nodes

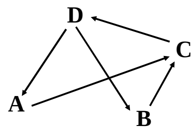
Let:
(1-c) = chance of random transition from a sink.
N = the number of pages

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

$M^* = cM + (1-c)K$
$R_i = M^* \times R_{i-1}$

Computing PageRank - Example

$$M = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$



$$M^* = \begin{pmatrix} 0.05 & 0.05 & 0.05 & 0.45 \\ 0.05 & 0.05 & 0.05 & 0.45 \\ 0.85 & 0.85 & 0.05 & 0.05 \\ 0.05 & 0.05 & 0.85 & 0.05 \end{pmatrix} \quad R_0 = \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{pmatrix} \quad R_{30} = \begin{pmatrix} 0.176 \\ 0.176 \\ 0.332 \\ 0.316 \end{pmatrix}$$

Ooops

- **What About Sparsity?**

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

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

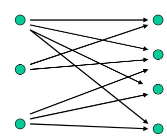
$$K = \begin{pmatrix} & \cdots & \\ \cdots & {}^{1}/_{N} & \cdots \\ & \cdots & \end{pmatrix}$$

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.



hubs **authorities**

(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 $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$
- **Stability**
  Small changes to graph → small changes to weights.
  – 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

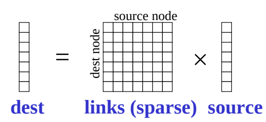
Representing 'Links' Table

- **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

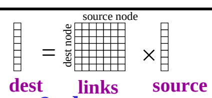
- **Size for Stanford WebBase: 1.01 GB**
  - Assumed to exceed main memory
  - (But source & dest assumed to fit)

Algorithm 1



$\forall$s Source[s] = 1/N
while residual >τ {
  $\forall$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
  }
  $\forall$d Dest[d] = (1-c) * Dest[d] + c/N /* dampening c= 1/N */
  residual = ‖ Source – Dest ‖ /* recompute every few iterations */
  Source = Dest
}

Analysis



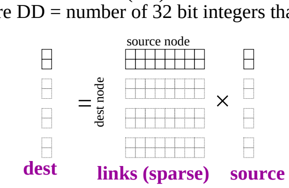
- **If memory can hold both source & dest**
  - IO cost per iteration is | *Links*|
  - Fine for a crawl of 24 M pages
  - But web > 8 B pages in 2005 [Google]
  - Increase from 320 M pages in 1997 [NEC study]
- **If memory only big enough to hold just dest…?**
  - Sort *Links* on source field
  - Read *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*|
- **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
- $Links_i$ only has ***some*** of the dest nodes for each source
  Specifically, $Links_i$ 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	

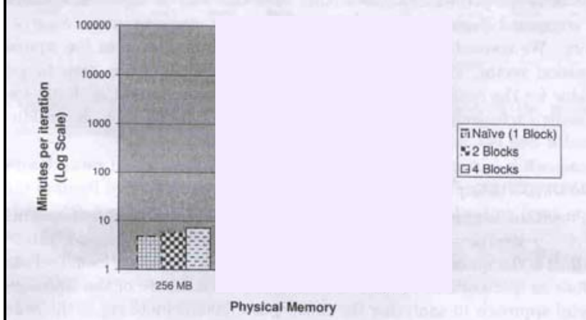
0	4	1	58	Buckets 32-63
1	3	1	56	
2	5	1	36	

0	4	1	94	Buckets 64-95
1	3	1	69	
2	5	1	78	

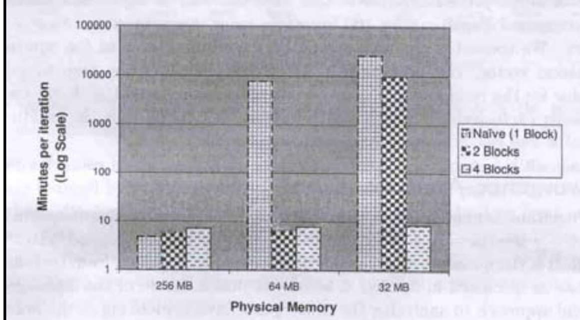
Analysis of Block Algorithm

- **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.**