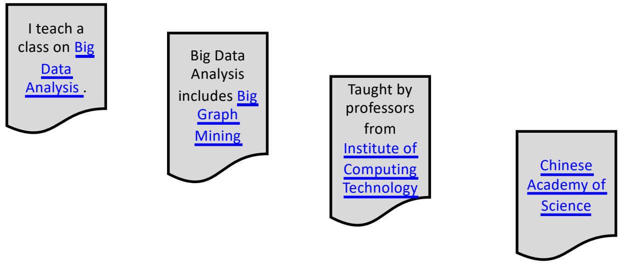
大数据分析

Link Analysis – PageRank

刘盛华

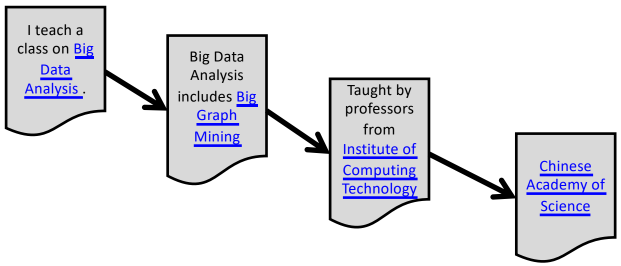
Web as a Graph

- Web as a directed graph:
  - Nodes: Webpages
  - Edges: Hyperlinks

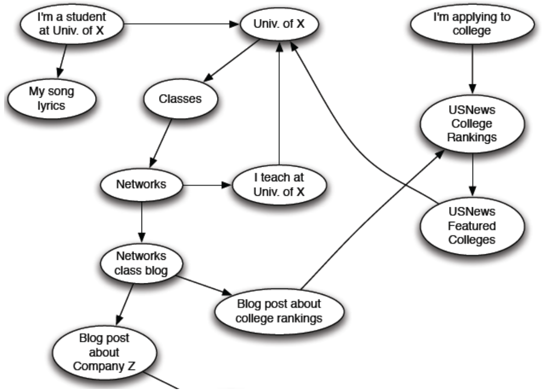


Web as a Graph

- Web as a directed graph:
  - Nodes: Webpages
  - Edges: Hyperlinks



Web as a Directed Graph



Broad Question

- How to organize the Web?
- First try: Human curated Web directories
  - Yahoo, DMOZ, 新浪
- Second try: Web Search
  - Information Retrieval investigates: Find relevant docs in a small and trusted set
    - Newspaper articles, Patents, etc.
  - But: Web is huge, full of untrusted documents, random things, web spam, etc.

5

Web Search: 2 Challenges

2 challenges of web search:

- (1) Web contains many sources of information Who to "trust"?
  - Trick: Trustworthy pages may point to each other!
- (2) What is the "best" answer to query "newspaper"?
  - No single right answer
  - Trick: Pages that actually know about newspapers might all be pointing to many newspapers

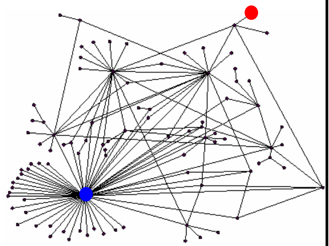
6

Ranking Nodes on the Graph

- All web pages are not equally "important"

  www.joe-schmoe.com vs. www.ict.ac.cn

- There is large diversity in the web-graph node connectivity. Let's rank the pages by the link structure!



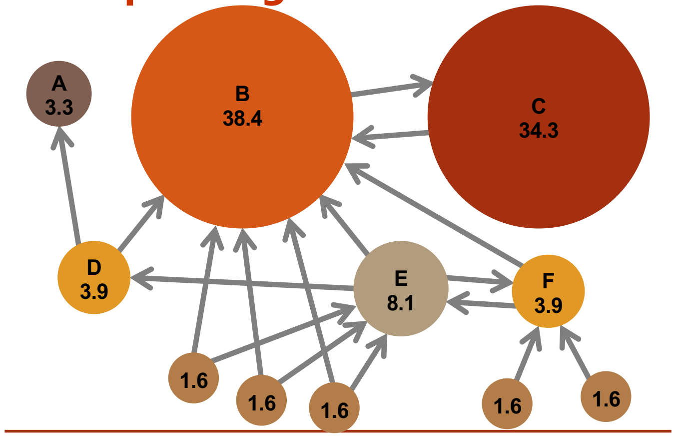
7

PageRank: The "Flow" Formulation

Links as Votes

- **Idea: Links as votes**
  - Page is more important if it has more links
    - In-coming links? Out-going links?
- **Think of in-links as votes:**
  - www.ict.ac.cn has 12,300 in-links
  - www.joe-schmoe.com has 1 in-link
- **Are all in-links are equal?**
  - Links from important pages count more
  - Recursive question!

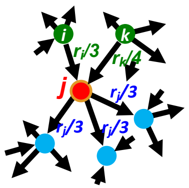
Example: PageRank Scores



Simple Recursive Formulation

- Each link's vote is proportional to the **importance** of its source page
- If page j with importance r_j has n out-links, each link gets r_j / n votes
- Page j's own importance is the sum of the votes on its in-links

$r_j = r_i/3 + r_k/4$



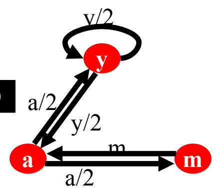
PageRank: The "Flow" Model

- A "vote" from an important page is worth more
- A page is important if it is pointed to by other important pages
- Define a "rank" r_j for page j

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

d_i … out-degree of node i

The web in 1839



"Flow" equations:

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

Solving the Flow Equations

Flow equations:
$r_y = r_y/2 + r_a/2$
$r_a = r_y/2 + r_m$
$r_m = r_a/2$

- **3 equations, 3 unknowns, no constants**
  - No unique solution (linearly dependent)
  - All solutions equivalent modulo the scale factor
- **Additional constraint forces uniqueness:**
  - $r_y + r_a + r_m = 1$
  - Solution: $r_y = \frac{2}{5}$, $r_a = \frac{2}{5}$, $r_m = \frac{1}{5}$
- **Gaussian elimination method works for small examples, but we need a better method for large web-size graphs**
- **We need a new formulation!**

PageRank: Matrix Formulation

- **Stochastic adjacency matrix M**
  - Let page i has d_i out-links
  - If $i \to j$, then $M_{ji} = \frac{1}{d_i}$ else $M_{ji} = 0$
    - M is a column stochastic matrix
      - Columns sum to 1
- **Rank vector r: vector with an entry per page**
  - r_i is the importance score of page i
  - $\sum_i r_i = 1$
- **The flow equations can be written**

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

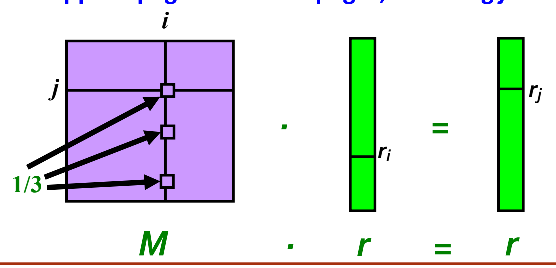
$$\boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r} = \sum_i r_i \boldsymbol{M}_{*i}$$

Example

- **Remember the flow equation:** $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
- **Flow equation in the matrix form**

$$\boldsymbol{M} \cdot \boldsymbol{r} = \boldsymbol{r}$$

  - Suppose page i links to 3 pages, including j



Eigenvector Formulation

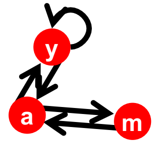
- **The flow equations can be written**

$$\boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$$

- **So the rank vector r is an eigenvector of the stochastic web matrix M**
  - In fact, its first or principal eigenvector, with corresponding eigenvalue *1*
    - Largest eigenvalue of M is 1 since M is column stochastic (with non-negative entries)
      - *We know r is unit length and each column of M sums to one, so $Mr \le 1$*
- **We can now efficiently solve for r!** The method is called Power iteration

NOTE: x is an eigenvector with the corresponding eigenvalue λ if: $Ax = \lambda x$

Example: Flow Equations & M



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$r = M \cdot r$

$r_y = r_y/2 + r_a/2$
$r_a = r_y/2 + r_m$
$r_m = r_a/2$

$$\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 1 \\ 0 & ½ & 0 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}$$

Power Iteration Method

- Given a web graph with n nodes, where the nodes are pages and edges are hyperlinks
- Power iteration: a simple iterative scheme
  - Suppose there are N web pages
  - Initialize: $r^{(0)} = [1/N,\ldots,1/N]^T$
  - Iterate: $r^{(t+1)} = M \cdot r^{(t)}$
  - Stop when $|r^{(t+1)} - r^{(t)}|_1 < \varepsilon$

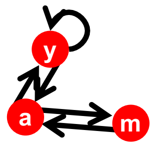
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

d_i …. out-degree of node i

$|\mathbf{x}|_1 = \sum_{1 \le i \le N} |x_i|$ is the L_1 norm
Can use any other vector norm, e.g., Euclidean

PageRank: How to solve?

- Power Iteration for all j:
  - Set $r_j = 1/N$
  - 1: $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - 2: $r_j = r'_j$
  - Goto 1
- Example:



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

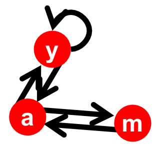
$r_y = r_y/2 + r_a/2$
$r_a = r_y/2 + r_m$
$r_m = r_a/2$

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 \\ 1/3 \\ 1/3 \end{matrix}$$

Iteration 0, 1, 2, …

PageRank: How to solve?

- Power Iteration for all j:
  - Set $r_j = 1/N$
  - 1: $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - 2: $r_j = r'_j$
  - Goto 1
- Example:



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$r_y = r_y/2 + r_a/2$
$r_a = r_y/2 + r_m$
$r_m = r_a/2$

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 & 1/3 & 5/12 & 9/24 & & 6/15 \\ 1/3 & 3/6 & 1/3 & 11/24 & \ldots & 6/15 \\ 1/3 & 1/6 & 3/12 & 1/6 & & 3/15 \end{matrix}$$

Iteration 0, 1, 2, …

Why Power Iteration works? (1)

Details!

- **Power iteration:**
  A method for finding dominant eigenvector (the vector corresponding to the largest eigenvalue)
  - $\boldsymbol{r}^{(1)} = \boldsymbol{M} \cdot \boldsymbol{r}^{(0)}$
  - $\boldsymbol{r}^{(2)} = \boldsymbol{M} \cdot \boldsymbol{r}^{(1)} = \boldsymbol{M}\left(\boldsymbol{M}\boldsymbol{r}^{(0)}\right) = \boldsymbol{M}^2 \cdot \boldsymbol{r}^{(0)}$
  - $\boldsymbol{r}^{(3)} = \boldsymbol{M} \cdot \boldsymbol{r}^{(2)} = \boldsymbol{M}\left(\boldsymbol{M}^2\boldsymbol{r}^{(0)}\right) = \boldsymbol{M}^3 \cdot \boldsymbol{r}^{(0)}$
- **Claim:**
  Sequence $\boldsymbol{M} \cdot \boldsymbol{r}^{(0)}, \boldsymbol{M}^2 \cdot \boldsymbol{r}^{(0)}, \ldots \boldsymbol{M}^k \cdot \boldsymbol{r}^{(0)}, \ldots$ approaches the dominant eigenvector of $\boldsymbol{M}$

Why Power Iteration works? (2)

Details!

- **Claim**: Sequence $\boldsymbol{M} \cdot \boldsymbol{r}^{(0)}, \boldsymbol{M}^2 \cdot \boldsymbol{r}^{(0)}, \ldots \boldsymbol{M}^k \cdot \boldsymbol{r}^{(0)}, \ldots$ approaches the dominant eigenvector of $\boldsymbol{M}$
- **Proof:**
  - Assume M has n linearly independent eigenvectors, $x_1, x_2, \ldots, x_n$ with corresponding eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$, where $\lambda_1 > \lambda_2 > \cdots > \lambda_n$
  - Vectors $x_1, x_2, \ldots, x_n$ form a basis and thus we can write: $r^{(0)} = c_1\, x_1 + c_2\, x_2 + \cdots + c_n\, x_n$
  - $\boldsymbol{M}\boldsymbol{r}^{(0)} = \boldsymbol{M}(\boldsymbol{c_1}\,\boldsymbol{x_1} + \boldsymbol{c_2}\,\boldsymbol{x_2} + \cdots + \boldsymbol{c_n}\,\boldsymbol{x_n})$
  - $= c_1(Mx_1) + c_2(Mx_2) + \cdots + c_n(Mx_n)$
  - $= c_1(\lambda_1 x_1) + c_2(\lambda_2 x_2) + \cdots + c_n(\lambda_n x_n)$
  - Repeated multiplication on both sides produces
  - $M^k r^{(0)} = c_1(\lambda_1^k x_1) + c_2(\lambda_2^k x_2) + \cdots + c_n(\lambda_n^k x_n)$

Why Power Iteration works? (3)

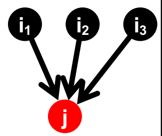
Details!

- **Claim:** Sequence $\boldsymbol{M} \cdot \boldsymbol{r}^{(0)}, \boldsymbol{M}^2 \cdot \boldsymbol{r}^{(0)}, \ldots \boldsymbol{M}^k \cdot \boldsymbol{r}^{(0)}, \ldots$ approaches the dominant eigenvector of $\boldsymbol{M}$
- **Proof (continued):**
  - Repeated multiplication on both sides produces
    $M^k r^{(0)} = c_1(\lambda_1^k x_1) + c_2(\lambda_2^k x_2) + \cdots + c_n(\lambda_n^k x_n)$
  - $M^k r^{(0)} = \lambda_1^k \left[c_1 x_1 + c_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k x_2 + \cdots + c_n \left(\frac{\lambda_n}{\lambda_1}\right)^k x_n \right]$
  - Since $\lambda_1 > \lambda_2$ then fractions $\frac{\lambda_2}{\lambda_1}, \frac{\lambda_3}{\lambda_1} \ldots < 1$
    and so $\left(\frac{\lambda_i}{\lambda_1}\right)^k = 0$ as $k \to \infty$ (for all $i = 2 \ldots n$).
  - **Thus:** $\boldsymbol{M^k r^{(0)}} \approx \boldsymbol{c_1\left(\lambda_1^k x_1\right)}$
    - Note if $c_1 = 0$ then the method won't converge

i.e. r0 is orthogonal to the first eigenvector

Random Walk Interpretation

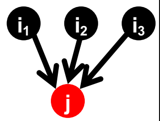
- **Imagine a random web surfer:**
  - At any time $\boldsymbol{t}$, surfer is on some page $\boldsymbol{i}$
  - At time $\boldsymbol{t+1}$, the surfer follows an out-link from $\boldsymbol{i}$ uniformly at random
  - Ends up on some page $\boldsymbol{j}$ linked from $\boldsymbol{i}$
  - Process repeats indefinitely



$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d_{out}}(\mathrm{i})}$$

- **Let:**
  - $\boldsymbol{p(t)}$ … vector whose $\boldsymbol{i}^{\text{th}}$ coordinate is the prob. that the surfer is at page $\boldsymbol{i}$ at time $\boldsymbol{t}$
  - So, $\boldsymbol{p(t)}$ is a probability distribution over pages

The Stationary Distribution



$p(t+1) = M \cdot p(t)$

- Where is the surfer at time $t+1$?
  - Follows a link uniformly at random
    $p(t+1) = M \cdot p(t)$
- Suppose the random walk reaches a state
  $p(t+1) = M \cdot p(t) = p(t)$
  then $p(t)$ is **stationary distribution** of a random walk
- **Our original rank vector** r satisfies $r = M \cdot r$
  - So, r is a stationary distribution for the random walk

Existence and Uniqueness

- A central result from the theory of random walks (a.k.a. Markov processes):

For graphs that satisfy **certain conditions**, the **stationary distribution is unique** and eventually will be reached no matter what the initial probability distribution at time **t = 0**

ref to:
Perron–Frobenius theorem [all entries are positive, or
Nonnegative Matrix, irreducible (connected), or
primitivity (k-connected)]

PageRank: The Google Formulation

PageRank: Three Questions

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}} \quad \text{or equivalently} \quad r = Mr$$

- Does this converge?
- Does it converge to what we want?
- Are results reasonable?

Does this converge?

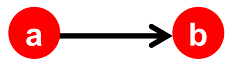
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d}_i}$$

- **Example:**

r_a	=	1	0	1	0
r_b		0	1	0	1

Iteration 0, 1, 2, …

Does it converge to what we want?



$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d}_i}$$

- **Example:**

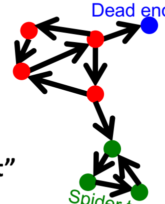
r_a	=	1	0	0	0
r_b		0	1	0	0

Iteration 0, 1, 2, …

PageRank: Problems

2 problems:

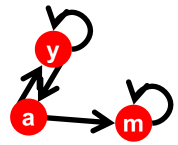
- (1) Some pages are **dead ends** (have no out-links)
  - Random walk has "nowhere" to go to
  - Such pages cause importance to "leak out"



- (2) **Spider traps:** (all out-links are within the group)
  - Random walked gets "stuck" in a trap
  - And eventually spider traps absorb all importance

Problem: Spider Traps

- **Power Iteration:**
  - Set $r_j = 1$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



m is a spider trap

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	1

$r_y = r_y/2 + r_a/2$
$r_a = r_y/2$
$r_m = r_a/2 + r_m$

- **Example:**

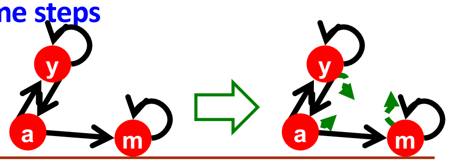
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 3/6 & 7/12 & 16/24 & & 1 \end{matrix}$$

Iteration 0, 1, 2, …

All the PageRank score gets "trapped" in node m.

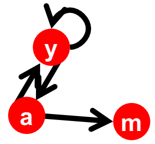
Solution: Teleports!

- The Google solution for spider traps: At each time step, the random surfer has two options
  - With prob. β, follow a link at random
  - With prob. 1-β, jump to some random page
  - Common values for β are in the range 0.8 to 0.9
- Surfer will teleport out of spider trap within a few time steps



Problem: Dead Ends

- Power Iteration:
  - Set $r_j = 1$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

$r_y = r_y/2 + r_a/2$
$r_a = r_y/2$
$r_m = r_a/2$

- Example:

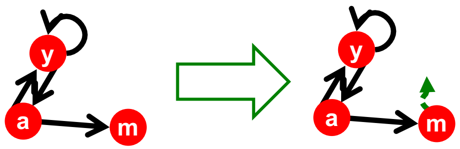
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\ 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{matrix}$$

Iteration 0, 1, 2, …

Here the PageRank "leaks" out since the matrix is not stochastic.

Solution: Always Teleport!

- Teleports: Follow random teleport links with probability 1.0 from dead-ends
  - Adjust matrix accordingly



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

	y	a	m
y	½	½	⅓
a	½	0	⅓
m	0	½	⅓

Why Teleports Solve the Problem?

Why are dead-ends and spider traps a problem and why do teleports solve the problem?

- Spider-traps are not a problem (converge), but with traps PageRank scores are not what we want
  - Solution: Never get stuck in a spider trap by teleporting out of it in a finite number of steps
- Dead-ends are a problem
  - The matrix is not column stochastic (zero column) so our initial assumptions are not met
  - Solution: Make matrix column stochastic by always teleporting when there is nowhere else to go

Solution: Random Teleports

- Google's solution that does it all:
  At each step, random surfer has two options:
  - With probability β, follow a link at random
  - With probability 1-β, jump to some random page
- PageRank equation [Brin-Page, 98]

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N}$$

d_i … out-degree of node i

This formulation assumes that M has no dead ends. We can either preprocess matrix M to remove all dead ends (add 1/N in M) or explicitly follow random teleport links with probability 1.0 from dead-ends (β=0).

The Google Matrix

- PageRank equation [Brin-Page, '98]

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N}$$

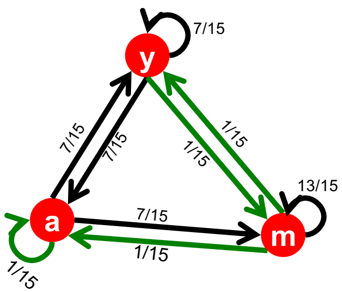
- The Google Matrix A:

$$A = \beta\, M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$$

$[1/N]_{N\times N}$…N by N matrix where all entries are 1/N

- We have a recursive problem: $r = A \cdot r$
  And the Power method still works!
- What is β?
  - In practice β=$0.8, 0.9$ (make 5 steps on avg., jump)

Random Teleports ($\beta = 0.8$)



$$0.8 \overset{\textbf{M}}{\begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix}} + 0.2 \overset{[1/N]_{N\times N}}{\begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}}$$

$$\begin{array}{c} y \\ a \\ m \end{array} \underbrace{\begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}}_{\textbf{A}}$$

$$\begin{array}{c} y \\ a \\ m \end{array} = \begin{array}{c} 1/3 \\ 1/3 \\ 1/3 \end{array} \quad \begin{array}{c} 0.33 \\ 0.20 \\ 0.46 \end{array} \quad \begin{array}{c} 0.24 \\ 0.20 \\ 0.52 \end{array} \quad \begin{array}{c} 0.26 \\ 0.18 \\ 0.56 \end{array} \quad \dots \quad \begin{array}{c} 7/33 \\ 5/33 \\ 21/33 \end{array}$$

How do we actually compute the PageRank?

Computing Page Rank

- Key step is matrix-vector multiplication
  - $r^{new} = A \cdot r^{old}$
- Easy if we have enough main memory to hold A, r^{old}, r^{new}
- Say N = 1 billion pages
  - We need 4 bytes for each entry (say)
  - 2 billion entries for vectors, approx 8GB
  - Matrix A has N^2 entries
    - 10^{18} is a large number!

$A = \beta \cdot M + (1-\beta)\ [1/N]_{N \times N}$

$$A = 0.8 \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 0 \\ 0 & ½ & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$$= \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

Matrix Formulation

- Suppose there are *N* pages
- Consider page *i*, with d_i out-links
- We have $M_{ji} = 1/|d_i|$ when $i \rightarrow j$
  and $M_{ji} = 0$ otherwise
- The random teleport is equivalent to:
  - Adding a teleport link from *i* to every other page and setting transition probability to $(1-\beta)/N$
  - Reducing the probability of following each out-link from $1/|d_i|$ to $\beta/|d_i|$
  - Equivalent: Tax each page a fraction $(1-\beta)$ of its score and redistribute evenly

Rearranging the Equation

- $r = A \cdot r$, where $A_{ji} = \beta M_{ji} + \frac{1-\beta}{N}$
- $r_j = \sum_{i=1}^{N} A_{ji} \cdot r_i$
- $r_j = \sum_{i=1}^{N} \left[\beta M_{ji} + \frac{1-\beta}{N}\right] \cdot r_i$

$$= \sum_{i=1}^{N} \beta M_{ji} \cdot r_i + \frac{1-\beta}{N} \sum_{i=1}^{N} r_i$$

$$= \sum_{i=1}^{N} \beta M_{ji} \cdot r_i + \frac{1-\beta}{N} \qquad \text{since } \sum r_i = 1$$

- So we get: $r = \beta M \cdot r + \left[\frac{1-\beta}{N}\right]_N$

Note: Here we assumed **M** has no dead-ends

$[x]_N$ … a vector of length *N* with all entries *x*

Sparse Matrix Formulation

- We just rearranged the PageRank equation

$$r = \beta M \cdot r + \left[\frac{1-\beta}{N}\right]_N$$

  - where $[(1-\beta)/N]_N$ is a vector with all *N* entries $(1-\beta)/N$
- *M* is a sparse matrix! (with no dead-ends)
  - 10 links per node, approx 10N entries
- So in each iteration, we need to:
  - Compute $r^{new} = \beta M \cdot r^{old}$
  - Add a constant value $(1-\beta)/N$ to each entry in r^{new}
    - Note if M contains dead-ends then $\sum_j r_j^{new} < 1$ and we also have to renormalize r^{new} so that it sums to 1

PageRank: The Complete Algorithm

- **Input:** Graph G and parameter β
  - Directed graph G (can have spider traps and dead ends)
  - Parameter β
- **Output:** PageRank vector r^{new}
  - Set: $r_j^{old} = \frac{1}{N}$
  - repeat until convergence: $\sum_j |r_j^{new} - r_j^{old}| > \varepsilon$
    - $\forall j$: $r'^{new}_j = \sum_{i \to j} \beta \frac{r_i^{old}}{d_i}$
      $r'^{new}_j = 0$ if in-degree of j is 0
    - Now re-insert the leaked PageRank:
      $\forall j$: $r_j^{new} = r'^{new}_j + \frac{1-S}{N}$
    - $r^{old} = r^{new}$ where: $S = \sum_j r'^{new}_j$

If the graph has no dead-ends then the amount of leaked PageRank is **1-β**. But since we have dead-ends the amount of leaked PageRank may be larger. We have to explicitly account for it by computing **S**.

45

Sparse Matrix Encoding

- Encode sparse matrix using only nonzero entries
  - Space proportional roughly to number of links
  - Say 10N, or 4*10*1 billion = 40GB
  - Still won't fit in memory, but will fit on disk

source node	degree	destination nodes
0	3	1, 5, 7
1	5	17, 64, 113, 117, 245
2	2	13, 23

46

Basic Algorithm: Update Step

- Assume enough RAM to fit r^{new} into memory
  - Store r^{old} and matrix M on disk
- 1 step of power-iteration is:

Initialize all entries of r^{new} = (1-β) / N
For each page i (of out-degree d_i):
Read into memory: i, d_i, $dest_1$, …, $dest_{d_i}$, $r^{old}(i)$
For j = 1…d_i
$r^{new}(dest_j)$ += β $r^{old}(i)$ / d_i

r^{new}: 0, 1, 2, 3, 4, 5, 6

source	degree	destination
0	3	1, 5, 6
1	4	17, 64, 113, 117
2	2	13, 23

r^{old}: 0, 1, 2, 3, 4, 5, 6

47

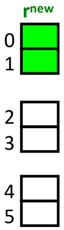
Analysis

- Assume enough RAM to fit r^{new} into memory
  - Store r^{old} and matrix M on disk
- In each iteration, we have to:
  - Read r^{old} and M
  - Write r^{new} back to disk
  - Cost per iteration of Power method:
    = 2|r| + |M|
- Question:
  - What if we could not even fit r^{new} in memory?

48

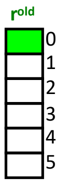
Block-based Update Algorithm

- Break r^{new} into k blocks that fit in memory
- Scan M and r^{old} once for each block



src	degree	destination
0	4	0, 1, 3, 5
1	2	0, 5
2	2	3, 4

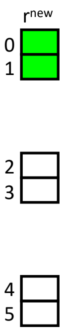
M



Analysis of Block Update

- Similar to nested-loop join in databases
  - Break r^{new} into k blocks that fit in memory
  - Scan M and r^{old} once for each block
- Total cost:
  - k scans of M and r^{old}
  - Cost per iteration of Power method: $k(|M| + |r|) + |r| = k|M| + (k+1)|r|$
- Can we do better?
  - Hint: M is much bigger than r (approx 10-20x), so we must avoid reading it k times per iteration

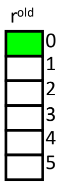
Block-Stripe Update Algorithm



src	degree	destination
0	4	0, 1
1	3	0
2	2	1

0	4	3
2	2	3

0	4	5
1	3	5
2	2	4



Break *M* into stripes! Each stripe contains only destination nodes in the corresponding block of r^{new}

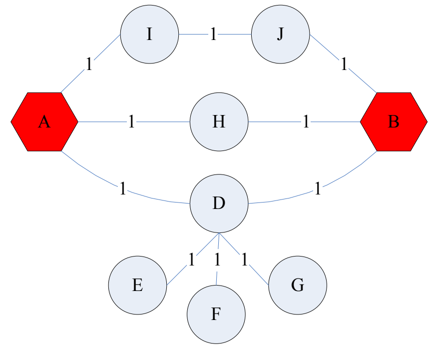
Block-Stripe Analysis

- Break M into stripes
  - Each stripe contains only destination nodes in the corresponding block of r^{new}
- Some additional overhead per stripe
  - But it is usually worth it
- Cost per iteration of Power method: $=|M|(1+\varepsilon) + (k+1)|r|$

Application to Measuring Proximity in Graphs

Random Walk with Restarts: S is a single element

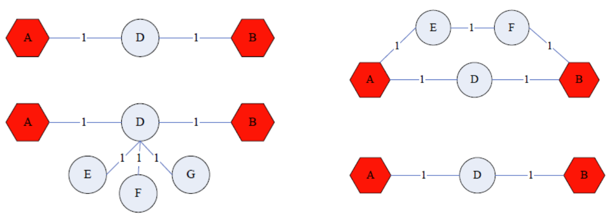
Proximity on Graphs



a.k.a.: Relevance, Closeness, 'Similarity'…

Good proximity measure?

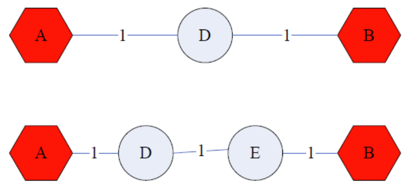
- **Shortest path is not good:**



- **No effect of degree-1 nodes (E, F, G)!**
- **Multi-faceted relationships**

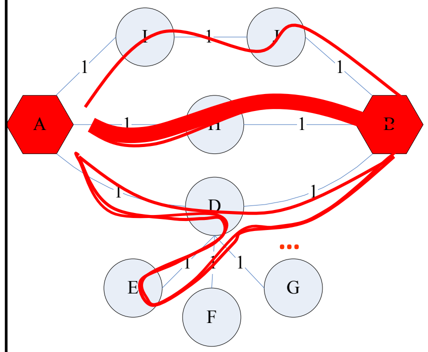
Good proximity measure?

- **Network flow is not good:**



- **Does not punish long paths**

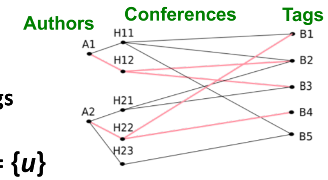
What is good notion of proximity?



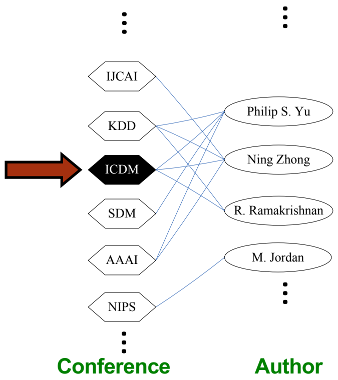
- Multiple connections
- Quality of connection
  - Direct & Indirect connections
  - Length, Degree, Weight…

SimRank: Idea

- **SimRank:** Random walks from a fixed node on *k*-partite graphs
- **Setting:** *k*-partite graph with *k* types of nodes
  - E.g.: Authors, Conferences, Tags
- **PageRank from node *u*:** teleport set *S* = {*u*}
- **Resulting scores measures similarity to node *u***
- **Problem:**
  - Must be done once for each node *u*
  - Suitable for sub-Web-scale applications



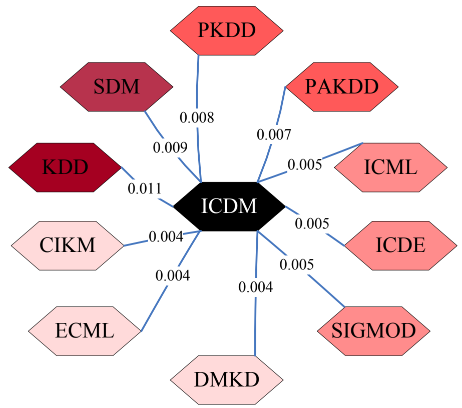
SimRank: Example



Q: What is most related conference to **ICDM**?

A: Topic-Specific PageRank with teleport set S={ICDM}

SimRank: Example



PageRank: Summary

- **"Normal" PageRank:**
  - Teleports uniformly at random to any node
  - All nodes have the same probability of surfer landing there: S = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
- **Topic-Specific PageRank also known as Personalized PageRank:**
  - Teleports to a topic specific set of pages
  - Nodes can have different probabilities of surfer landing there: S = [0.1, 0, 0, 0.2, 0, 0, 0.5, 0, 0, 0.2]
- **Random Walk with Restarts:**
  - Topic-Specific PageRank where teleport is always to the same node. S=[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

61

Questions?