

# **Link Analysis Algorithms**

## **Page Rank**

### **Part 2**

INF 553

With slide contributions from Leskovec, Rajaraman, Ullman

# Graph Data: Social Networks



## Facebook social graph

4-degrees of separation [Backstrom-Boldi-Rosa-Ugander-Vigna, 2011]

# New Topic: Graph Data

## High dim. data

Locality sensitive hashing

Clustering

Dimensionality reduction

## Graph data

PageRank, SimRank

Community Detection

Spam Detection

## Infinite data

Filtering data streams

Web advertising

Queries on streams

## Machine learning

SVM

Decision Trees

Perceptron, kNN

## Apps

Recommender systems

Association Rules

Duplicate document detection

# **RECALL: THE FLOW MODEL**

# Eigenvector Formulation

- ◆ The flow equations can be written

$$M \cdot r = r$$

- ◆ So the rank vector  $r$  is an eigenvector of the stochastic web matrix  $M$

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

$$Ax = \lambda x$$

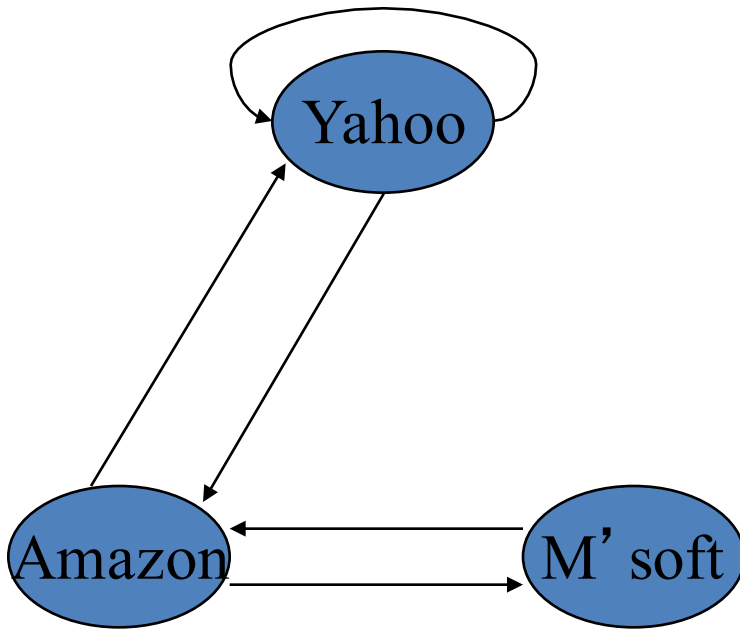
- ◆ We can now efficiently solve for  $r$ !

- 1. Power Iteration:

[https://en.wikipedia.org/wiki/Power\\_iteration](https://en.wikipedia.org/wiki/Power_iteration)

- 2. Use the principal eigenvector

# Example



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

$$r_a = r_y / 2 + r_m$$

$$r_m = r_a / 2$$

|   | y   | a   | m |
|---|-----|-----|---|
| y | 1/2 | 1/2 | 0 |
| a | 1/2 | 0   | 1 |
| m | 0   | 1/2 | 0 |

$$\mathbf{r} = \mathbf{M}\mathbf{r}$$

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}$$

# Power Iteration method

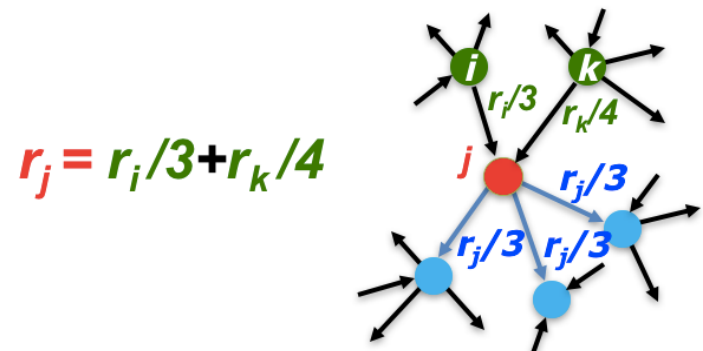
- ◆ Simple iterative scheme (aka **relaxation**)
- ◆ Suppose there are  $N$  web pages
- ◆ Initialize:  $\mathbf{r}^0 = [1/N, \dots, 1/N]^T$
- ◆ Iterate:  $\mathbf{r}^{k+1} = \mathbf{M}\mathbf{r}^k$
- ◆ Stop when  $\|\mathbf{r}^{k+1} - \mathbf{r}^k\|_1 < \epsilon$

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

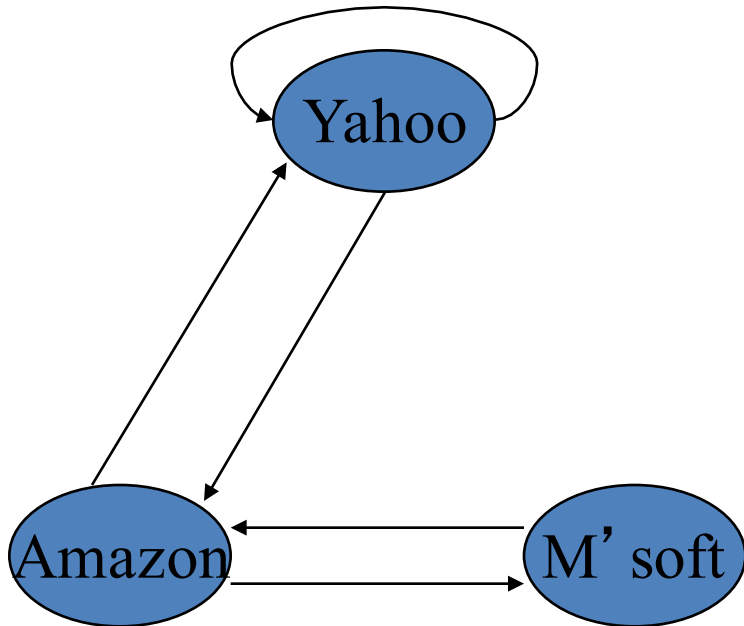
$d_i$  .... out-degree of node  $i$

$\|\mathbf{x}\|_1 = \sum_{1 \leq i \leq N} |x_i|$  is the **L1** norm

Can use any other vector norm, e.g., Euclidean



# Power Iteration Example



|   | y   | a   | m |
|---|-----|-----|---|
| y | 1/2 | 1/2 | 0 |
| a | 1/2 | 0   | 1 |
| m | 0   | 1/2 | 0 |

$$\mathbf{r}^{k+1} = \mathbf{M}\mathbf{r}^k$$

$$\mathbf{r}^0 \quad \mathbf{r}^1 = \mathbf{M}\mathbf{r}^0$$

$$\begin{array}{l} \mathbf{r}_y \\ \mathbf{r}_a \\ \mathbf{r}_m \end{array} = \begin{array}{cccccc} 1/3 & 2/6 & 5/12 & 3/8 & & 2/5 \\ 1/3 & 3/6 & 1/3 & 11/24 & \dots & 2/5 \\ 1/3 & 1/6 & 3/12 & 1/6 & & 1/5 \end{array}$$

Iteration 0, 1, 2, ...



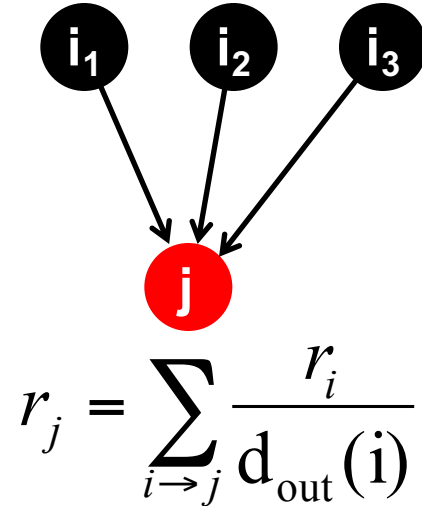
# Recall: Random Walk Interpretation

## ■ Imagine a random web surfer:

- At any time  $t$ , surfer is on some page  $i$
- At time  $t + 1$ , the surfer follows an out-link from  $i$  uniformly at random
- Ends up on some page  $j$  linked from  $i$
- Process repeats indefinitely

## ■ Let:

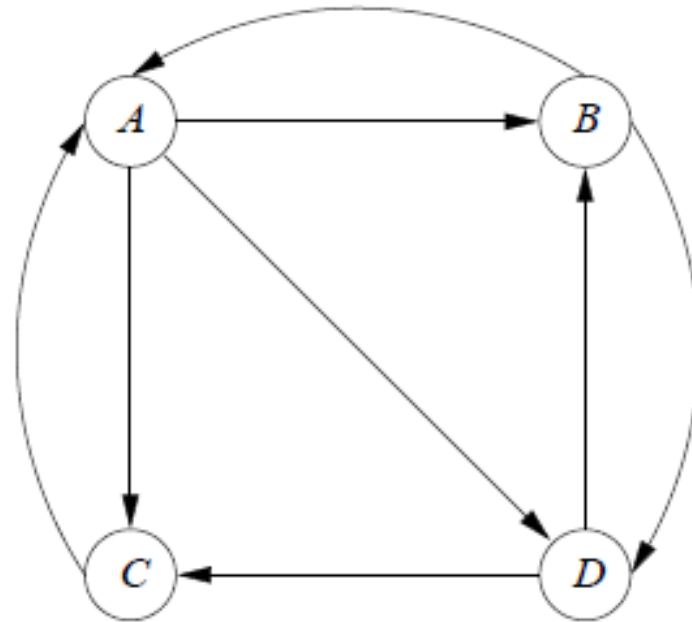
- $p(t)$  ... vector whose  $i^{\text{th}}$  coordinate is the prob. that the surfer is at page  $i$  at time  $t$
- So,  $p(t)$  is a probability distribution over pages  
-> rank vector.



# Recall: Transition Matrix

- ◆  $M[i,j]$  = prob. of going from node  $j$  to node  $i$ 
  - If  $j$  has  $k$  outgoing edges, prob. for each edge =  $1/k$

$$M = \begin{array}{c} \begin{array}{c} A \\ B \\ C \\ D \end{array} \begin{array}{cc|cc} A & B & C & D \\ \hline 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{array} \end{array}$$



# Prob. of Locations of Surfer

- ◆ Represented as a **column vector,  $v$**
- ◆ Initially, surfer can be at any page with equal probability

$$v^0 = \begin{matrix} A \\ B \\ C \\ D \end{matrix} \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$$

# Limiting Distribution

- ◆  $v^1 = Mv^0$

- ◆  $v^2 = Mv^1 (=M^2v^0)$

- ◆ ...

- ◆  $v^i = Mv^{i-1} (=M^i v^0)$

- ◆ ...

- ◆  $v = Mv$

(from some step  $k$  on, **v does not change any more**)

# Compute Next Distribution

◆  $v^1 = M v^0$

➤  $v_i^1 = \sum_{j=1}^n M_{ij} v_j^0$

|   | A   | B   | C | D   |  |
|---|-----|-----|---|-----|--|
| A | 0   | 1/2 | 1 | 0   | $\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$ |
| B | 1/3 | 0   | 0 | 1/2 |  |
| C | 1/3 | 0   | 0 | 1/2 |  |
| D | 1/3 | 1/2 | 0 | 0   |  |

◆ E.g.,  $v_0^1 = 0 * \frac{1}{4} + \frac{1}{2} * \frac{1}{4} + 1 * \frac{1}{4} + 0 * \frac{1}{4} = \frac{3}{8}$

➤ i.e., prob. at A is 3/8 (or 9/24) after step 1

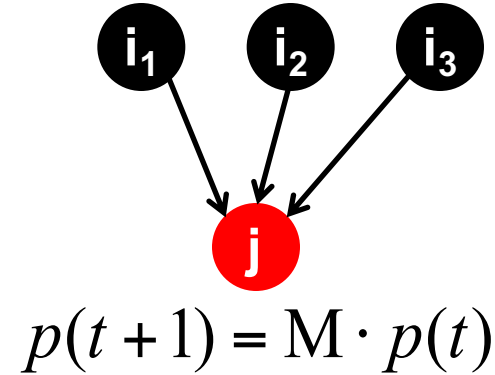
$$v^0 = \begin{matrix} A \\ B \\ C \\ D \end{matrix} \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} \quad v^1 = \begin{matrix} A \\ B \\ C \\ D \end{matrix} \begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}$$

# The Stationary Distribution

## ◆ Where is the surfer at time $t+1$ ?

- Follows a link uniformly at random

$$\mathbf{p}(t + 1) = \mathbf{M} \cdot \mathbf{p}(t)$$



- Suppose the random walk reaches a state

$$\mathbf{p}(t + 1) = \mathbf{M} \cdot \mathbf{p}(t) = \mathbf{p}(t)$$

then  $\mathbf{p}(t)$  is **stationary distribution** of a random walk

- **Our original rank vector  $\mathbf{r}$**  satisfies  $\mathbf{r} = \mathbf{M} \cdot \mathbf{r}$

- **So,  $\mathbf{r}$  is a stationary distribution for the random walk.**

# Existence and Uniqueness

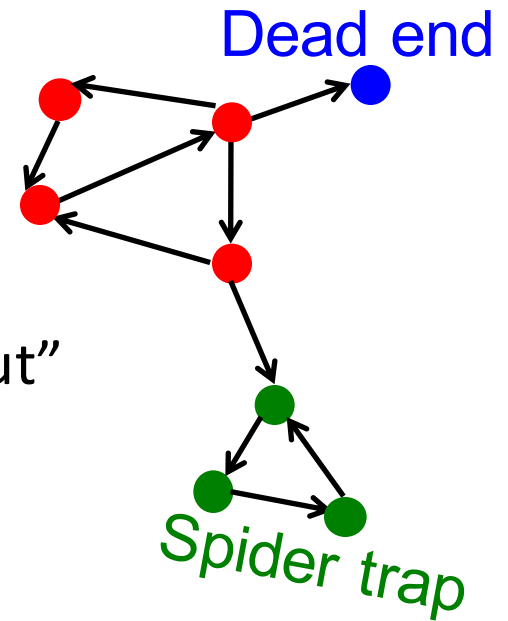
- ◆ A central result from the theory of random walks (a.k.a. Markov processes):

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  $t = 0$

# PageRank: Problems

## Two problems:

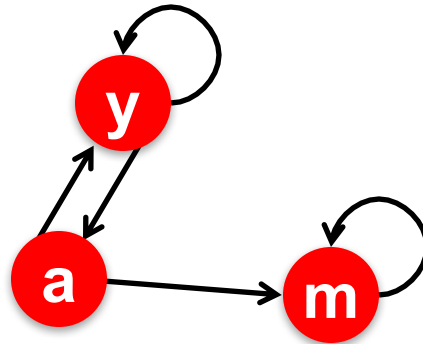
- ◆ (1) Some pages are **dead ends** (have no out-links)
  - Random walk has “nowhere” to go to
  - Such pages cause importance to “leak out”
- ◆ (2) **Spider traps:** (all out-links are within the group)
  - Random walked gets “stuck” in a trap
  - And eventually spider traps absorb all importance.





# Problem: Spider Traps

All outlinks  
are within  
a group



m is a spider trap

|   | y   | a   | m |
|---|-----|-----|---|
| y | 1/2 | 1/2 | 0 |
| a | 1/2 | 0   | 0 |
| m | 0   | 1/2 | 1 |

$$\mathbf{r}^{k+1} = \mathbf{M}\mathbf{r}^k$$

$$\mathbf{r}_y = \mathbf{r}_y/2 + \mathbf{r}_a/2$$

$$\mathbf{r}_a = \mathbf{r}_y/2$$

$$\mathbf{r}_m = \mathbf{r}_a/2 + \mathbf{r}_m$$

◆ Example:

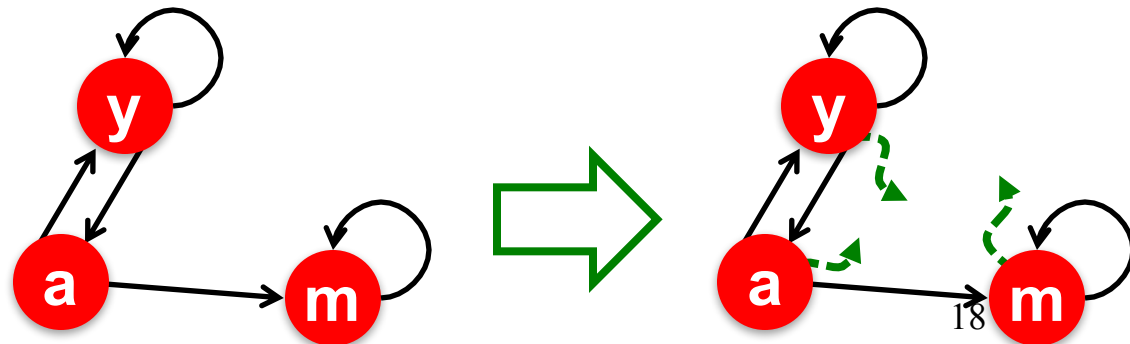
$$\begin{bmatrix} \mathbf{r}_y \\ \mathbf{r}_a \\ \mathbf{r}_m \end{bmatrix} = \begin{array}{ccccc} 1/3 & 2/6 & 3/12 & 5/24 & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 3/6 & 7/12 & 16/24 & & 1 \end{array}$$

Iteration 0, 1, 2, ...

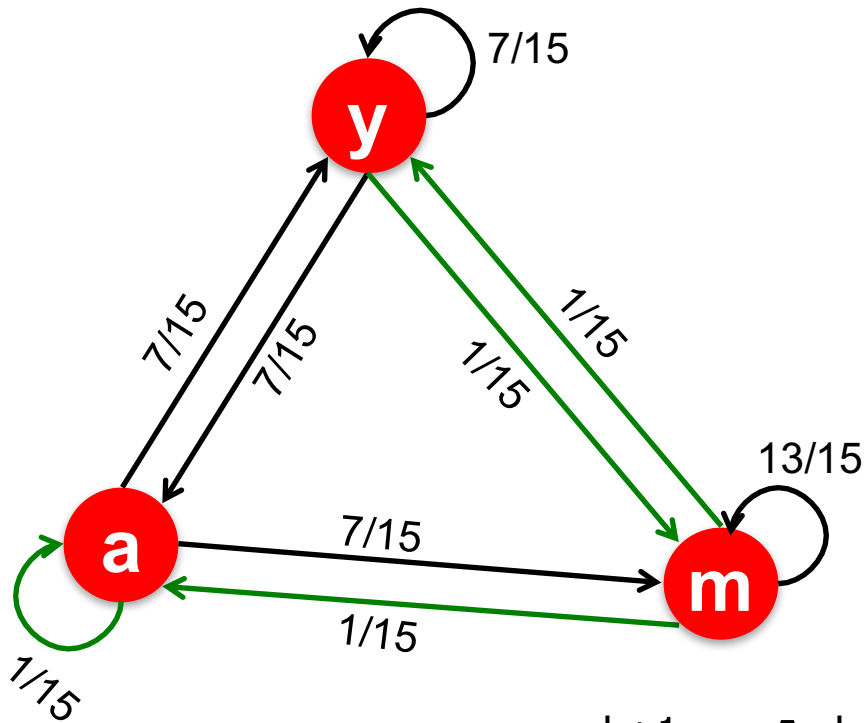
All the PageRank score gets “trapped” in node m <sup>17</sup>

# Solution: Teleports!

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



# Random Teleports ( $\beta = 0.8$ )



$$0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

|   | y    | a    | m     |
|---|------|------|-------|
| y | 7/15 | 7/15 | 1/15  |
| a | 7/15 | 1/15 | 1/15  |
| m | 1/15 | 7/15 | 13/15 |

$$\mathbf{r}^{k+1} = \mathbf{A} \mathbf{r}^k$$

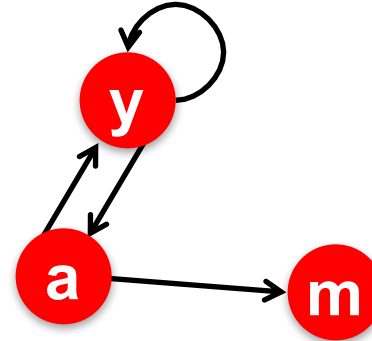
**A**

$$\begin{array}{l} \mathbf{r}_y \\ \mathbf{r}_a \\ \mathbf{r}_m \end{array} = \begin{array}{ccccc} 1/3 & 0.33 & 0.24 & 0.26 & 7/33 \\ 1/3 & 0.20 & 0.20 & 0.18 & 5/33 \\ 1/3 & 0.46 & 0.52 & 0.56 & 21/33 \end{array}$$

# Problem: Dead Ends

Pages with no outlinks are “dead ends” for the random surfer:

Nowhere to go on next step



|   | y   | a   | m |
|---|-----|-----|---|
| y | 1/2 | 1/2 | 0 |
| a | 1/2 | 0   | 0 |
| m | 0   | 1/2 | 0 |

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

$$r_a = r_y/2$$

$$r_m = r_a/2$$

$$\mathbf{r}^{k+1} = \mathbf{M}\mathbf{r}^k$$

◆ Example:

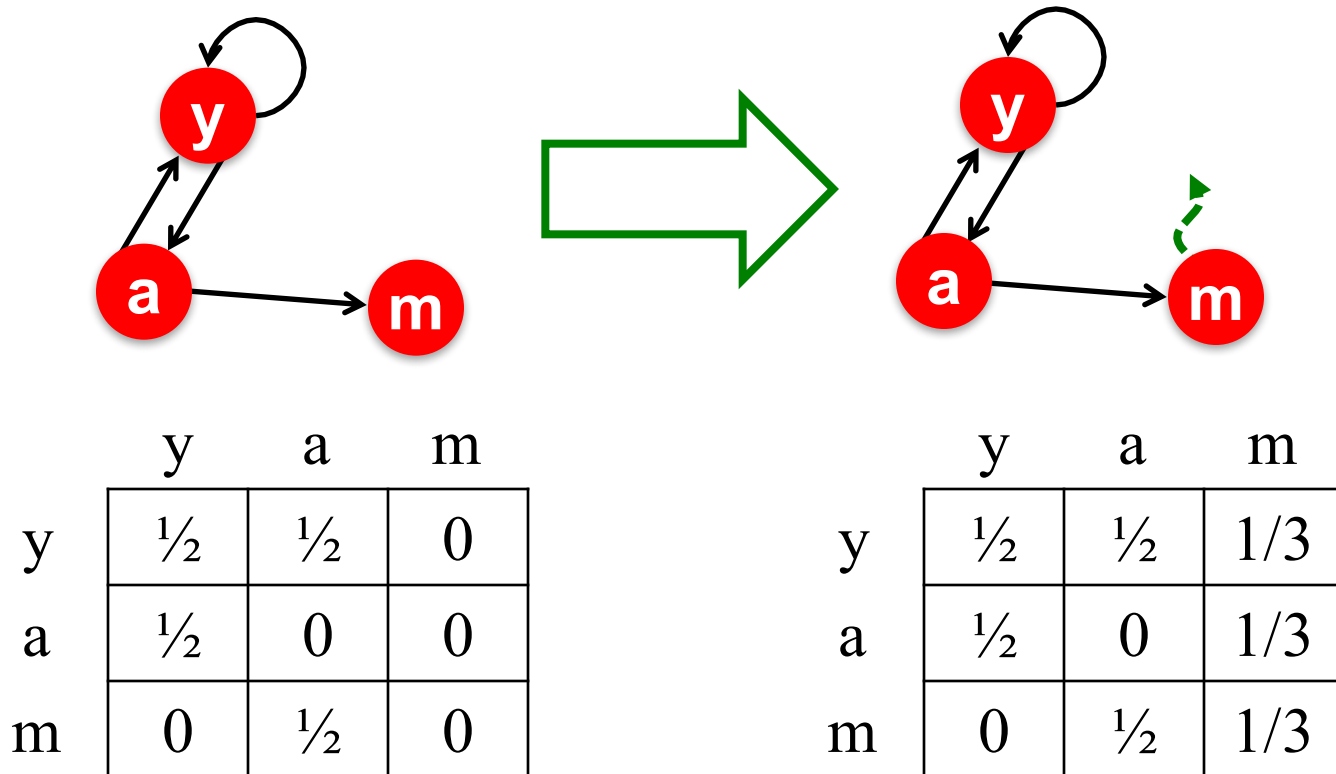
$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{bmatrix}$$

Here the PageRank “leaks” out since the matrix is not column stochastic (not all columns total to 1)

# Solution 1 for Dead Ends: Always Teleport!

- ◆ **Teleports:** Follow random teleport links with probability 1.0 from dead-ends

➤ Adjust matrix accordingly



# Why Do Teleports Solve the Problem?

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

## ◆ Spider-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

- The matrix is **not column stochastic** so our initial assumptions are not met
- **Solution:** Make matrix column stochastic by always teleporting when there is nowhere else to go.

# How is Page Rank Information Used in a Search Engine?

- ◆ Each search engine has a **secret formula** that **decides the order in which to show pages** to a user in response to a search query
  - Google: thought to use over 250 different properties of pages
- ◆ **Search engine decides on a linear order of search results**
- ◆ To be considered for ranking, **page has to have at least one of the query search terms**
  - Unless all search terms are present, little chance of being in top ten results
- ◆ **Among qualified pages, score computed for each**
  - **PageRank** is an important component of the score
  - **Other components:** presence/absence of search terms in headers, links to the page.

# Sparse Matrix Encoding Example

**Example 5.7:** Let us reprise the example Web graph from Fig. 5.1, whose transition matrix is

$$M = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

Recall that the rows and columns represent nodes  $A$ ,  $B$ ,  $C$ , and  $D$ , in that order. In Fig. 5.11 is a compact representation of this matrix.<sup>5</sup>

| Source | Degree | Destinations |
|--------|--------|--------------|
| $A$    | 3      | $B, C, D$    |
| $B$    | 2      | $A, D$       |
| $C$    | 1      | $A$          |
| $D$    | 2      | $B, C$       |

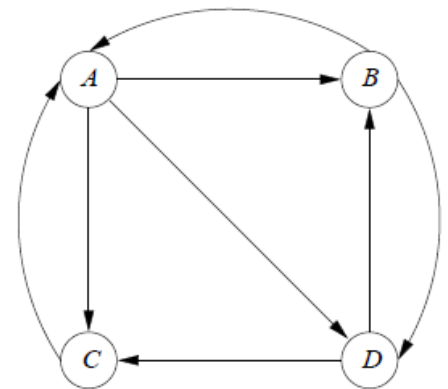


Figure 5.11: Represent a transition matrix by the out-degree of each node and the list of its successors



# PageRang Iteration using MapReduce

- ◆ If  $n$  is small enough that each Map task can store the full vector  $v$  and  $v'$  in main memory
  - With additional steps to multiply each component of  $Mv$  by constant  $\beta$  and to add  $(1 - \beta)/n$
- ◆ Given the size of the Web,  $v$  is much too large to fit in main memory
- ◆ Break  $M$  into vertical stripes (chp. 2.3.1) and break  $v$  into corresponding horizontal stripes to execute the MapReduce.

# Matrix-Vector Multiplication by MapReduce

- ◆  $n \times n$  matrix  $M$ , vector  $v$  of length  $n$
- ◆  $M \times V = X$
- ◆ the vector  $x$  of length  $n$ , whose  $i_{\text{th}}$  element  $x_i$  is given by

$$x_i = \sum_{j=1}^n m_{ij} v_j$$

# Use of Combiners to Consolidate the Result Vector

- ◆ Partition the vector into  $k$  stripes and matrix into  $k^2$  blocks
- ◆ E.g.,  $k=4$

$$\begin{bmatrix} \mathbf{v}'_1 \\ \mathbf{v}'_2 \\ \mathbf{v}'_3 \\ \mathbf{v}'_4 \end{bmatrix} = \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \mathbf{v}_4 \end{bmatrix}$$

# Use of Combiners to Consolidate the Result Vector (Cont'd)

- ◆ Use  $k^2$  Map tasks
- ◆ Each task gets **one square of the matrix  $M$  ( $M_{ij}$ )** and **one stripe of the vector  $v$  ( $v_j$ )**
  - each stripe of the vector is sent to  $k$  different Map tasks
  - $v_j$  is sent to the task handling  $M_{ij}$  for each of the  $k$  possible values of  $i$
- ◆ Advantage: keep both the  $j_{th}$  stripe of  $v$  and the  $i_{th}$  stripe of  $v'$  in main memory as we process  $M_{ij}$

|        |
|--------|
| $v'_1$ |
| $v'_2$ |
| $v'_3$ |
| $v'_4$ |

=

|          |          |          |          |
|----------|----------|----------|----------|
| $M_{11}$ | $M_{12}$ | $M_{13}$ | $M_{14}$ |
| $M_{21}$ | $M_{22}$ | $M_{23}$ | $M_{24}$ |
| $M_{31}$ | $M_{32}$ | $M_{33}$ | $M_{34}$ |
| $M_{41}$ | $M_{42}$ | $M_{43}$ | $M_{44}$ |

|       |
|-------|
| $v_1$ |
| $v_2$ |
| $v_3$ |
| $v_4$ |

Note that all terms generated from  $M_{ij}$  and  $v_j$  contribute to  $v'_i$  and no other stripe of  $v'$

# PageRank Iteration Using MapReduce

- ◆ Assume rank vector  $\mathbf{r}$  does not fit in main memory
- ◆ **Block stripe update method**: break  $\mathbf{M}$  into vertical stripes and break  $\mathbf{r}$  into corresponding horizontal stripes
- ◆ **Allows us to execute MapReduce efficiently**
- ◆ Process no more of  $\mathbf{r}$  at any one Map task than can fit in main memory
- ◆ **Trying to use a combiner gets more complicated**: Section 5.2.3
  - Partition matrix  $\mathbf{M}$  into  $k^2$  blocks, vector  $\mathbf{r}$  in  $k$  stripes
  - Use  $k^2$  Map tasks
  - Each Map task gets: **one square of matrix:  $M_{ij}$** , **one stripe of vector,  $r_j$**
  - **Each stripe  $r_j$  sent to  $k$  Map tasks**, transmitted over network  $k$  times
  - **But each  $M_{ij}$  transmitted only once.**

# Some Problems with Page Rank

## ◆ Measures generic popularity of a page

- Biased against topic-specific authorities
- **Solution:** Topic-Specific PageRank

## ◆ Susceptible to Link spam

- Artificial link topographies created in order to boost page rank
- **Solution:** TrustRank

## ◆ Uses a single measure of importance

- Other models of importance
- **Solution:** Hubs-and-Authorities.

# Topic-Specific PageRank

# Topic-Specific PageRank

- ◆ **Instead of generic popularity, can we measure popularity within a topic?**
- ◆ **Goal:** Evaluate Web pages not just according to their popularity, but by how close they are to a particular topic, e.g. “sports” or “history”
- ◆ **Allows search queries to be answered based on interests of the user**
  - **Example:** Query “Trojan” wants different pages depending on whether you are interested in sports, history and computer security.



# Topic-Specific PageRank

- ◆ Random walker has a small probability of teleporting at any step
- ◆ **Where a Teleport can go:**
  - **Standard PageRank:** Any page with equal probability
    - To avoid dead-end and spider-trap problems
  - **Topic Specific PageRank:** A topic-specific set of “relevant” pages (teleport set)
- ◆ **Idea: Bias the random walk**
  - When walker teleports, she picks a page from a set  $S$
  - $S$  contains only pages that are relevant to the topic
    - E.g., Open Directory (DMOZ) pages for a given topic/query
  - For each teleport set  $S$ , we get a different vector  $r_S$

# Discovering the Topic PageRank Vectors $r_s$ for Topic Set $S$

- ◆ If the search engine can deduce the user's interests, then it can do a better job of returning relevant pages
  - Ideally, would like to have a private PageRank vector that gives importance of each page to that user (not feasible!)
- ◆ Topic-Sensitive PageRank: creates one vector for some small number of topics
  - Bias PageRank to favor pages of that topic
- ◆ Create different PageRank vectors for different topics
  - The 16 top-level categories of the Open Directory (DMOZ): arts, business, sports,...
- ◆ If we can determine that the user is interested in one of these topics (e.g., by content of pages they recently viewed), use the PageRank vector for that topic when deciding on ranking of pages for query results.

# Biased Random Walk

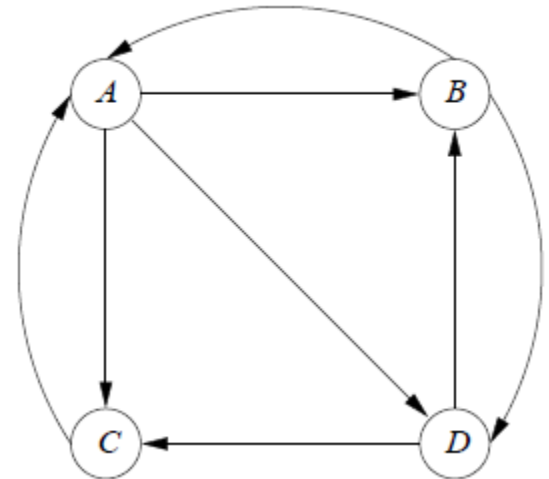
- ◆ Bias toward a set  $S$  of pages
  - $S$ : called teleport set
  - $\mathbf{e}_S$ : a vector with 1 for pages in  $S$ , 0 otherwise

$$\mathbf{v}' = \beta M \mathbf{v} + (1 - \beta) \mathbf{e}_S / |S|$$

# Example

- ◆ Teleport set  $S = \{B, D\}$ ,  $\beta = .8$
- ◆  $e_S$ : a vector with 1 for pages in  $S$ , 0 otherwise

$$M = \begin{matrix} & \begin{matrix} A & B & C & D \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \end{matrix} & \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \end{matrix}$$



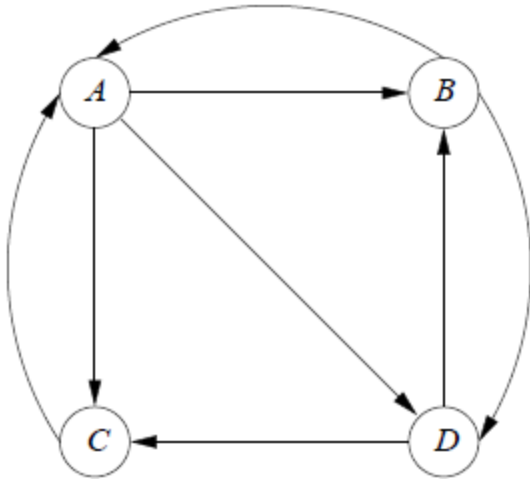
$$\mathbf{v}' = \beta M \mathbf{v} + (1 - \beta) \mathbf{e}_S / |S|$$

$$= 0.8 M \mathbf{v} + 0.2 \mathbf{e}_S / 2$$

$$\mathbf{v}' = \begin{bmatrix} 0 & 2/5 & 4/5 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix} \mathbf{v} + \begin{bmatrix} 0 \\ 1/10 \\ 0 \\ 1/10 \end{bmatrix}$$

2/10 \* 1/2

# Example



$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} \quad \begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix} \quad \begin{bmatrix} 15/48 \\ 11/48 \\ 11/48 \\ 11/48 \end{bmatrix} \quad \begin{bmatrix} 11/32 \\ 7/32 \\ 7/32 \\ 7/32 \end{bmatrix} \quad \cdots \quad \begin{bmatrix} 3/9 \\ 2/9 \\ 2/9 \\ 2/9 \end{bmatrix}$$

Unbiased: A has the largest

$$\begin{bmatrix} 0/2 \\ 1/2 \\ 0/2 \\ 1/2 \end{bmatrix}, \begin{bmatrix} 2/10 \\ 3/10 \\ 2/10 \\ 3/10 \end{bmatrix}, \begin{bmatrix} 42/150 \\ 41/150 \\ 26/150 \\ 41/150 \end{bmatrix}, \begin{bmatrix} 62/250 \\ 71/250 \\ 46/250 \\ 71/250 \end{bmatrix}, \cdots, \begin{bmatrix} 54/210 \\ 59/210 \\ 38/210 \\ 59/210 \end{bmatrix}$$

Biased: B and D has largest PageRanks

# Integrating Topic-Sensitive PageRank Into a Search Engine

1. **Decide on topics** for which to create specialized PageRank vectors
2. **Pick a teleport set for each of these topics**
  - Use that set to **compute topic-sensitive PageRank for that topic**
3. **Determine the topic or set of topics most relevant to a particular search query**
  - More on this in next slide
4. **Use the PageRank vector for those topic(s) to order the responses to the search query.**

# How to Select the Topic Set that is Most Relevant to the Search Query?

## ◆ Tricky

## ◆ Several methods proposed:

1. User can pick from a topic menu
2. Infer the topic(s) from the context of the query
  - Words that appear in Web pages recently searched by user or recent user queries
  - E.g., query is launched from a web page talking about a known topic
    - History of queries: e.g., “basketball” followed by “Jordan”
  - Classification of documents by topic: studied for decades
3. Infer the topic(s) from information about the user
  - E.g., the user’s bookmarks, their interest on Facebook, etc.

# **TrustRank: Combatting Web Spam**



# What is Web Spam?

## ◆ Spamming:

- Any deliberate action to boost a web page's position in search engine results, not commensurate with page's real value

## ◆ Spam:

- Web pages that are the result of spamming

## ◆ This is a very broad definition

- **SEO** industry might disagree!
- SEO = search engine optimization

## ◆ Approximately **10-15%** of web pages are spam

# Web Search

## ◆ Early search engines:

- Crawl the Web
- Index pages by the words they contained
- Respond to **search queries** (lists of words) with the pages containing those words

## ◆ Early page ranking:

- Attempt to order pages matching a search query by “importance”
- **First search engines considered:**
  - **(1)** Number of times query words appeared
  - **(2)** Prominence of word position, e.g. title, header.

# First Spammers

- ◆ As people began to use search engines to find things on the Web, those with commercial interests tried to **exploit search engines** to bring people to their own site – whether they wanted to be there or not
- ◆ **Example:**
  - Shirt-seller might pretend to be about “movies”
- ◆ **Techniques for achieving high relevance/importance for a web page.**

# First Spammers: Term Spam

- ◆ How do you make your page **appear** to be about movies?
- ◆ **(1) Add the word movie 1,000 times to your page**
  - Set text color to the background color, so only search engines would see it
- ◆ **(2) Or, run the query “movie” on your target search engine**
  - See what page came first in the listings
  - **Copy it into your page, make it “invisible”**
- ◆ These and similar techniques are called **term spam**.

# Google's Solution to Term Spam

- ◆ **Believe what people say about you, rather than what you say about yourself**
  - Use words in the **anchor text (words that appear underlined to represent the link)** and its surrounding text
- ◆ **PageRank** as a tool to measure the “importance” of Web pages
  - **Doesn't depend on content of pages.**


# Why It Works?

## ◆ Our hypothetical shirt-seller loses

- Saying he is about movies doesn't help, because **others don't say he is about movies**
- His page isn't very important, so it **won't be ranked high for shirts or movies**

## ◆ Example:

- Shirt-seller creates 1,000 pages, each links to his with "movie" in the anchor text
- **These pages have no links in, so they get little PageRank**
- So the shirt-seller **can't beat truly important movie pages, like IMDB (Internet Movie Database).**



# SPAM FARMING

# Google vs. Spammers: Round 2!

- ◆ Once Google became the dominant search engine, spammers began to work out ways to fool Google
- ◆ **Spam farms** were developed to concentrate PageRank on a single page
- ◆ **Link spam:**
  - Creating link structures that boost PageRank of a particular page





# Link Spamming

## ◆ Three kinds of web pages from a spammer's point of view

### ➤ Inaccessible pages

### ➤ Accessible pages

- e.g., blog comments pages
- spammer can post links to his pages

### ➤ Owned pages

- Completely controlled by spammer
- May span multiple domain names.

# Link Farms

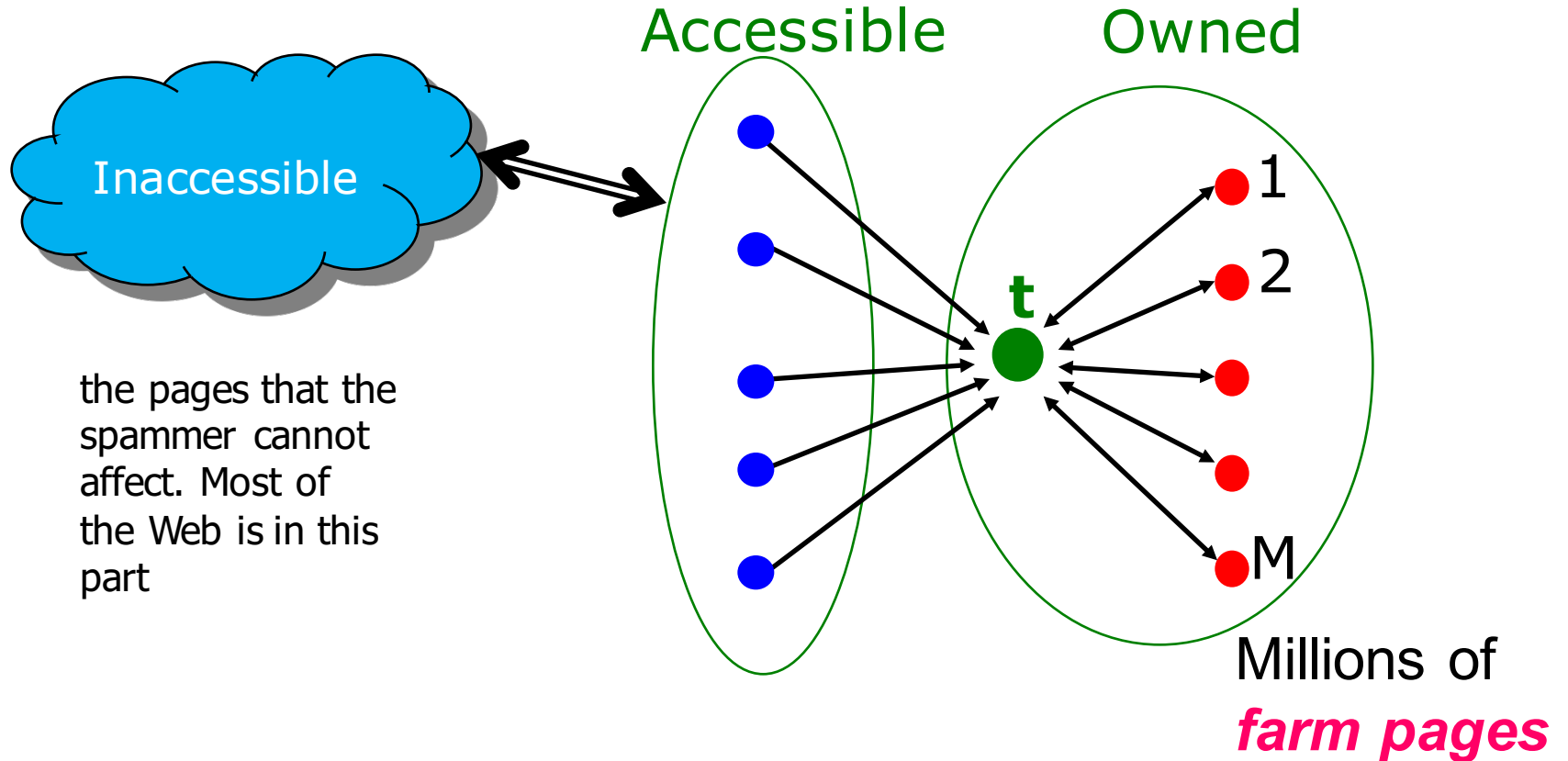
## ◆ Spammer's goal:

- Maximize the PageRank of target page  $t$

## ◆ Technique:

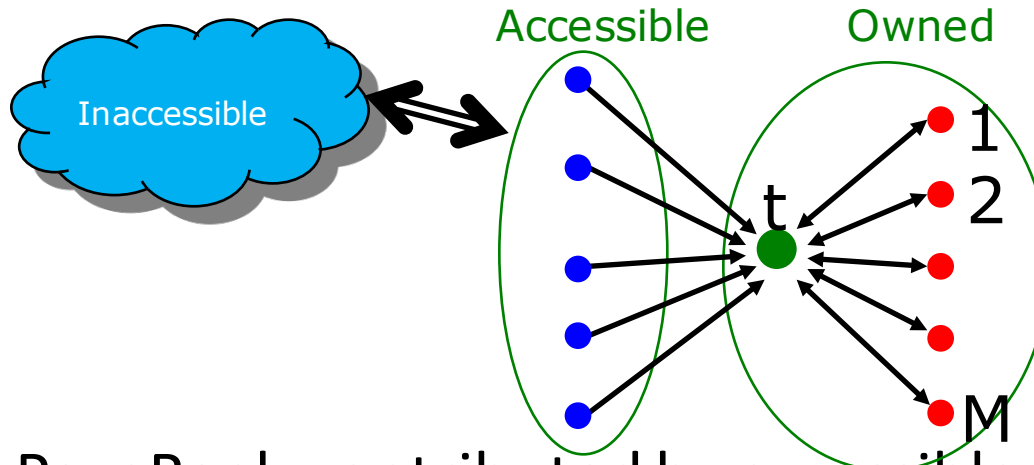
- Get as many links from accessible pages as possible to target page  $t$
- Construct “link farm” to get PageRank multiplier effect.

# Link Farms



**One of the most common and effective organizations for a link farm**

# Analysis



◆  $x$ : PageRank contributed by accessible pages

◆  $y$ : PageRank of target page  $t$

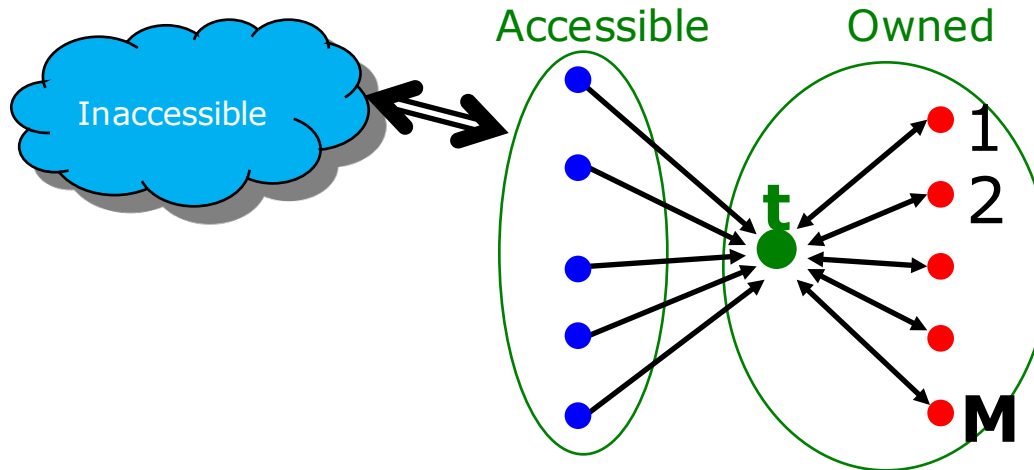
◆ Rank of each “farm” page =  $\frac{\beta y}{M} + \frac{1-\beta}{N}$

$$\begin{aligned} \text{◆ } y &= x + \beta M \left[ \frac{\beta y}{M} + \frac{1-\beta}{N} \right] + \frac{1-\beta}{N} \\ &= x + \beta^2 y + \frac{\beta(1-\beta)M}{N} + \frac{1-\beta}{N} \end{aligned}$$

Very small; ignore  
Now we solve for  $y$

$$\text{◆ } y = \frac{x}{1-\beta^2} + c \frac{M}{N} \quad \text{where } c = \frac{\beta}{1+\beta}$$

# Analysis



N...# pages on the web  
M...# of pages spammer owns

◆  $y = \frac{x}{1-\beta^2} + c \frac{M}{N}$  where  $c = \frac{\beta}{1+\beta}$

◆ For  $\beta = 0.85$ ,  $1/(1-\beta^2) = 3.6$

◆ Multiplier effect for acquired PageRank

◆ By making **M** large, spammer can make **y** as large as they want.

# **TrustRank:**

## **Combating the Web Spam**

# Combating Spam

## ◆ Combating term spam

- Analyze text using statistical methods
- Similar to email spam filtering
- Also useful: Detecting approximate duplicate pages

## ◆ Combating link spam

- **Detection and blacklisting of structures that look like spam farms**
  - Leads to another war – hiding and detecting spam farms
- **TrustRank** = topic-specific PageRank with a teleport set of **trusted pages**
  - **Example:** .edu domains, similar domains for non-US schools.

# TrustRank: Idea

- ◆ **Basic principle: Approximate isolation**

- It is rare for a “good” page to point to a “bad” (spam) page

- ◆ Sample a set of **seed pages** from the web

- ◆ Have an **oracle (human)** to identify the good pages and the spam pages in the seed set

- **Expensive task**, so we must make seed set as small as possible.



# Trust Propagation

- ◆ Call the subset of seed pages that are identified as **good** the **trusted pages**
- ◆ Perform a topic-sensitive PageRank with **teleport set = trusted pages**
  - **Propagate trust through links:**
    - Each page gets a trust value between **0** and **1**
- ◆ **Solution 1:** Use a threshold value and mark all pages below the trust threshold as spam.

# Why is it a good idea?

## ◆ Trust attenuation:

- The degree of trust conferred by a trusted page decreases with the distance in the graph

## ◆ Trust splitting:

- The larger the number of out-links from a page, the less scrutiny the page author gives each out-link
- Trust is **split** across out-links.

# Picking the Seed Set

## ◆ Two conflicting considerations:

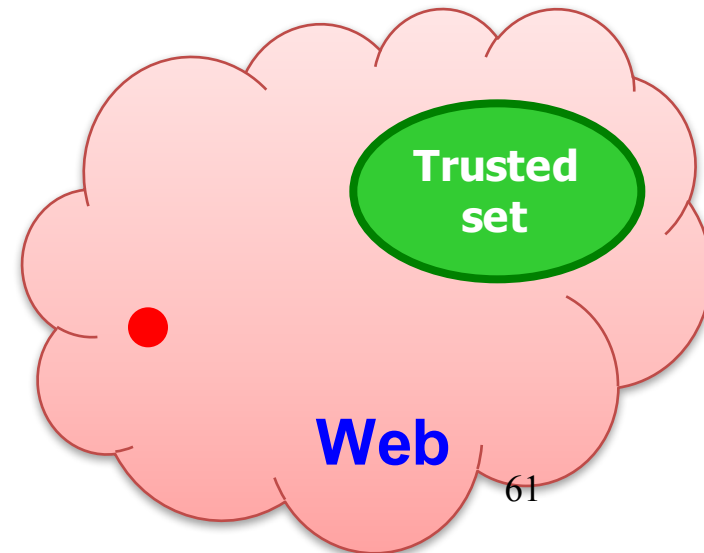
- Human has to inspect each seed page, so seed set must be as small as possible
- Must ensure every **good page** gets adequate trust rank, so need make all good pages reachable from seed set by short paths.

# Approaches to Picking Seed Set

- ◆ Suppose we want to pick a seed set of  $k$  pages
- ◆ **How to do that?**
- ◆ **(1) PageRank:**
  - Pick the top  $k$  pages by PageRank
  - Theory is that you can't get a bad page's rank really high
- ◆ **(2) Use trusted domains** whose membership is controlled, like .edu, .mil, .gov

# Spam Mass

- ◆ In the **TrustRank** model, we start with good pages and propagate trust
- ◆ **Complementary view:**  
What fraction of a page's PageRank comes from **spam** pages?
- ◆ In practice, we don't know all the spam pages, so we need to estimate.



# Spam Mass Estimation

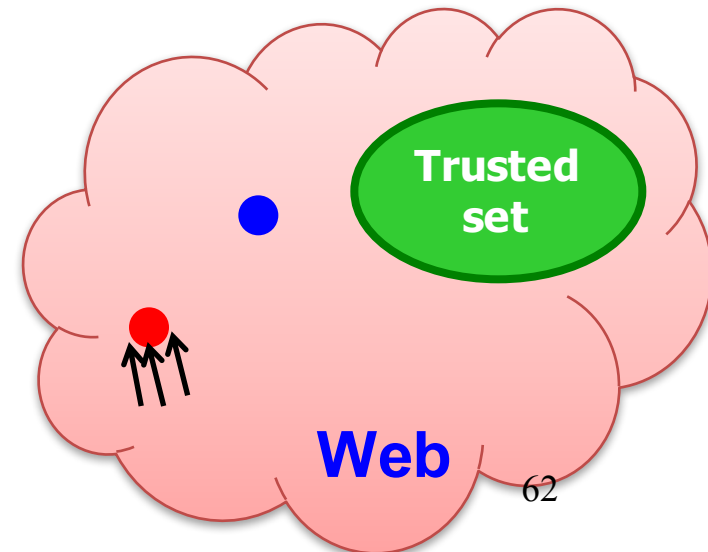
## Solution 2:

- ◆  $r_p$  = PageRank of page  $p$
- ◆  $r_p^+$  = PageRank of  $p$  with teleport into **trusted** pages only
- ◆ **Then:** What fraction of a page's PageRank comes from **spam** pages?

$$r_p^- = r_p - r_p^+$$

- ◆ **Spam mass of  $p$**   $= \frac{r_p^-}{r_p}$

- Pages with high spam mass are spam.



# **HITS: Hubs and Authorities**

# TrustRank

- ◆ A variant of topic-sensitive PageRank
  - Topic = a set of trustworthy pages
- ◆ Identify trustworthy pages
  - Manually examine pages with high PageRanks
  - Pick pages within controlled domains: edu, gov, mil
  - Avoid borderline pages, e.g., blog sites, forums.

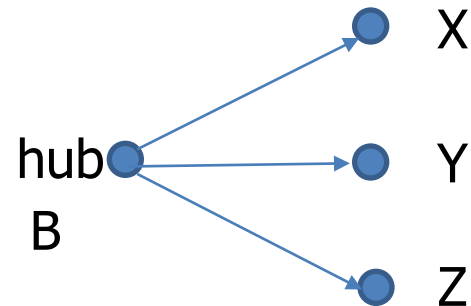
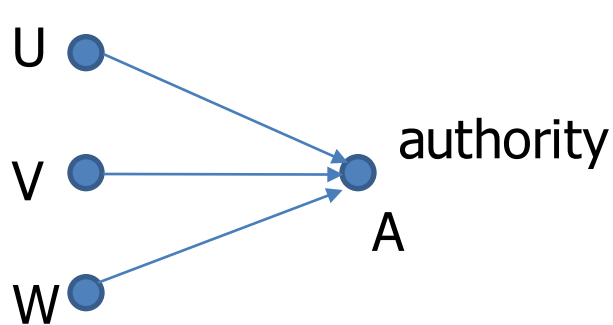


# HITS Algorithm

- ◆ Hyperlink-induced topic search
- ◆ Each page has two scores
  - Authority: how important the page is about topic?
  - Hub: does it provide many pointers to the topic?
- ◆ Examples
  - Hub: department pages listing all courses
  - Authority: individual course pages.

# Authority and Hub

- ◆ A good authority page is one
  - **pointed** by many good hub pages



- ◆ A good hub page is one
  - **pointing to** many good authority pages.

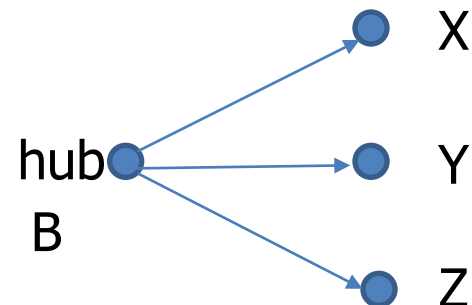
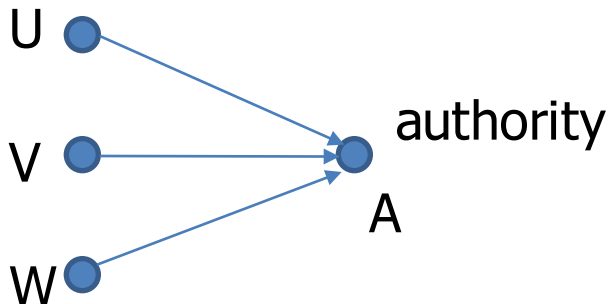
# Computing Authority and Hubbiness

## ◆ Authority of a page =

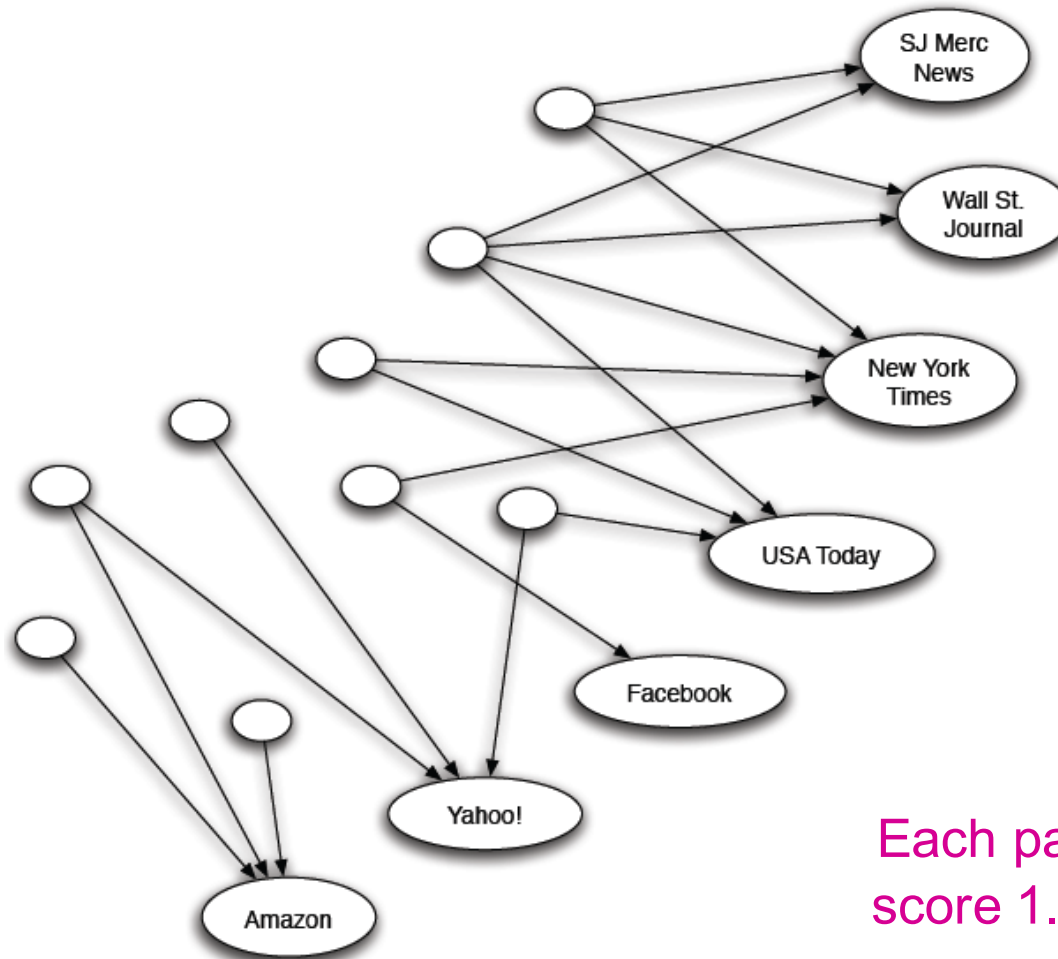
- sum of hubbiness of pages pointing to it
- $a(A) = h(U) + h(V) + h(W)$

## ◆ Hubbiness of a page =

- sum of authority of pages it points to
- $h(B) = a(X) + a(Y) + a(Z)$



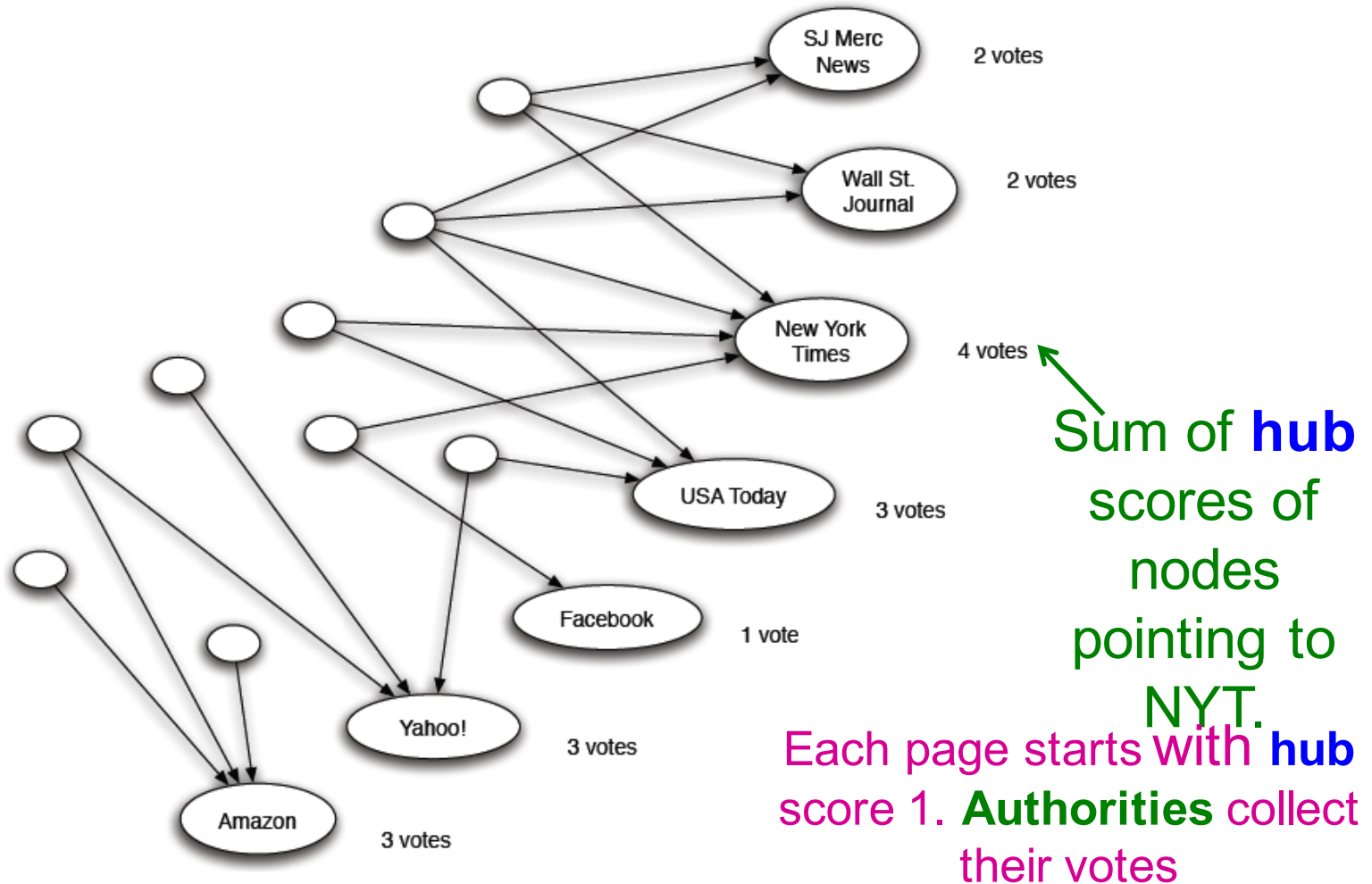
# Counting in-links: Authority



Each page starts with **hub** score 1. **Authorities** collect their votes

(Note this is idealized example. In reality graph is not bipartite and each page has both the hub and authority score)

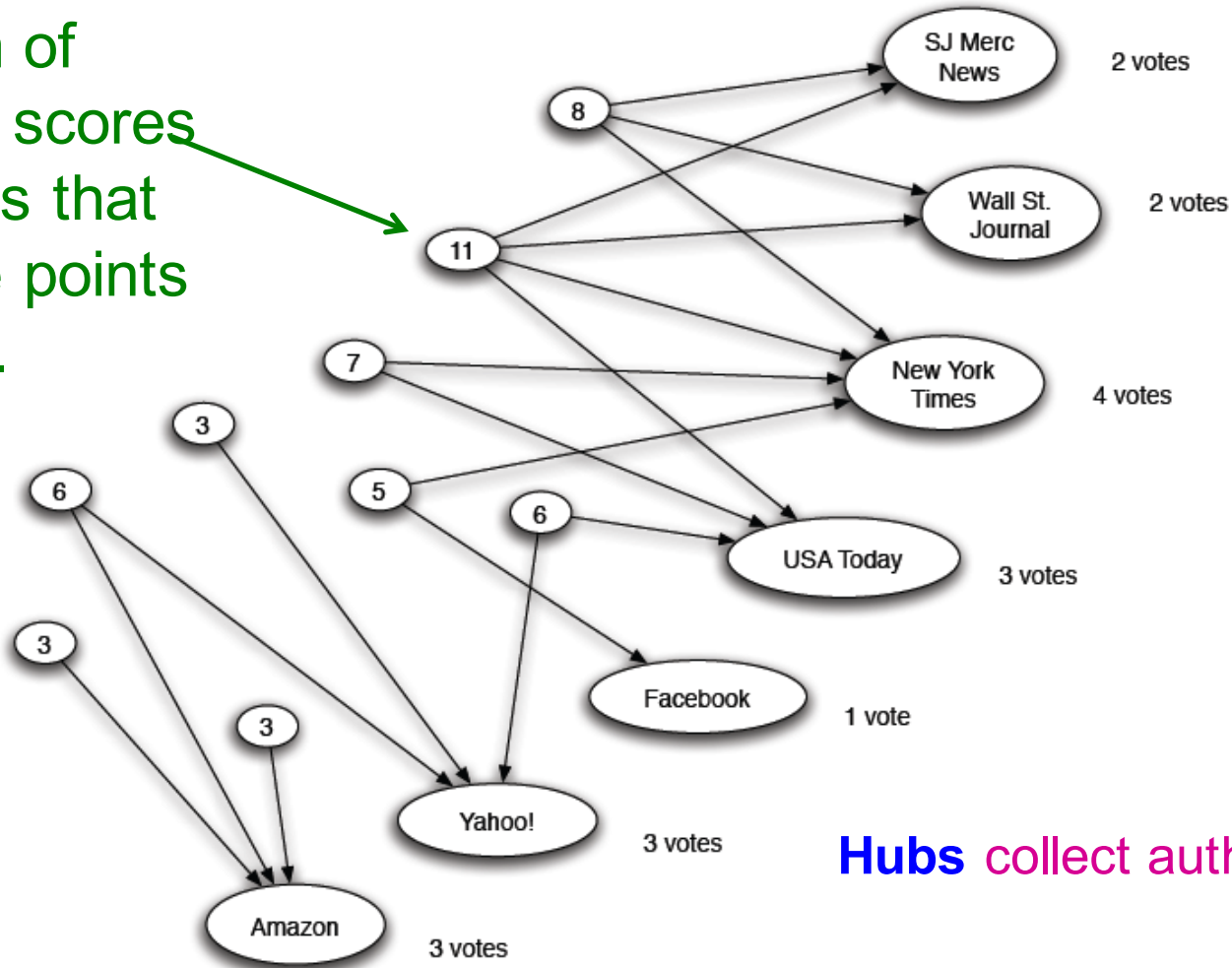
# Counting in-links: Authority



(Note this is idealized example. In reality graph is not bipartite and each page has both the hub and authority score)

# Expert Quality: Hub

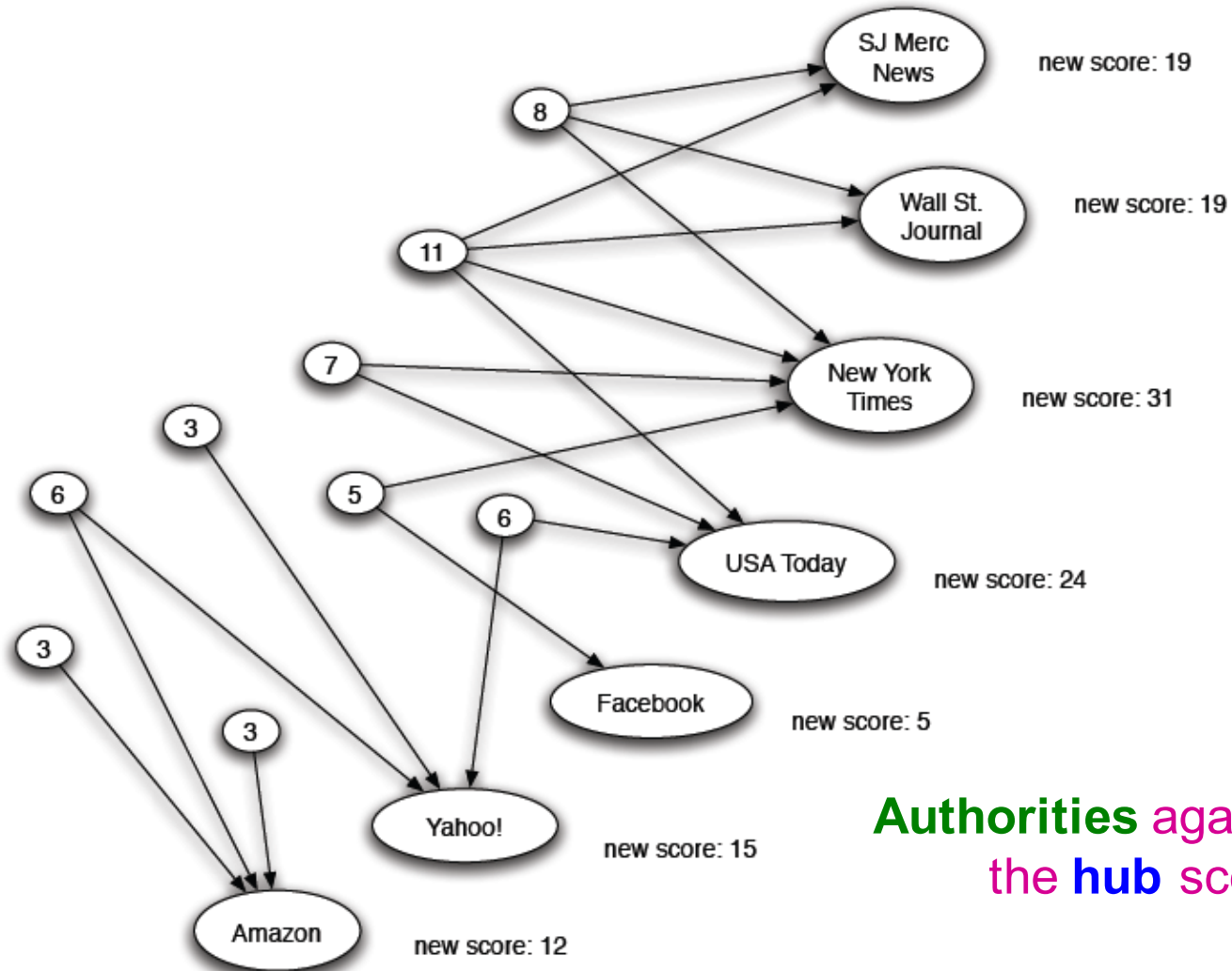
Sum of  
authority scores  
of nodes that  
the node points  
to.



**Hubs** collect authority scores

(Note this is idealized example. In reality graph is not bipartite and each page has both the hub and authority score)

# Reweighting



**Authorities** again collect  
the **hub** scores

(Note this is idealized example. In reality graph is not bipartite and each page has both the hub and authority score)

# Mutually Recursive Definition

- ◆ A good hub links to many good authorities
- ◆ A good authority is linked from many good hubs
- ◆ Model using two scores for each node:
  - Hub score and Authority score
  - Represented as vectors  $\mathbf{h}$  and  $\mathbf{a}$ .



# Hubs and Authorities

## ◆ Each page $i$ has 2 scores:

- Authority score:  $a_i$
- Hub score:  $h_i$

## HITS algorithm:

◆ Initialize:  $a_j^{(0)} = 1/\sqrt{N}$ ,  $h_j^{(0)} = 1/\sqrt{N}$

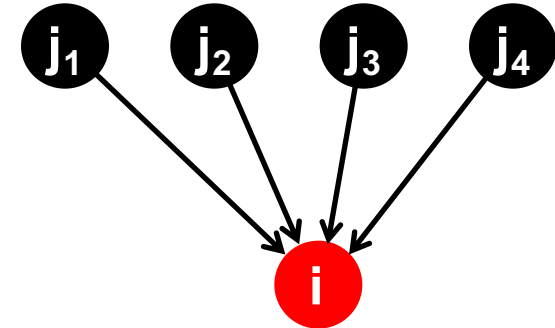
◆ Then keep iterating until **convergence**:

➤  $\forall i$ : Authority:  $a_i^{(t+1)} = \sum_{j \rightarrow i} h_j^{(t)}$

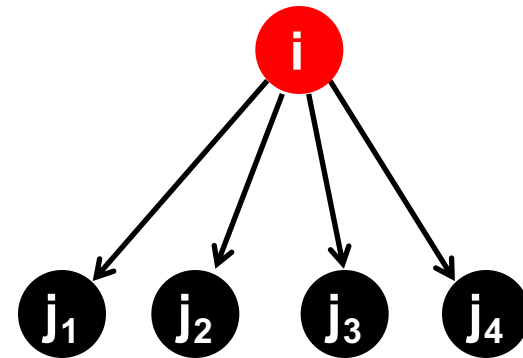
➤  $\forall i$ : Hub:  $h_i^{(t+1)} = \sum_{i \rightarrow j} a_j^{(t)}$

➤  $\forall i$ : Normalize:

$$\sum_i \left(a_i^{(t+1)}\right)^2 = 1, \sum_j \left(h_j^{(t+1)}\right)^2 = 1$$



$$a_i = \sum_{j \rightarrow i} h_j$$



$$h_i = \sum_{i \rightarrow j} a_j$$

# Hubs and Authorities

## ◆ HITS converges to a single stable point

## ◆ Notation:

➤ Vector  $\mathbf{a} = (a_1 \dots, a_n)$ ,  $\mathbf{h} = (h_1 \dots, h_n)$

➤ Adjacency matrix  $\mathbf{A}$  ( $N \times N$ ):  $A_{ij} = 1$  if  $i \rightarrow j$ , 0 otherwise

◆ Then  $h_i = \sum_{i \rightarrow j} a_j$

can be rewritten as  $h_i = \sum_j A_{ij} \cdot a_j$

So:  $\mathbf{h} = \mathbf{A} \cdot \mathbf{a}$

◆ Similarly,  $a_i = \sum_{j \rightarrow i} h_j$

can be rewritten as  $a_i = \sum_j A_{ji} \cdot h_j = \mathbf{A}^T \cdot \mathbf{h}$

↑  
Transpose

# Hubs and Authorities

## ◆ HITS algorithm in vector notation:

➤ Set:  $a_i = h_i = \frac{1}{\sqrt{n}}$

Repeat until convergence:

➤  $h = A \cdot a$

➤  $a = A^T \cdot h$

➤ Normalize  $a$  and  $h$

◆ Then:  $a = A^T \cdot (A \cdot a)$

$\underbrace{\underbrace{\text{new } h}_{\text{new } a}}$

**Convergence criterion:**

$$\sum_i \left( h_i^{(t)} - h_i^{(t-1)} \right)^2 < \varepsilon$$
$$\sum_i \left( a_i^{(t)} - a_i^{(t-1)} \right)^2 < \varepsilon$$

**$a$  is updated (in 2 steps):**

$$a = A^T (A a) = (A^T A) a$$

**$h$  is updated (in 2 steps):**

$$h = A (A^T h) = (A A^T) h$$

Repeated matrix powering

# Existence and Uniqueness

◆  $h = \lambda A a$

$$\lambda = 1 / \sum h_i$$

◆  $a = \mu A^T h$

$$\mu = 1 / \sum a_i$$

◆  $h = \lambda \mu A A^T h$

◆  $a = \lambda \mu A^T A a$

◆ Under reasonable assumptions about  $A$ ,  
HITS **converges to vectors  $h^*$  and  $a^*$** :

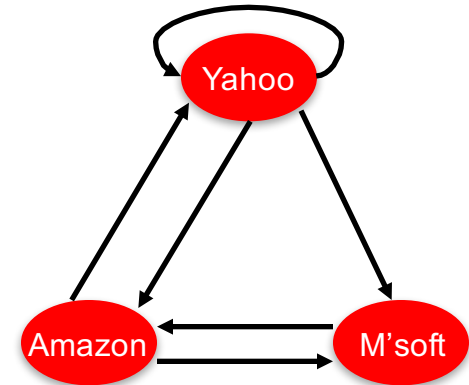
➤  $h^*$  is the **principal eigenvector** of matrix  $A A^T$

➤  $a^*$  is the **principal eigenvector** of matrix  $A^T A$

# Example of HITS

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$



$$h(\text{yahoo}) = \begin{matrix} .58 & .80 & .80 & .79 & \dots & \mathbf{.788} \end{matrix}$$

$$h(\text{amazon}) = \begin{matrix} .58 & .53 & .53 & .57 & \dots & .577 \end{matrix}$$

$$h(\text{m'soft}) = \begin{matrix} .58 & .27 & .27 & .23 & \dots & .211 \end{matrix}$$

$$a(\text{yahoo}) = \begin{matrix} .58 & .58 & .62 & .62 & \dots & .628 \end{matrix}$$

$$a(\text{amazon}) = \begin{matrix} .58 & .58 & .49 & .49 & \dots & .459 \end{matrix}$$

$$a(\text{m'soft}) = \begin{matrix} .58 & .58 & .62 & .62 & \dots & \mathbf{.628} \end{matrix}$$

# PageRank and HITS

- ◆ PageRank and HITS are two solutions to the same problem:
  - What is the value of an in-link from  $u$  to  $v$ ?
  - In the PageRank model, the value of the link depends on the links into  $u$
  - In the HITS model, it depends on the value of the other links out of  $u$
- ◆ The destinies of PageRank and HITS post-1998 were very different.

# Summary

## ◆ PageRank

- Power Iteration method

## ◆ Measures generic popularity of a page

- Biased against topic-specific authorities
- **Solution:** Topic-Specific PageRank

## ◆ Susceptible to Link spam

- Artificial link topographies created in order to boost page rank
- **Solution:** TrustRank

## ◆ Uses a single measure of importance

- Other models of importance
- **Solution:** Hubs-and-Authorities.