CS425: Algorithms for Web Scale Data

Lecture 1: PageRank Formulation

Most of the slides are from the Mining of Massive Datasets book.
These slides have been modified for CS425. The original slides can be accessed at: www.mmds.org

Lecture Overview

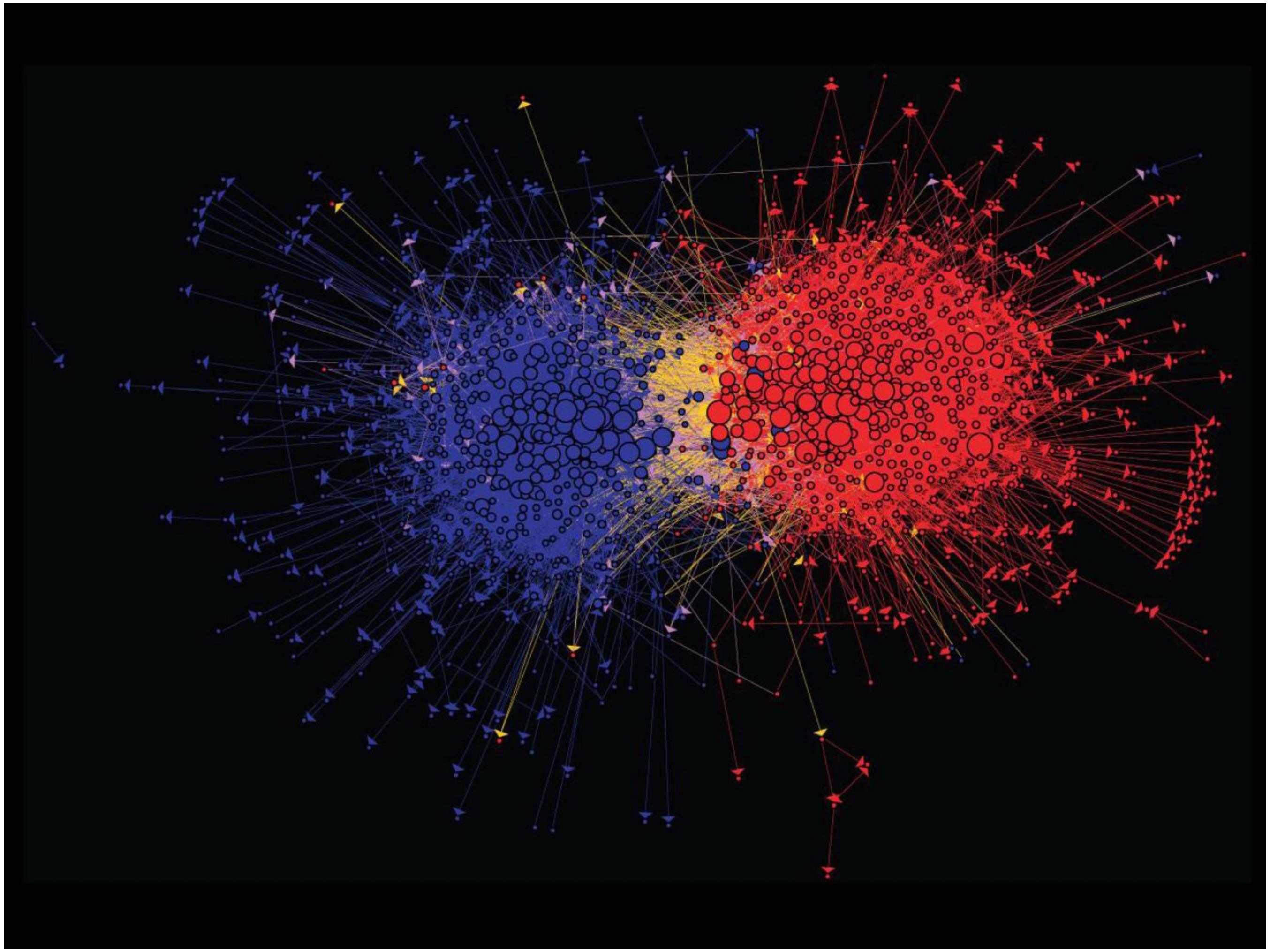
- Graph data overview
- Problems with early search engines
- PageRank Model
  - Flow Formulation
  - Matrix Interpretation
  - Random Walk Interpretation
  - Google's Formulation
- How to Compute PageRank

Graph Data: Social Networks

Facebook social graph

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

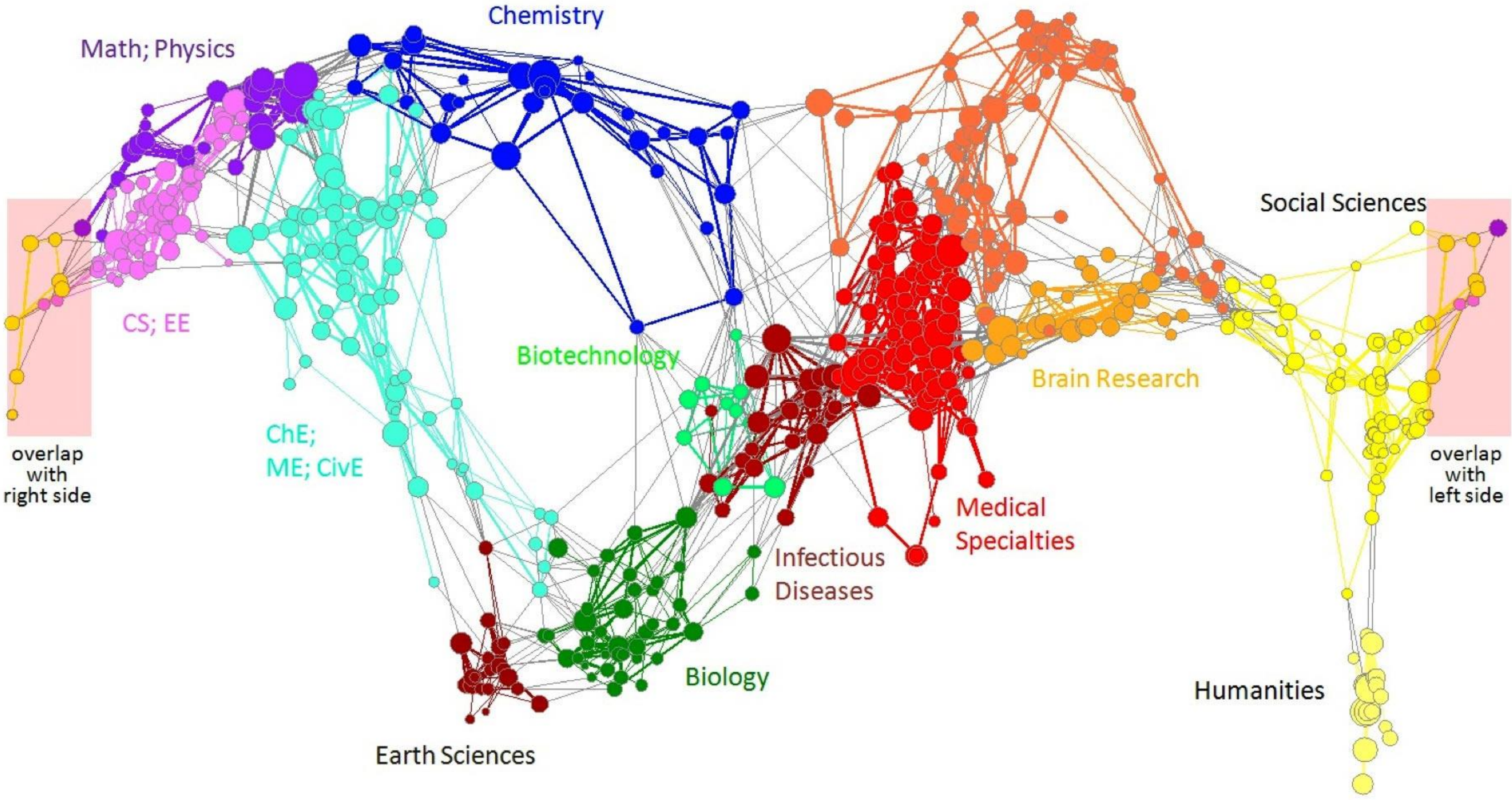
Graph Data: Media Networks



Connections between political blogs

Polarization of the network [Adamic-Glance, 2005]

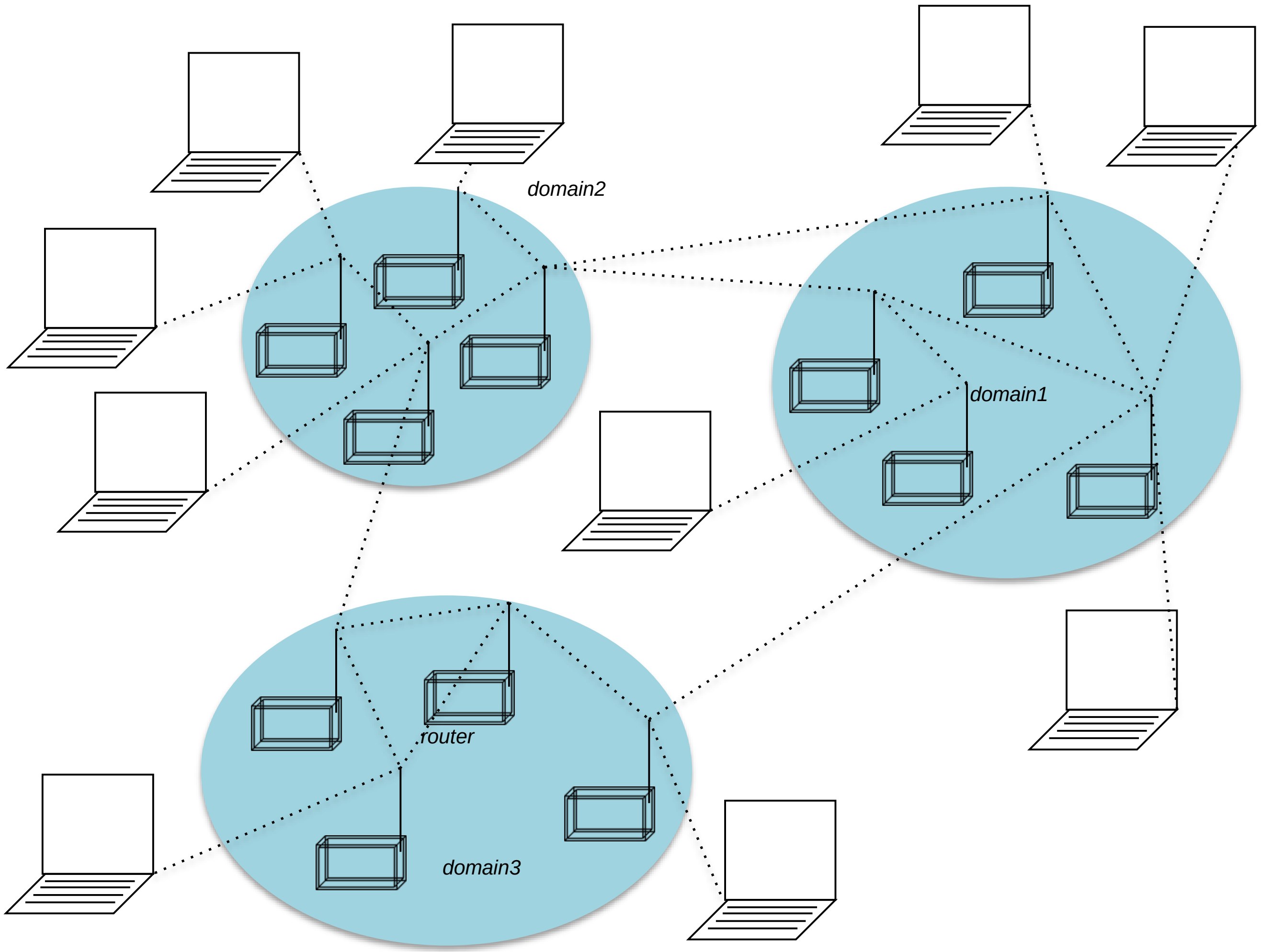
Graph Data: Information Nets



Citation networks and Maps of science

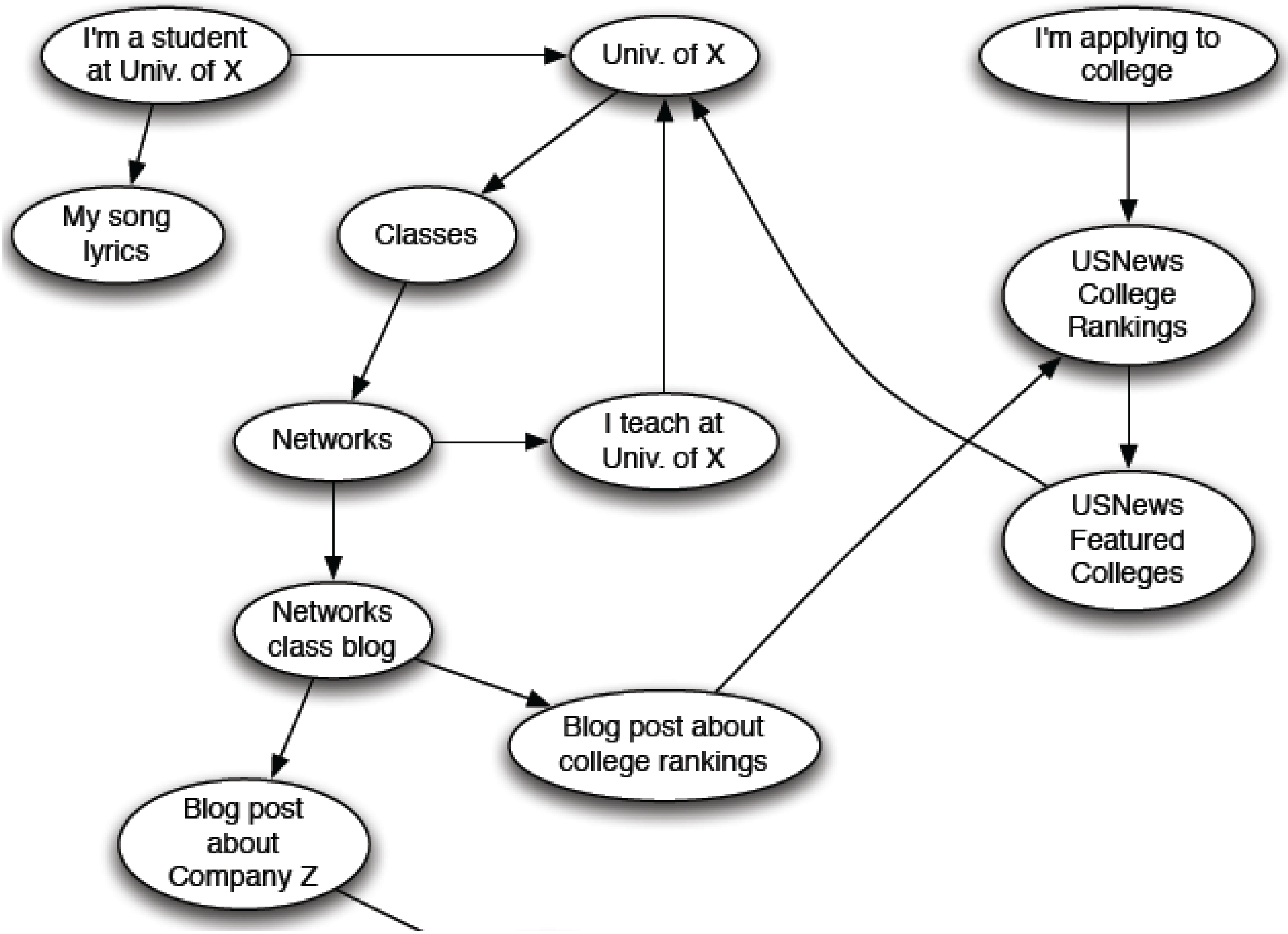
[Börner et al., 2012]

Graph Data: Communication Nets



Internet

Web as a Directed Graph



Broad Question

- **How to organize the Web?**
- **First try:** Human curated **Web directories**
  - Yahoo, DMOZ, LookSmart
- **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.

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

Early Search Engines

- Inverted index
  - Data structure that return pointers to all pages a term occurs

- Which page to return first?
  - Where do the search terms appear in the page?
  - How many occurrences of the search terms in the page?

- What if a spammer tries to fool the search engine?

Fooling Early Search Engines

- Example: A spammer wants his page to be in the top search results for the term "movies".
- Approach 1:
  - Add thousands of copies of the term "movies" to your page.
  - Make them invisible.
- Approach 2:
  - Search the term "movies".
  - Copy the contents of the top page to your page.
  - Make it invisible.
- Problem: Ranking only based on page contents
- Early search engines almost useless because of spam.

Google's Innovations

- *Basic idea: Search engine believes what other pages say about you instead of what you say about yourself.*

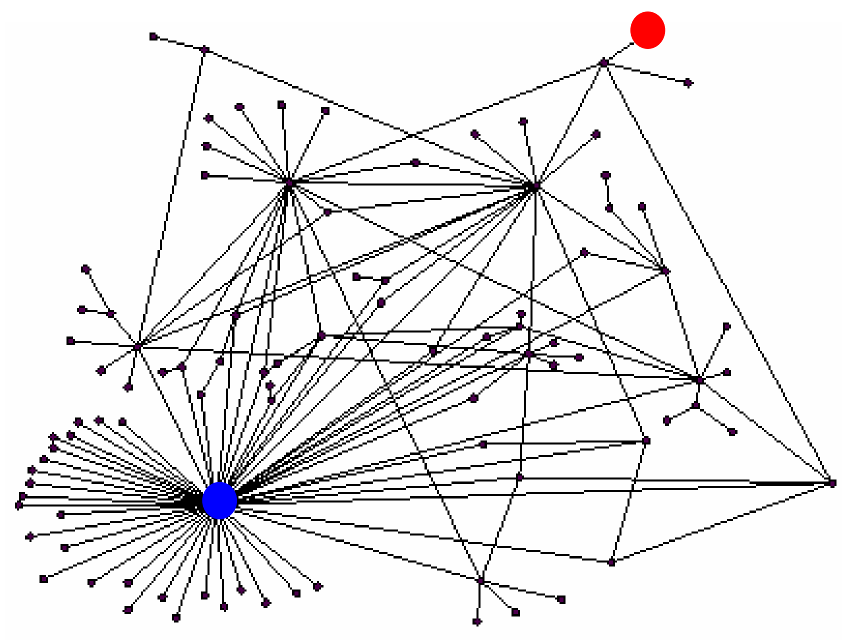
- Main innovations:
  1. Define the importance of a page based on:
     - How many pages point to it?
     - **How important are those pages?**
  2. Judge the contents of a page based on:
     - Which terms appear in the page?
     - **Which terms are used to link to the page?**

Ranking Nodes on the Graph

- **All web pages are not equally "important"**

  www.joe-schmoe.com vs. www.stanford.edu

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



Link Analysis Algorithms

- We will cover the following **Link Analysis approaches** for computing **importances of nodes in a graph**:
  - Page Rank
  - Topic-Specific (Personalized) Page Rank
  - Web Spam Detection Algorithms

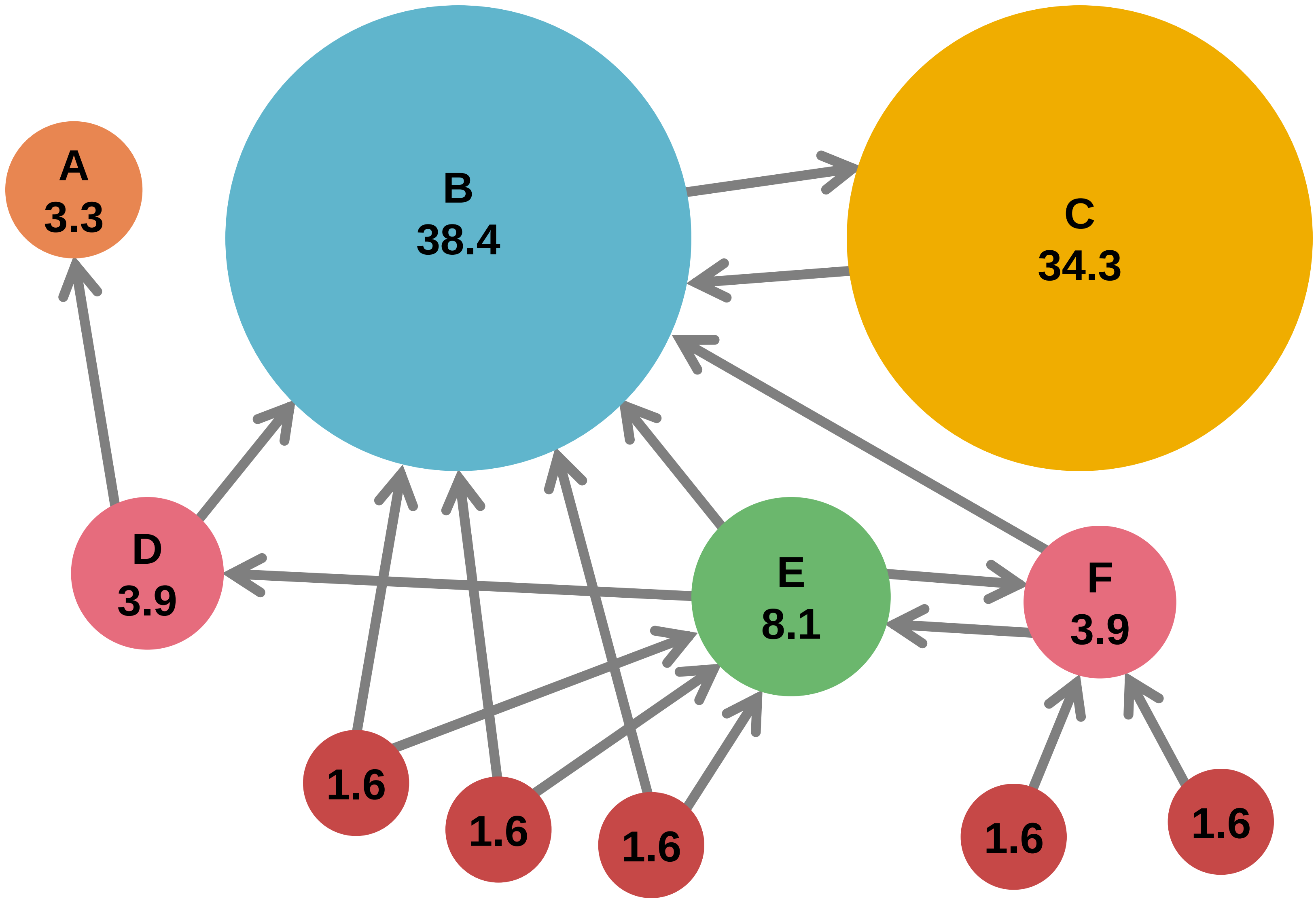
PageRank:
The “Flow” Formulation

Links as Votes

- **Think of in-links as votes:**
  - www.stanford.edu has 23,400 in-links
  - www.joe-schmoe.com has 1 in-link

- **Are all in-links are equal?**
  - **Links from important pages count more**
  - Recursive question!

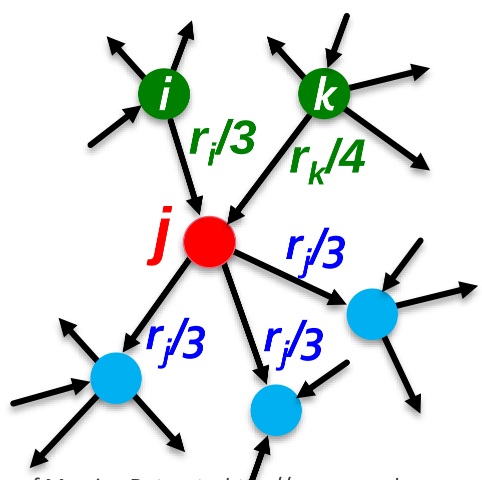
Example: PageRank Scores



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

$$r_j = r_i/3 + r_k/4$$

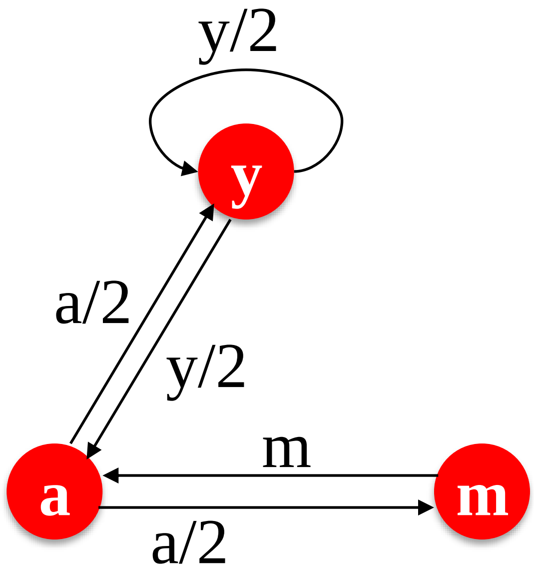


PageRank: The "Flow" Model

- **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**

$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d}_i}$$

d_i … out-degree of node i



"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
  - All solutions equivalent modulo the scale factor
- **Additional constraint forces uniqueness:**
  - $\boldsymbol{r_y + r_a + r_m = 1}$
  - **Solution:** $\boldsymbol{r_y = \frac{2}{5},\ r_a = \frac{2}{5},\ r_m = \frac{1}{5}}$
- **Gaussian elimination method works for small examples, but we need a better method for large web-size graphs**
- **We need a new formulation!**

PageRank:
The Matrix Formulation

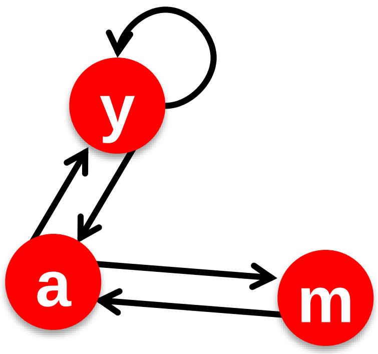
PageRank: Matrix Formulation

- **Adjacency matrix M**
  - Let page i have d_i out-links
  - If $i \rightarrow j$, then $M_{ji} = \frac{1}{d_i}$ else $M_{ji} = 0$
- **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**

$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d}_i}$$

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

Example: Flow Equations & M



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$$r = M \cdot r$$

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

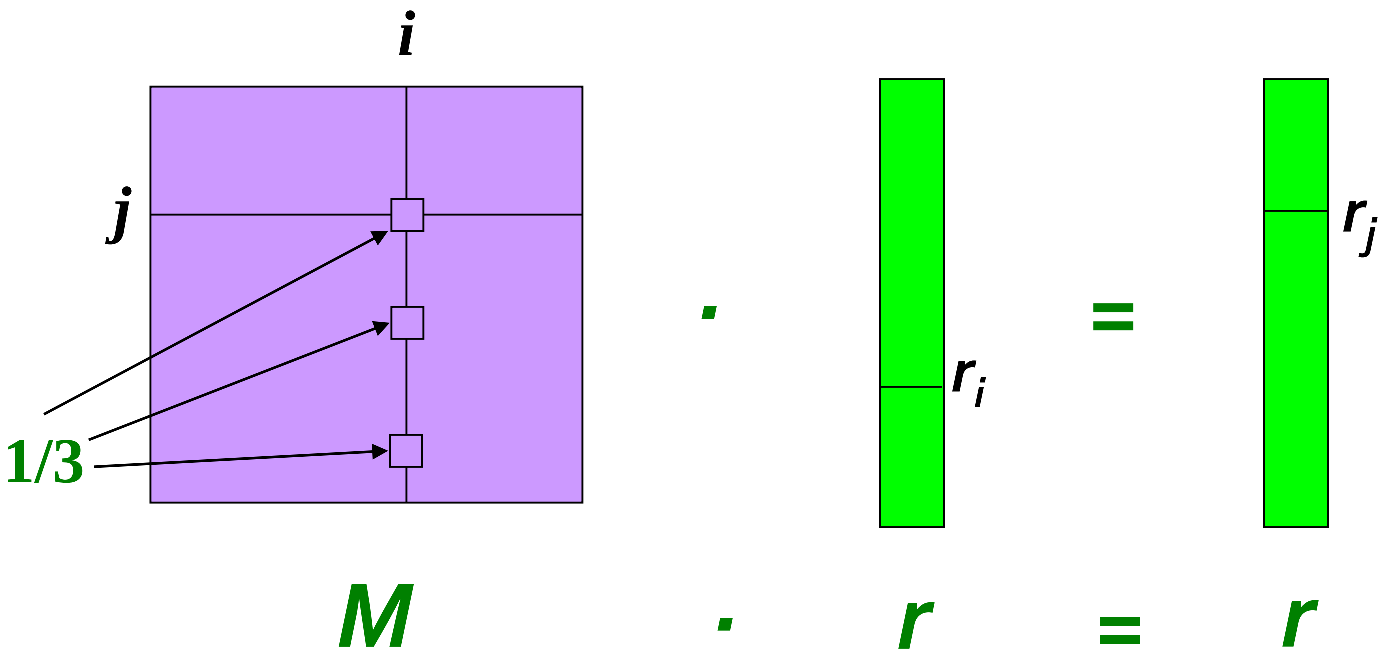
$$\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 1 \\ 0 & ½ & 0 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}$$

Example

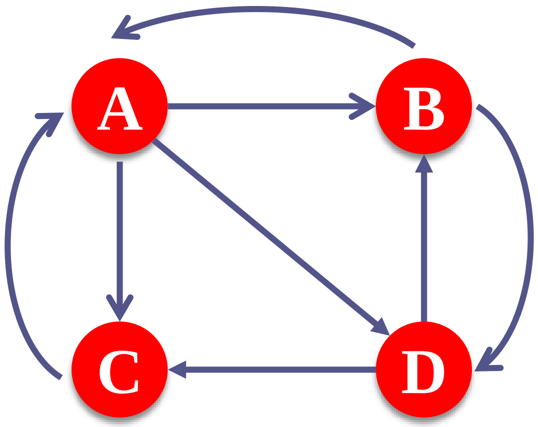
- **Remember the flow equation:** $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
- **Flow equation in the matrix form**

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

  - **Suppose page *i* links to 3 pages, including *j***



Exercise: Matrix Formulation



$$
\begin{array}{ccccc}
\mathbf{M} & & \mathbf{r} & & \mathbf{r} \\
\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} & \cdot & \begin{bmatrix} r_A \\ r_B \\ r_C \\ r_D \end{bmatrix} & = & \begin{bmatrix} r_A \\ r_B \\ r_C \\ r_D \end{bmatrix}
\end{array}
$$

Linear Algebra Reminders

- A is a *column stochastic matrix* iff each of its columns add up to 1 and there are no negative entries.
  - Our adjacency matrix M is column stochastic. Why?
- If there exist a vector x and a scalar λ such that $Ax = \lambda x$, then:
  - x is an *eigenvector* and λ is an *eigenvalue* of A
  - The *principal eigenvector* is the one that corresponds to the largest eigenvalue.
- The largest eigenvalue of a column stochastic matrix is 1.

$$Ax = x, \text{ where } x \text{ is the principal eigenvector}$$

Eigenvector Formulation

- **PageRank flow formulation:**

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

- 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***

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

- **We can now efficiently solve for *r*!**
  The method is called Power iteration

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: $\mathbf{r}^{(0)} = [1/N,\ldots.,1/N]^T$
  - Iterate: $\mathbf{r}^{(t+1)} = \mathbf{M} \cdot \mathbf{r}^{(t)}$
  - Stop when $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}|_1 < \varepsilon$

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

d_i …. out-degree of node i

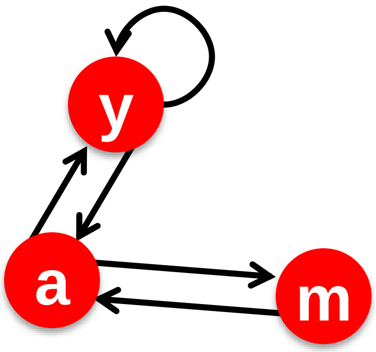
$|\mathbf{x}|_1 = \sum_{1 \le i \le N} |x_i|$ is the $\mathbf{L_1}$ norm
Can use any other vector norm, e.g., Euclidean

PageRank: How to solve?

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

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

Iteration 0, 1, 2, …



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

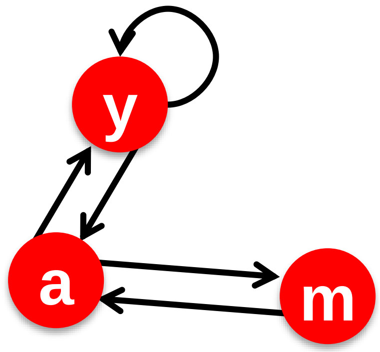
$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

PageRank: How to solve?

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



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

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

- **Example:**

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

Iteration 0, 1, 2, …

Power Iteration Convergence

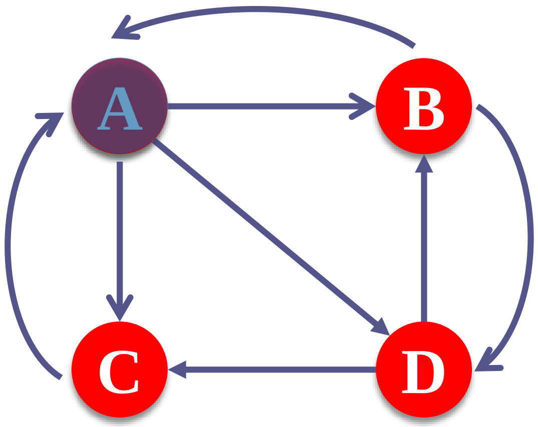
- **Power iteration:**
  A method for finding principal eigenvector (the vector corresponding to the largest eigenvalue)
  - $\boldsymbol{r}^{(1)} = \boldsymbol{M} \cdot \boldsymbol{r}^{(0)}$
  - $\boldsymbol{r}^{(2)} = \boldsymbol{M} \cdot \boldsymbol{r}^{(1)} = \boldsymbol{M}\left(\boldsymbol{M}\boldsymbol{r}^{(1)}\right) = \boldsymbol{M}^{2} \cdot \boldsymbol{r}^{(0)}$
  - $\boldsymbol{r}^{(3)} = \boldsymbol{M} \cdot \boldsymbol{r}^{(2)} = \boldsymbol{M}\left(\boldsymbol{M}^{2}\boldsymbol{r}^{(0)}\right) = \boldsymbol{M}^{3} \cdot \boldsymbol{r}^{(0)}$
- **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 $\boldsymbol{M}$

PageRank:
Random Walk Interpretation

Random Walk Interpretation of PageRank

- Consider a web surfer:
  - He starts at a random page
  - He follows a random link at every time step
  - After a sufficiently long time:
    - What is the probability that he is at page j?
      - This probability corresponds to the page rank of j.

Example: Random Walk

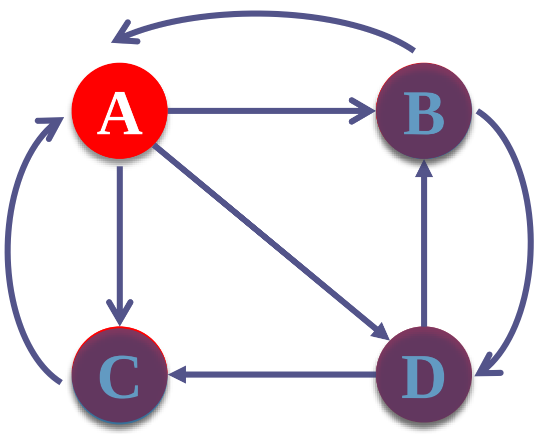


Time t = 0: Assume the random surfer is at A.

Time t = 1:

p(A, 1) = ? 0
p(B, 1) = ? 1/3
p(C, 1) = ? 1/3
p(D, 1) = ? 1/3

Example: Random Walk



Time t = 1:

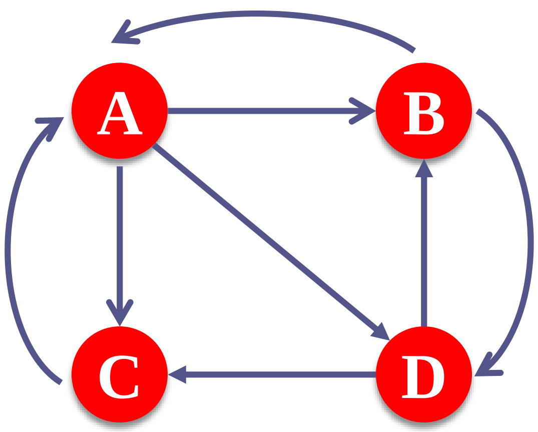
p(B, 1) = 1/3
p(C, 1) = 1/3
p(D, 1) = 1/3

Time t=2:

p(A, 2) = ?

p(A, 2) = p(B, 1) . p(B→A) + p(C, 1) . p(C→A)
= 1/3 . 1/2 + 1/3 . 1 = 3/6

Example: Transition Matrix



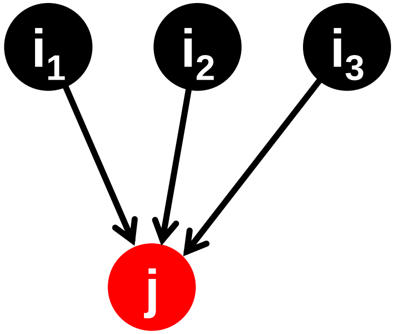
$$\underbrace{\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}}_{\mathbf{M}} \cdot \underbrace{\begin{bmatrix} p_A \\ p_B \\ p_C \\ p_D \end{bmatrix}}_{\mathbf{p(t)}} = \underbrace{\begin{bmatrix} p_A \\ p_B \\ p_C \\ p_D \end{bmatrix}}_{\mathbf{p(t+1)}}$$

p(A, t+1) = p(B, t) . p(B→A) + p(C, t) . p(C→A)

p(C, t+1) = p(A, t) . p(A→C) + p(D, t) . p(D→C)

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
- **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

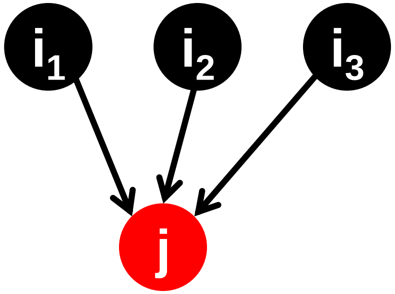


$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d_{out}(i)}}$$

The Stationary Distribution

- **Where is the surfer at time *t+1*?**
  - Follows a link uniformly at random

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



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

- 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**

Rank of page j = Probability that the surfer is at page j after a long 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**

Summary So Far

- PageRank formula: $$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$ $\mathrm{d_i}$ …. out-degree of node i

- Iterative algorithm:
  1. Initialize rank of each page to 1/N (where N is the number of pages)
  2. Compute the next page rank values using the formula above
  3. Repeat step 2 until the page rank values do not change much

- Same algorithm, but different interpretations

Summary So Far (cont'd)

- Eigenvector interpretation:
  - Compute the principal eigenvector of stochastic adjacency matrix M

$$r = M \cdot r$$

  - Power iteration method

- Random walk interpretation:
  - Rank of page i is the probability that a surfer is at i after random walk

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

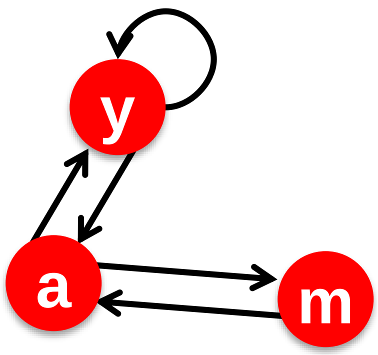
  - Guaranteed to converge to a unique solution under certain conditions

Convergence Conditions

- To guarantee convergence to a meaningful and unique solution, the transition matrix must be:
  1. Column stochastic
  2. Irreducible
  3. Aperiodic

Column Stochastic

- Column stochastic:
  - All values in the matrix are non-negative
  - Sum of each column is 1



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

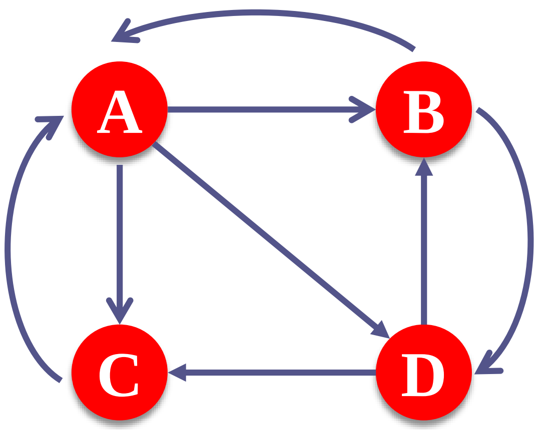
$$\mathbf{r_y = r_y/2 + r_a/2}$$
$$\mathbf{r_a = r_y/2 + r_m}$$
$$\mathbf{r_m = r_a/2}$$

What if we remove the edge m → a ?

No longer column stochastic

Irreducible

- Irreducible: From any state, there is a non-zero probability of going to another.
  - Equivalent to: Strongly connected graph



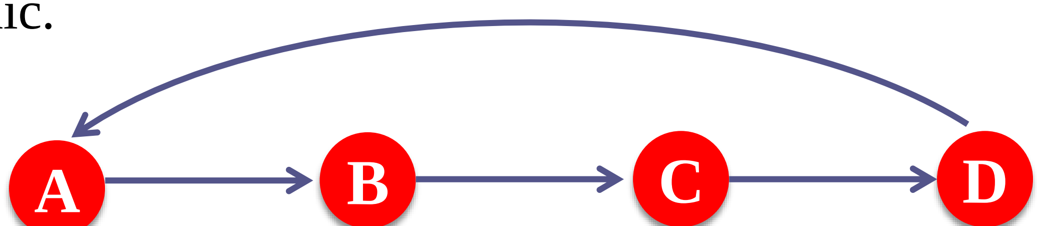
Irreducible graph

What if we remove the edge C → A ?

No longer irreducible.

Aperiodic

- State i has period k if any return to state i must occur in *multiples of k time steps*.
- If k = 1 for a state, it is called aperiodic.
  - Returning to the state at irregular intervals
- A Markov chain is aperiodic if all its states are aperiodic.
  - If Markov chain is irreducible, one aperiodic state means all stated are aperiodic.



$$\left.\begin{array}{c} t_0 \\ t_0 + 4 \\ t_0 + 8 \end{array}\right\} k = 4$$

How to make this aperiodic?

Add any self edge

PageRank:
The Google Formulation

PageRank: Three Questions

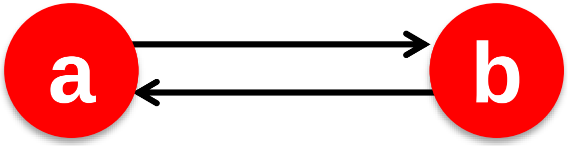
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

or equivalently

$$r = Mr$$

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

Does this converge?



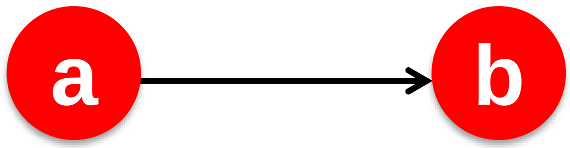
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d}_\mathrm{i}}$$

- **Example:**

$$\begin{matrix} r_a \\ r_b \end{matrix} = \begin{matrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{matrix}$$

Iteration 0, 1, 2, …

Does it converge to what we want?



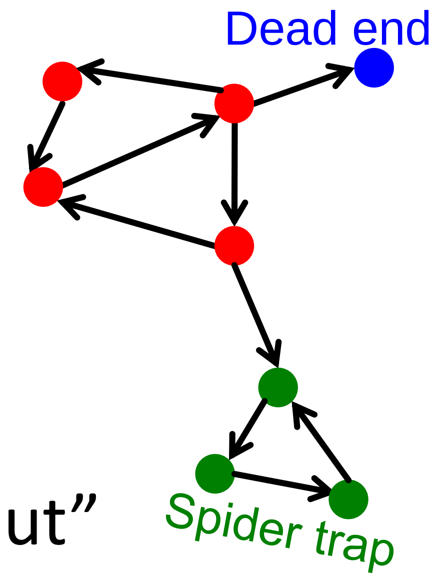
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

- **Example:**

$$\begin{array}{ccccc} \mathrm{r_a} & & 1 & 0 & 0 & 0 \\ & = & & & & \\ \mathrm{r_b} & & 0 & 1 & 0 & 0 \end{array}$$

Iteration 0, 1, 2, …

PageRank: Problems



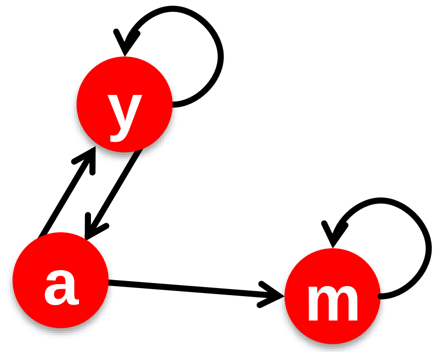
2 problems:

- **(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 walk gets "stuck" in a trap
  - And eventually spider traps absorb all importance

Problem: Spider Traps

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

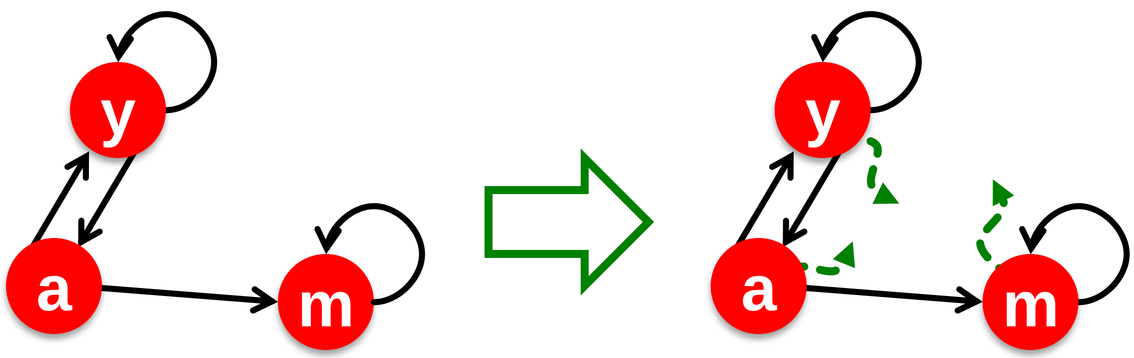
r_y	1/3	2/6	3/12	5/24		0
r_a =	1/3	1/6	2/12	3/24	…	0
r_m	1/3	3/6	7/12	16/24		1

Iteration 0, 1, 2, …

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

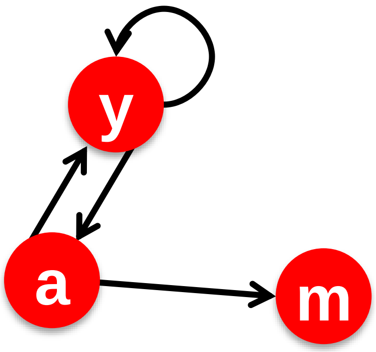
Solution: Teleports!

- **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

$\mathbf{r_y = r_y/2 + r_a/2}$

$\mathbf{r_a = r_y/2}$

$\mathbf{r_m = r_a/2}$

- **Example:**

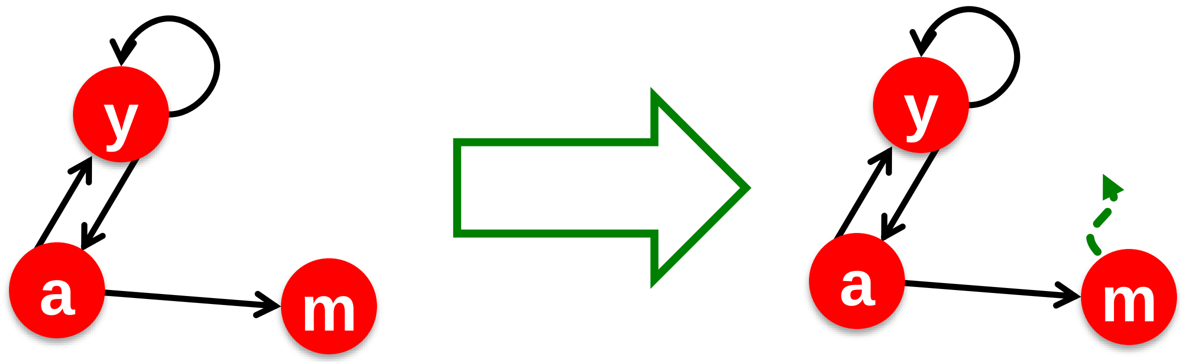
$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\ 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{matrix}$$

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:** 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 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 $1\text{-}\beta$, jump to some random page

- **PageRank equation** [Brin-Page, 98]

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

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 or explicitly follow random teleport links with probability 1.0 from dead-ends.

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*:**

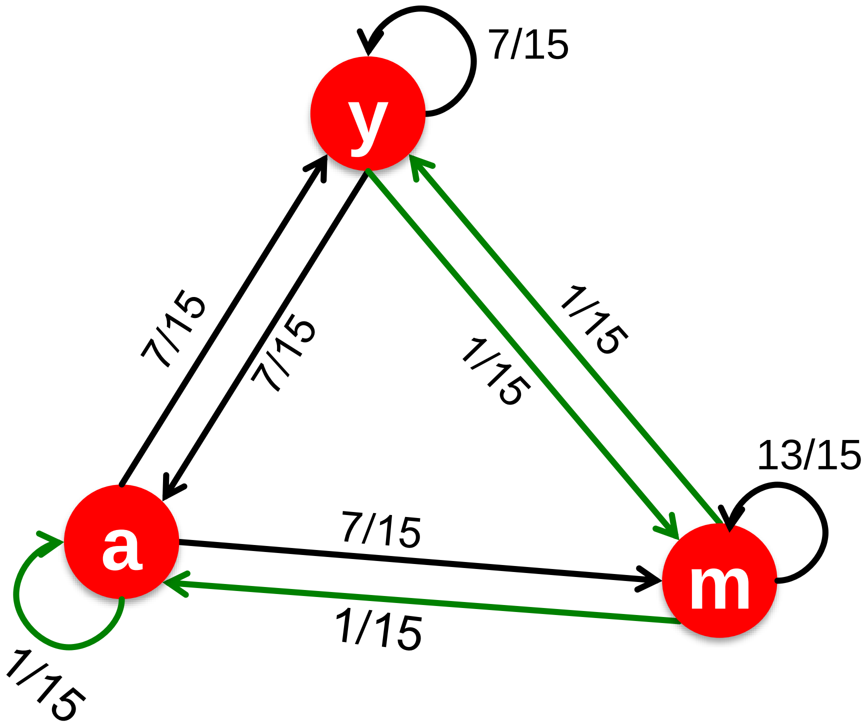
$[1/N]_{NxN}$…N by N matrix where all entries are 1/N

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

- **We have a recursive problem:** $\boldsymbol{r} = \boldsymbol{A} \cdot \boldsymbol{r}$

  And the Power method still works!
- **What is β?**
  - In practice β =*0.8,0.9* (make *5* steps on avg., jump)

Random Teleports ($\beta = 0.8$)



$$\begin{array}{c}\mathbf{M}\\ 0.8\begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix}\end{array} + 0.2 \begin{array}{c}[1/N]_{NxN}\\ \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}\end{array}$$

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

A

y		1/3	0.33	0.24	0.26		7/33
a	=	1/3	0.20	0.20	0.18	. . .	5/33
m		1/3	0.46	0.52	0.56		21/33

Matrix Formulation

- Suppose there are N pages
- Consider page i, with $\mathbf{d}_i$ out-links
- We have $M_{ji} = 1/|d_i|$ when $i \to j$
  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

How do we actually compute the PageRank?

Computing Page Rank

- **Key step is matrix-vector multiplication**
  - $\boldsymbol{r}^{\text{new}} = \boldsymbol{A} \cdot \boldsymbol{r}^{\text{old}}$
- Easy if we have enough main memory to hold **A**, $\mathbf{r}^{\text{old}}$, $\mathbf{r}^{\text{new}}$
- **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!

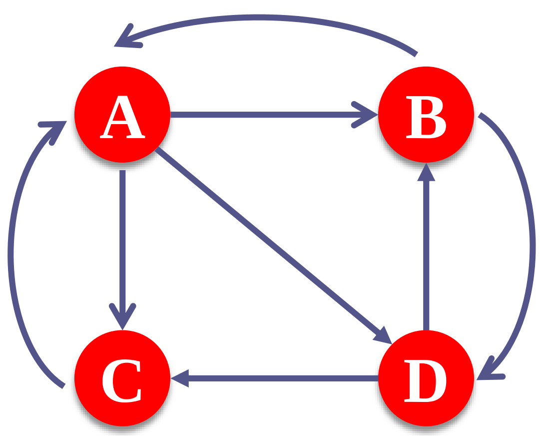
$$\mathbf{A} = \beta \cdot \mathbf{M} + (1-\beta)\,[1/N]_{N \times N}$$

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

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

Matrix Sparseness

- Reminder: Our original matrix was sparse.
  - On average: ~10 out-links per vertex
  - # of non-zero values in matrix M: ~10N
- **Teleport links make matrix M dense**.
- Can we convert it back to the sparse form?



Original matrix without teleports

0	1/2	1	0
1/3	0	0	1/2
1/3	0	0	1/2
1/3	1/2	0	0

Rearranging the Equation

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

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

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

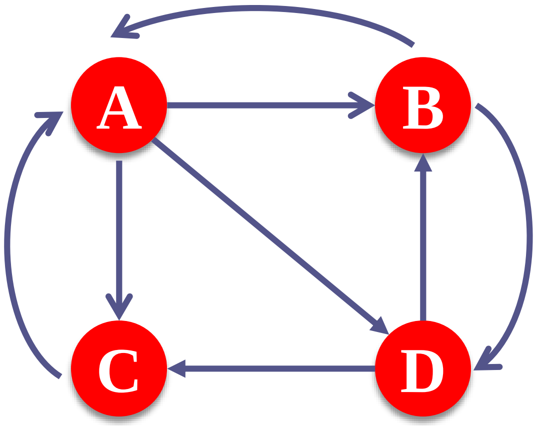
since $\sum r_i = 1$

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

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

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

Example: Equation with Teleports



$$\overset{\mathbf{r}^{\mathbf{new}}}{\begin{bmatrix} r_A \\ r_B \\ r_C \\ r_D \end{bmatrix}} = \beta \overset{\mathbf{M}}{\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}} \cdot \overset{\mathbf{r}^{\mathbf{old}}}{\begin{bmatrix} r_A \\ r_B \\ r_C \\ r_D \end{bmatrix}} + (1-\beta) \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$$

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

Sparse Matrix Formulation

- We just rearranged the **PageRank equation**

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

  - 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: Without Dead Ends

- **Input: Graph G and parameter β**
  - Directed graph G (cannot have **dead ends**)
  - Parameter β
- **Output: PageRank vector r^{new}**
  - **Set:** $r_j^{old} = \frac{1}{N}$
  - **repeat until convergence:** $\sum_j \left| r_j^{new} - r_j^{old} \right| > \varepsilon$
    - $\forall j$: $\boldsymbol{r_j^{new} = \sum_{i \to j} \beta \frac{r_i^{old}}{d_i}}$
      $\boldsymbol{r_j^{new} = 0}$ if in-degree of $\boldsymbol{j}$ is **0**
    - **Add constant terms:**
      $\forall \boldsymbol{j}$: $\boldsymbol{r_j^{new} = r_j^{new} + \frac{1-\beta}{N}}$
    - $\boldsymbol{r^{old} = r^{new}}$

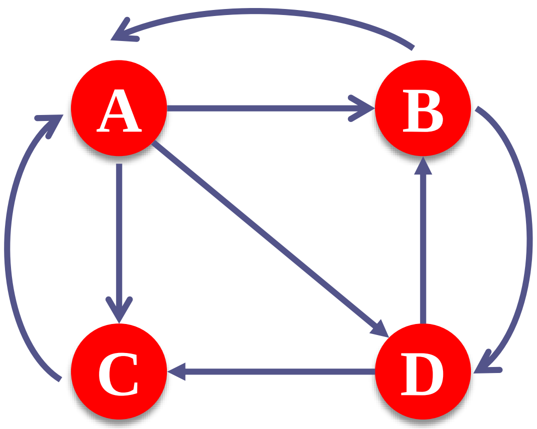
PageRank: The Complete Algorithm

- **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:** $\sum_j \left| r_j^{new} - r_j^{old} \right| > \varepsilon$
    - $\forall j$: $\boldsymbol{r'^{new}_j = \sum_{i \to j} \beta \frac{r_i^{old}}{d_i}}$
      $\boldsymbol{r'^{new}_j = 0}$ if in-degree of $\boldsymbol{j}$ is **0**
    - **Now re-insert the leaked PageRank:**
      $\forall \boldsymbol{j}$: $\boldsymbol{r_j^{new} = r'^{new}_j + \frac{1-S}{N}}$ **where:** $S = \sum_j r'^{new}_j$
    - $\boldsymbol{r^{old} = r^{new}}$

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**.

Sparse Matrix Encoding: First Try

Store a triplet for each nonzero entry: (row, column, weight)



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

(2, 1, 1/3); (3, 1, 1/3); (4, 1, 1/3); (1, 2, 1/2); (4, 2, 1/2); (1, 3, 1); …

Assume 4 bytes per integer and 8 bytes per float: 16 bytes per entry

Inefficient: Repeating the column index and weight multiple times

Sparse Matrix Encoding

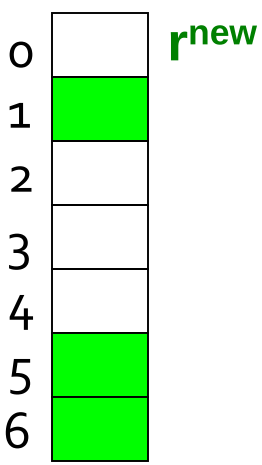
- **Store entries per source node**
  - Source index and degree stored once per node
  - Space proportional roughly to number of links
  - Say 10N, or 4*10*1 billion = 40GB
  - **Still won't fit in memory, but will fit on disk**

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

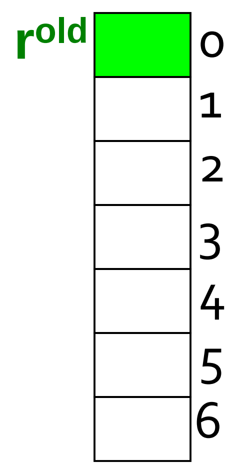
Basic Algorithm: Update Step

- **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 $\mathbf{r^{new} = (1\text{-}\beta) / N}$
For each page i (of out-degree d_i):
  Read into memory: i, d_i, $dest_1, \ldots, dest_{d_i}$, $r^{old}(i)$
  For $\mathbf{j = 1 \ldots d_i}$
    $\mathbf{r^{new}(dest_j) \mathrel{+}= \beta\, r^{old}(i) / d_i}$



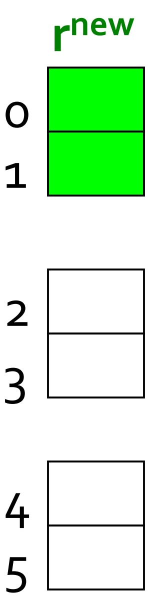
source	degree	destination
0	3	1, 5, 6
1	4	17, 64, 113, 117
2	2	13, 23



Analysis

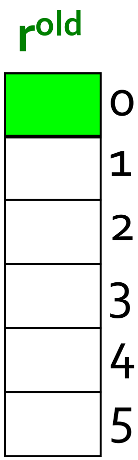
- **Assume enough RAM to fit r^{new} into memory**
  - Store r^{old} and matrix M on disk
- **In each iteration, we have to:**
  - Read r^{old} and M
  - Write r^{new} back to disk
  - **Cost per iteration of Power method:**
    $= 2|r| + |M|$
- **Question:**
  - What if we could not even fit r^{new}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

Block-based Update Algorithm



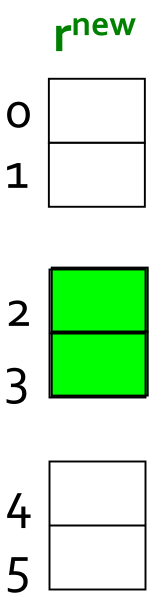
src	degree	destination
0	4	0, 1, 3, 5
1	2	0, 5
2	2	3, 4

M



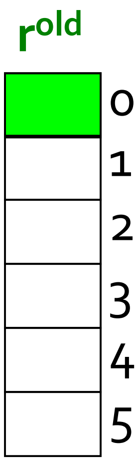
- Break r^{new} into k blocks that fit in memory
- Scan M and r^{old} once for each block

Block-based Update Algorithm



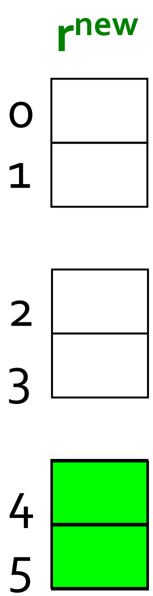
src	degree	destination
0	4	0, 1, 3, 5
1	2	0, 5
2	2	3, 4

M



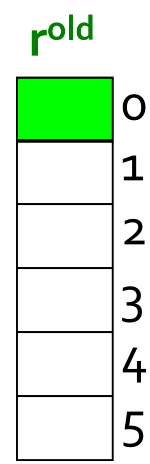
- Break r^{new} into k blocks that fit in memory
- Scan M and r^{old} once for each block

Block-based Update Algorithm



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

M

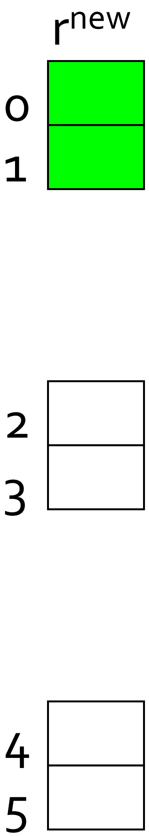


- Break r^{new} into k blocks that fit in memory
- Scan M and r^{old} once for each block

Analysis of Block Update

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

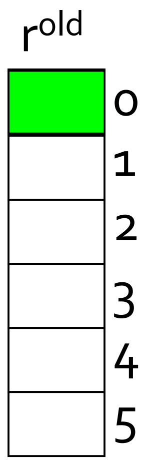
Block-Stripe Update Algorithm



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

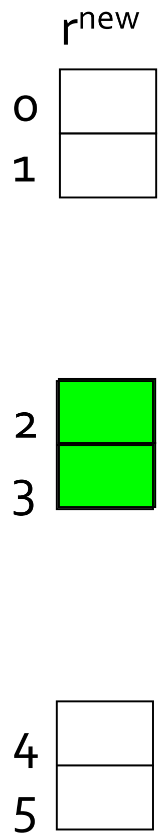
0	4	3
2	2	3

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}

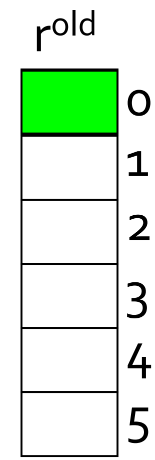
Block-Stripe Update Algorithm



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 $\boldsymbol{r}^{new}$

Block-Stripe Update Algorithm

r^{new}

0
1

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

2
3

src	degree	destination
0	4	3
2	2	3

4
5

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

r^{old}

0
1
2
3
4
5

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

Block-Stripe Analysis

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

Some Problems with Page Rank

- **Measures generic popularity of a page**
  - Biased against topic-specific authorities
  - **Solution:** Topic-Specific PageRank (**next**)
- **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