SUNY Korea CSE549
Spring 2017
Instructor: Sael lee

Random Walk

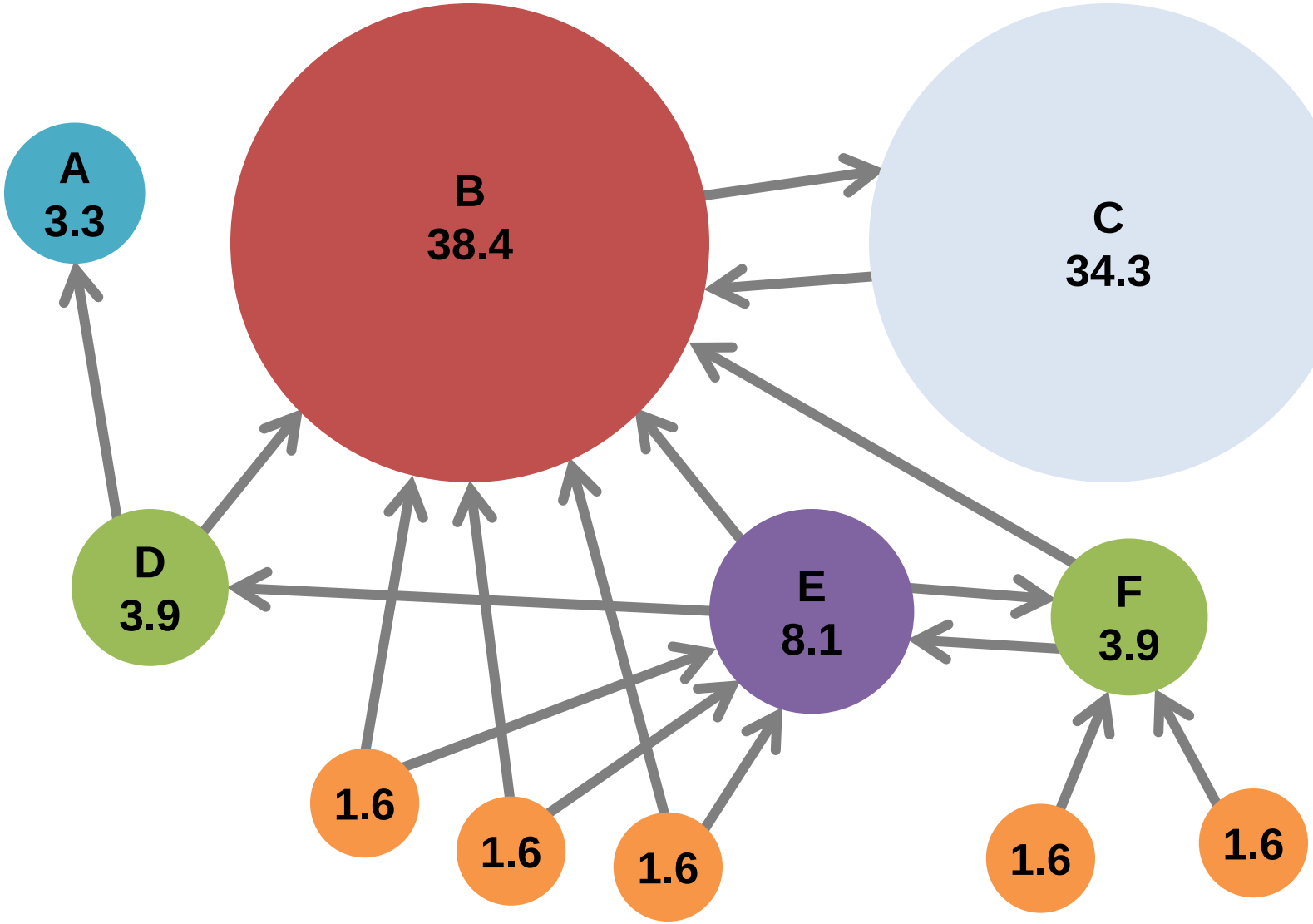
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets,
http://www.mmds.org

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.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!

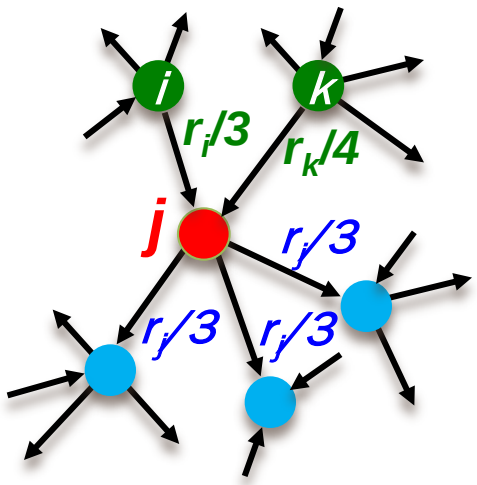
Example: PageRank Scores



Simple Recursive Formulation

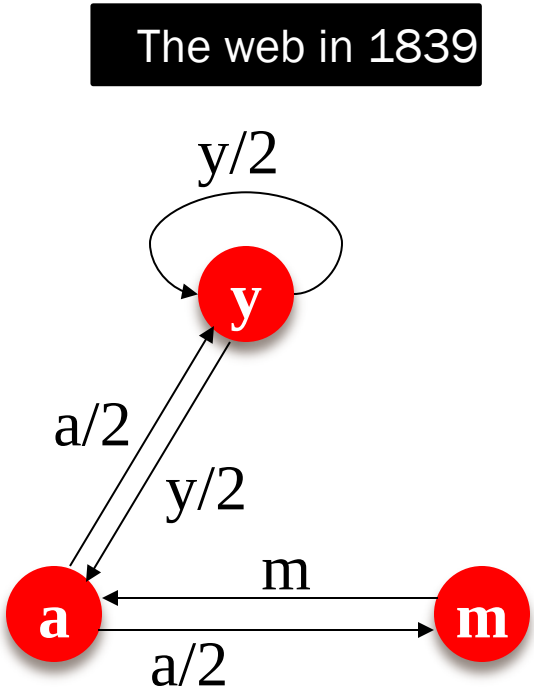
- Each link's vote is proportional to the **importance** of its source page
- If page ***j*** with importance r_j has ***n*** out-links, each link gets r_j / n votes
- Page ***j***'s own importance is the sum of the votes on its in-links

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



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



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

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

- **3 equations, 3 unknowns, no constants**
  - No unique solution
  - 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!**

Flow equations:

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

PageRank: Matrix Formulation

- **Stochastic adjacency matrix M**
  - Let page i has d_i out-links
  - If $i \rightarrow j$, 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**

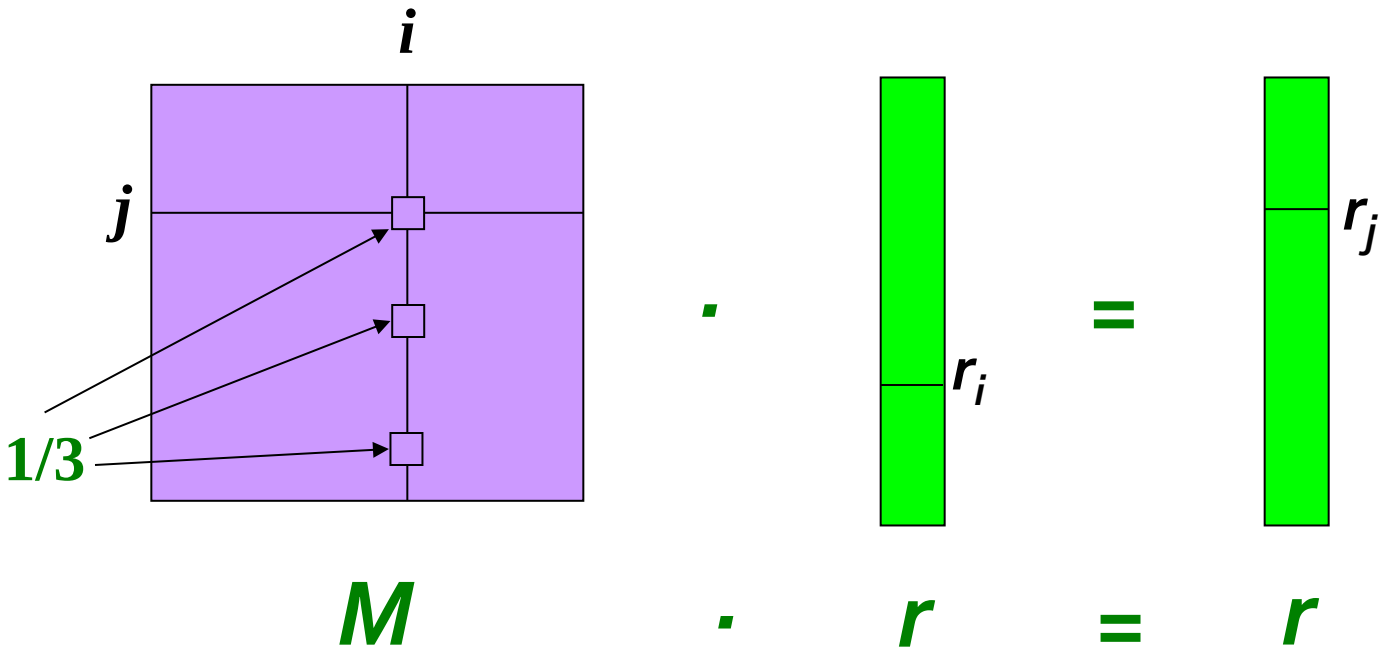
$$r = M \cdot r \qquad r_j = \sum_{i \rightarrow j} \frac{r_i}{\mathrm{d_i}}$$

Example

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

$$M \cdot r = r$$

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



Eigenvector Formulation

- **The flow equations can be written**

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

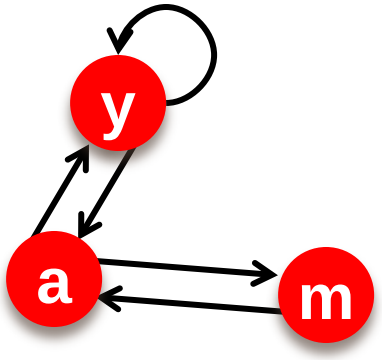
- So the **rank vector** $\boldsymbol{r}$ is an **eigenvector** of the stochastic web matrix $\boldsymbol{M}$
  - In fact, its first or principal eigenvector, with corresponding eigenvalue ***1***
    - Largest eigenvalue of $\boldsymbol{M}$ is **1** since $\boldsymbol{M}$ is column stochastic (with non-negative entries)
      - *We know* $\boldsymbol{r}$ *is unit length and each column of* $\boldsymbol{M}$ *sums to one, so* $\boldsymbol{Mr} \leq \mathbf{1}$

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

NOTE: ***x*** is an eigenvector with the corresponding eigenvalue **λ** if:

$$\boldsymbol{Ax} = \boldsymbol{\lambda x}$$

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

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

d_i …. out-degree of node i

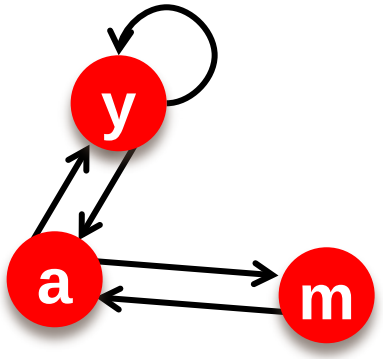
$|\mathbf{x}|_1 = \sum_{1 \le i \le N} |x_i|$ is the **L_1** norm
Can use any other vector norm, e.g., Euclidean

PageRank: How to solve?

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

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

Iteration 0, 1, 2, …



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

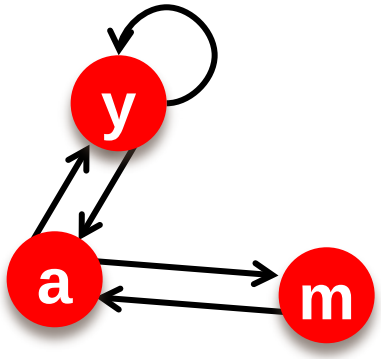
$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

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**
- **Example:**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

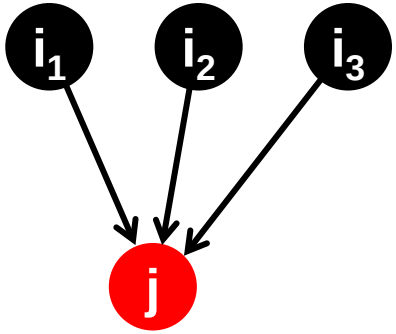
$r_y = r_y/2 + r_a/2$
$r_a = r_y/2 + r_m$
$r_m = r_a/2$

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

Iteration 0, 1, 2, …

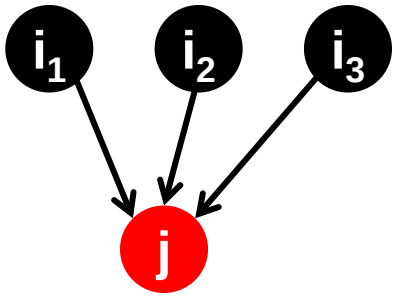
Random Walk Interpretation

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



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

The Stationary Distribution



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

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

- Suppose the random walk reaches a state $\boldsymbol{p(t+1) = M \cdot p(t) = p(t)}$

  then $\boldsymbol{p(t)}$ is **stationary distribution** of a random walk
- **Our original rank vector $\boldsymbol{r}$** satisfies $\boldsymbol{r = M \cdot r}$
  - **So, $\boldsymbol{r}$ 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 **strongly connected** and **has no deadend**,
the **stationary distribution is unique** and eventually will be reached no matter what the initial probability distribution at time **t = 0**

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets,
http://www.mmds.org

PageRank: The Google Formulation

PageRank: Three Questions

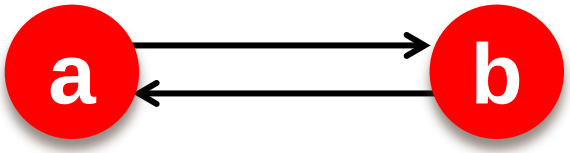
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

or equivalently

$$r = Mr$$

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

Does this converge?



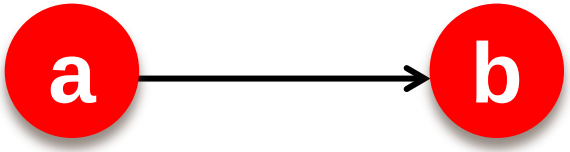
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

- **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:**

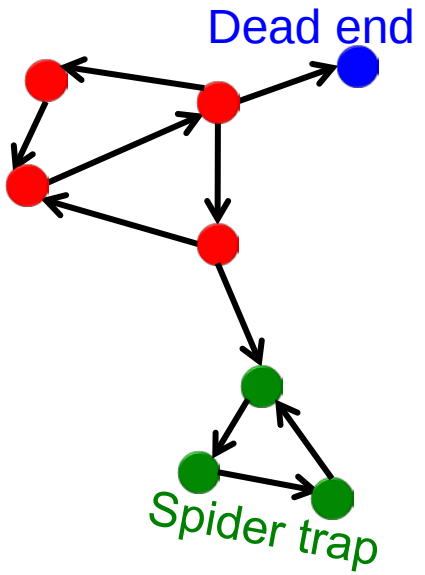
r_a	=	1	0	0	0
r_b		0	1	0	0

Iteration 0, 1, 2, …

PageRank: Problems

2 problems:

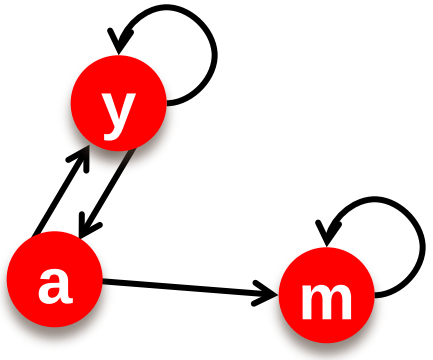
- **(1)** Some pages are **dead ends** (have no out-links)
  - Random walk has "nowhere" to go to
  - Such pages cause importance to "leak out"



- **(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:**

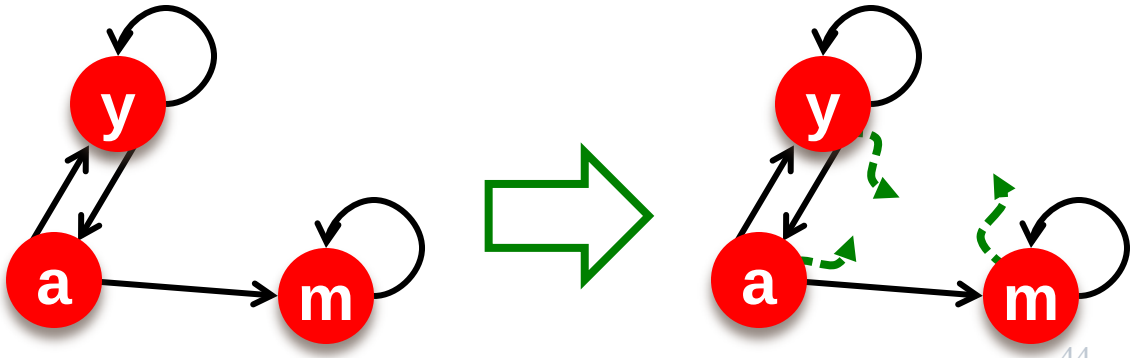
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\ 1/3 & 3/6 & 7/12 & 16/24 & & 1 \end{matrix}$$

Iteration 0, 1, 2, …

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

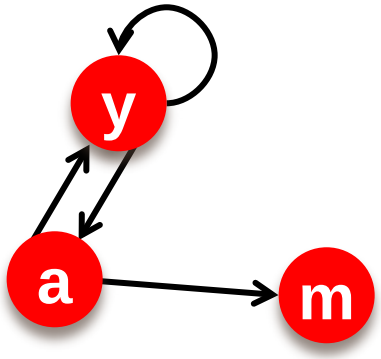
Solution: Teleports!

- **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:**

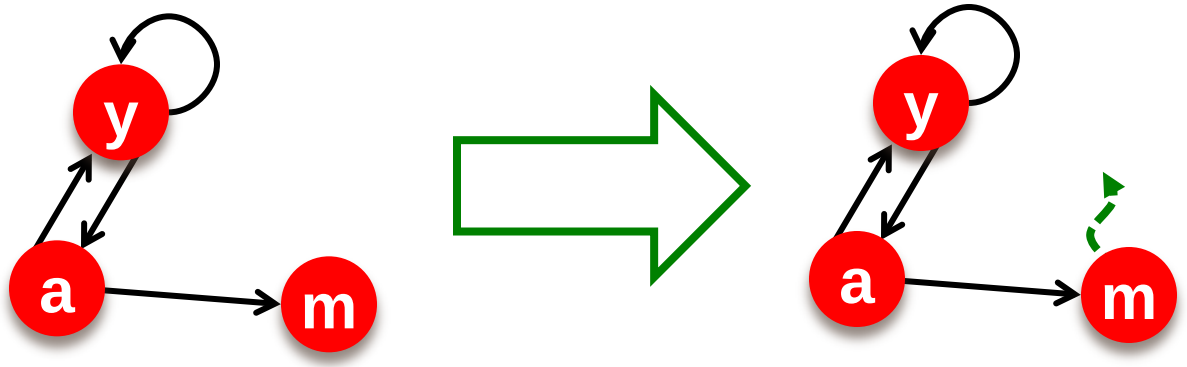
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{matrix}$$

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, 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 so our initial assumptions are not met
  - **Solution:** Make matrix column stochastic by always teleportin g 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 $1-\beta$, 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 $\boldsymbol{M}$ has no dead ends. We can either preprocess matrix $\boldsymbol{M}$ 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]

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

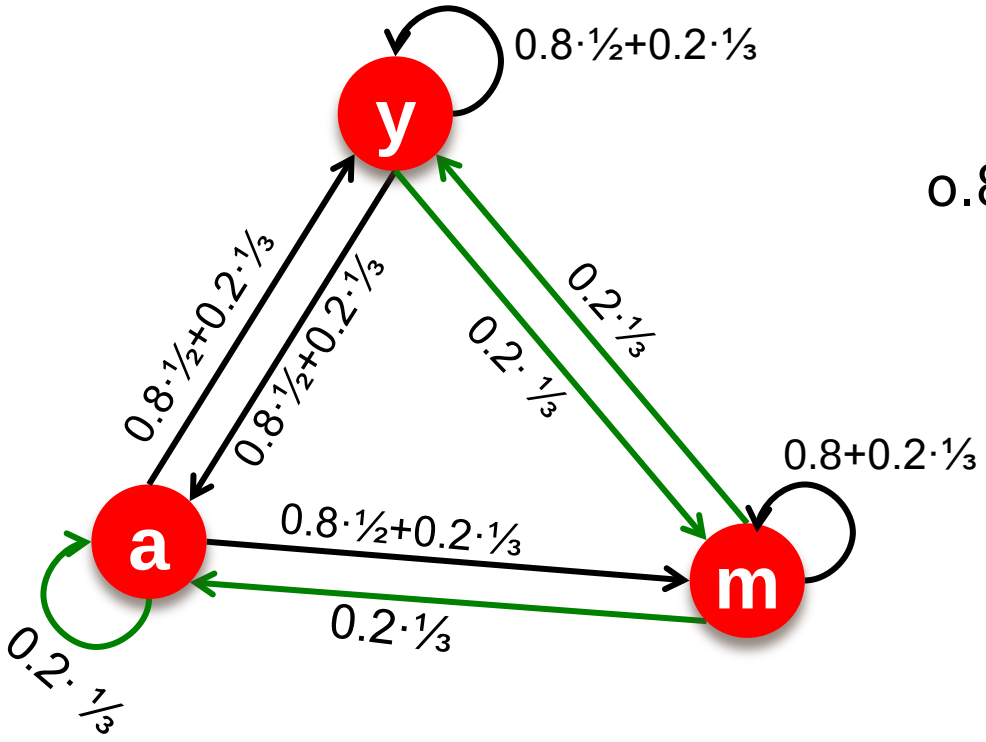
- **The Google Matrix A:**

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

$[1/N]_{NxN}$…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 ($\beta = 0.8$)



M

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

$[1/N]_{NxN}$

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

A

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

r = A r

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

PageRank: The Complete Algorithm

- **Input: Graph G and parameter β**
  - Directed graph G with **spider traps** and **dead ends**
  - Parameter β
- **Output: PageRank vector r**
  - **Set:** $r_j^{(0)} = \frac{1}{N}, \quad t = 1$
  - **do:**
    - $\forall j$: $r'^{(t)}_j = \sum_{i \to j} \beta \frac{r_i^{(t-1)}}{d_i}$

      $r'^{(t)}_j = 0$ if in-degree of j is 0
    - **Now re-insert the leaked PageRank:**

      $\forall j$: $r_j^{(t)} = r'^{(t)}_j + \frac{1-S}{N}$

      where: $S = \sum_j r'^{(t)}_j$
    - $t = t + 1$
  - **while** $\sum_j \left| r_j^{(t)} - r_j^{(t-1)} \right| > \varepsilon$

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**.

Some Problems with PageRank

- **Measures generic popularity of a page**
  - Will ignore/miss topic-specific authorities
  - **Solution:** Topic-Specific PageRank (**next**)
- **Uses a single measure of importance**
  - Other models of importance
  - **Solution:** Hubs-and-Authorities
- **Susceptible to Link spam**
  - Artificial link topographies created in order to boost page rank
  - **Solution:** TrustRank

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets,
http://www.mmds.org

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
- **Teleport can go to:**
  - **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 pick 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 $\boldsymbol{r_S}$

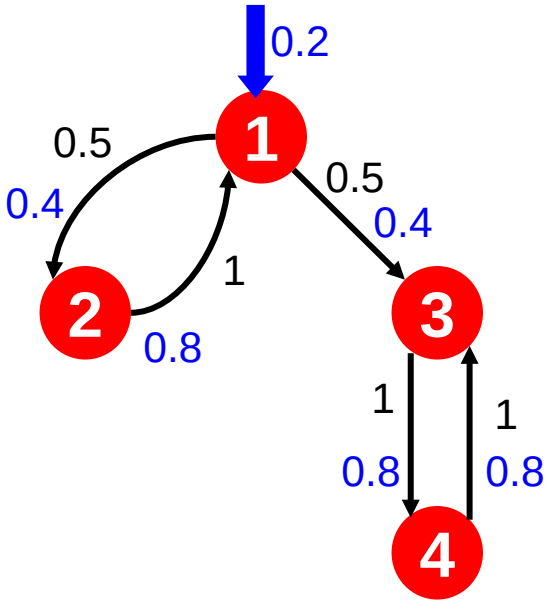
Matrix Formulation

- **To make this work all we need is to update the teleportation part of the PageRank formulation:**

$$A_{ij} = \begin{cases} \beta M_{ij} + (1-\beta)/|S| & \text{if } i \in S \\ \beta M_{ij} + 0 & \text{otherwise} \end{cases}$$

  - A is stochastic!
- We weighted all pages in the teleport set S equally
  - **Could also assign different weights to pages!**
- **Compute as for regular PageRank:**
  - Multiply by M, then add a vector
  - Maintains sparseness

Example: Topic-Specific PageRank



Suppose $S = \{1\}$, $\beta = 0.8$

Node	Iteration				
	0	1	2	…	stable
1	0.25	0.4	0.28		0.294
2	0.25	0.1	0.16		0.118
3	0.25	0.3	0.32		0.327
4	0.25	0.2	0.24		0.261

S={1}, β=0.90:
r=[0.17, 0.07, 0.40, 0.36]
S={1} , β=0.8:
r=[0.29, 0.11, 0.32, 0.26]
S={1}, β=0.70:
r=[0.39, 0.14, 0.27, 0.19]

S={1,2,3,4}, β=0.8:
r=[0.13, 0.10, 0.39, 0.36]
S={1,2,3} , β=0.8:
r=[0.17, 0.13, 0.38, 0.30]
S={1,2} , β=0.8:
r=[0.26, 0.20, 0.29, 0.23]
S={1} , β=0.8:
r=[0.29, 0.11, 0.32, 0.26]

Discovering the Topic Vector S

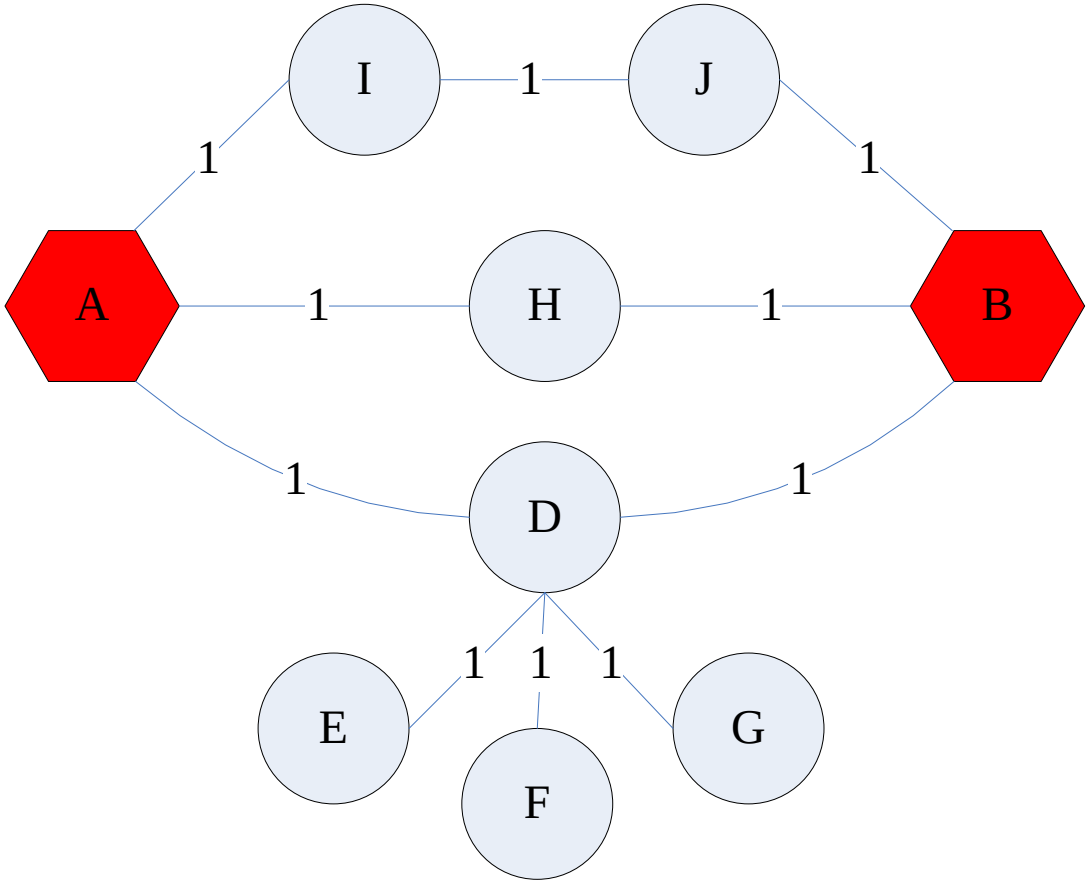
- **Create different PageRanks for different topics**
  - The 16 DMOZ top-level categories:
    - arts, business, sports,…
- **Which topic ranking to use?**
  - User can pick from a menu
  - Classify query into a topic
  - Can use the **context** of the query
    - E.g., query is launched from a web page talking about a known topic
    - History of queries e.g., “basketball” followed by “Jordan”
  - User context, e.g., user’s bookmarks, …

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets,
http://www.mmds.org

Application to Measuring Proximity in Graphs

Random Walk with Restarts: S is a single element

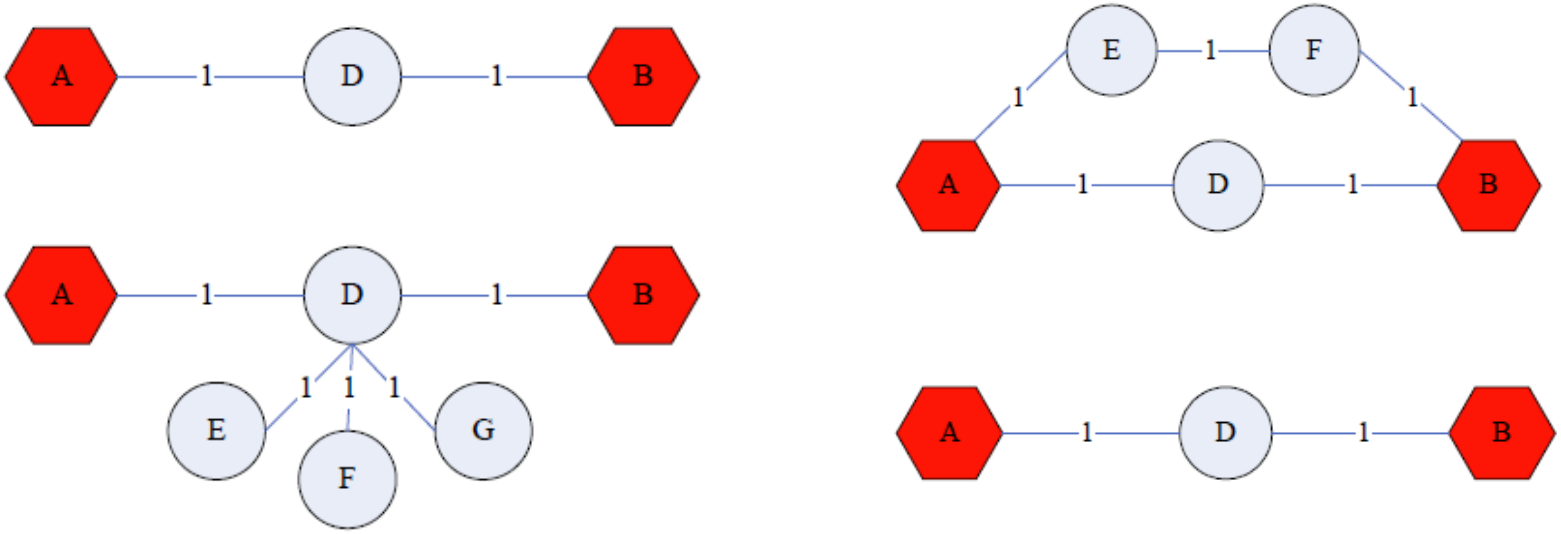
Proximity on Graphs



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

Good proximity measure?

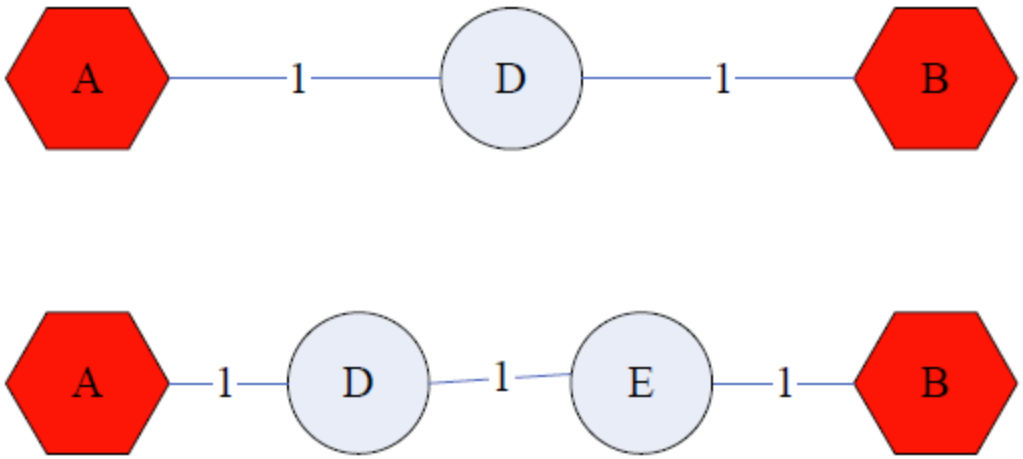
- **Shortest path is not good:**



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

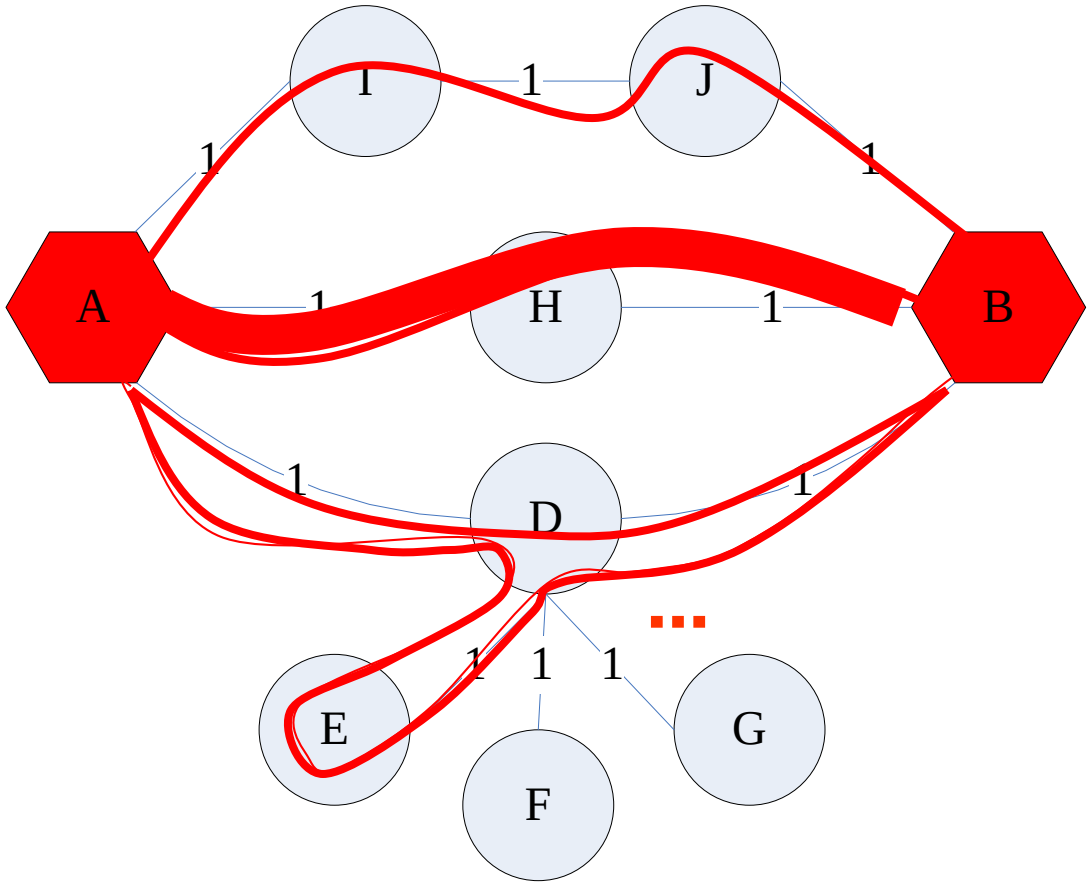
Good proximity measure?

- **Network flow is not good:**



- **Does not punish long paths**

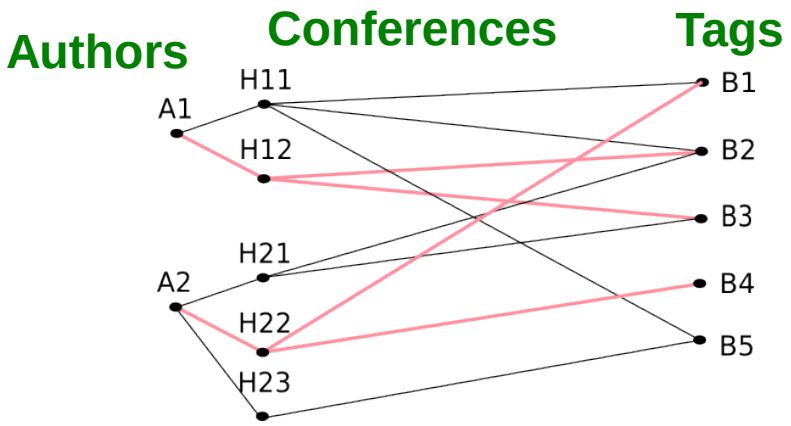
What is good notion of proximity?



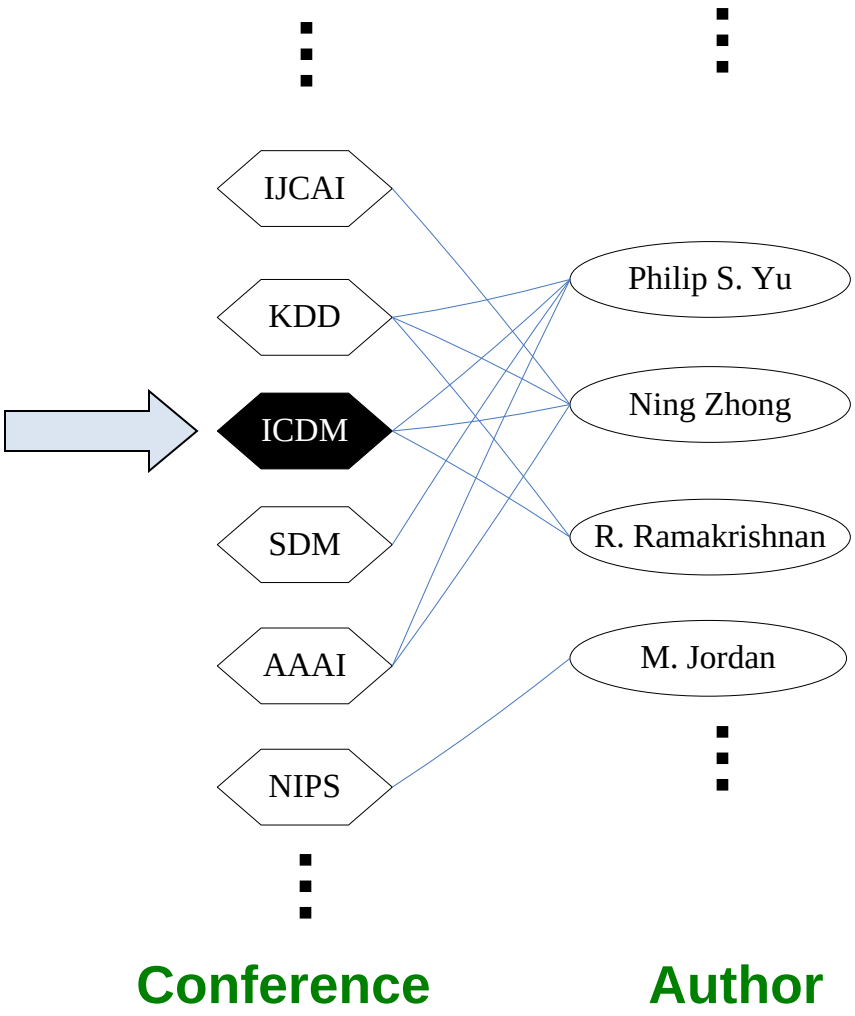
- Multiple connections
- Quality of connection
  - Direct & Indirect connections
  - Length, Degree, Weight…

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
- **Topic Specific 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



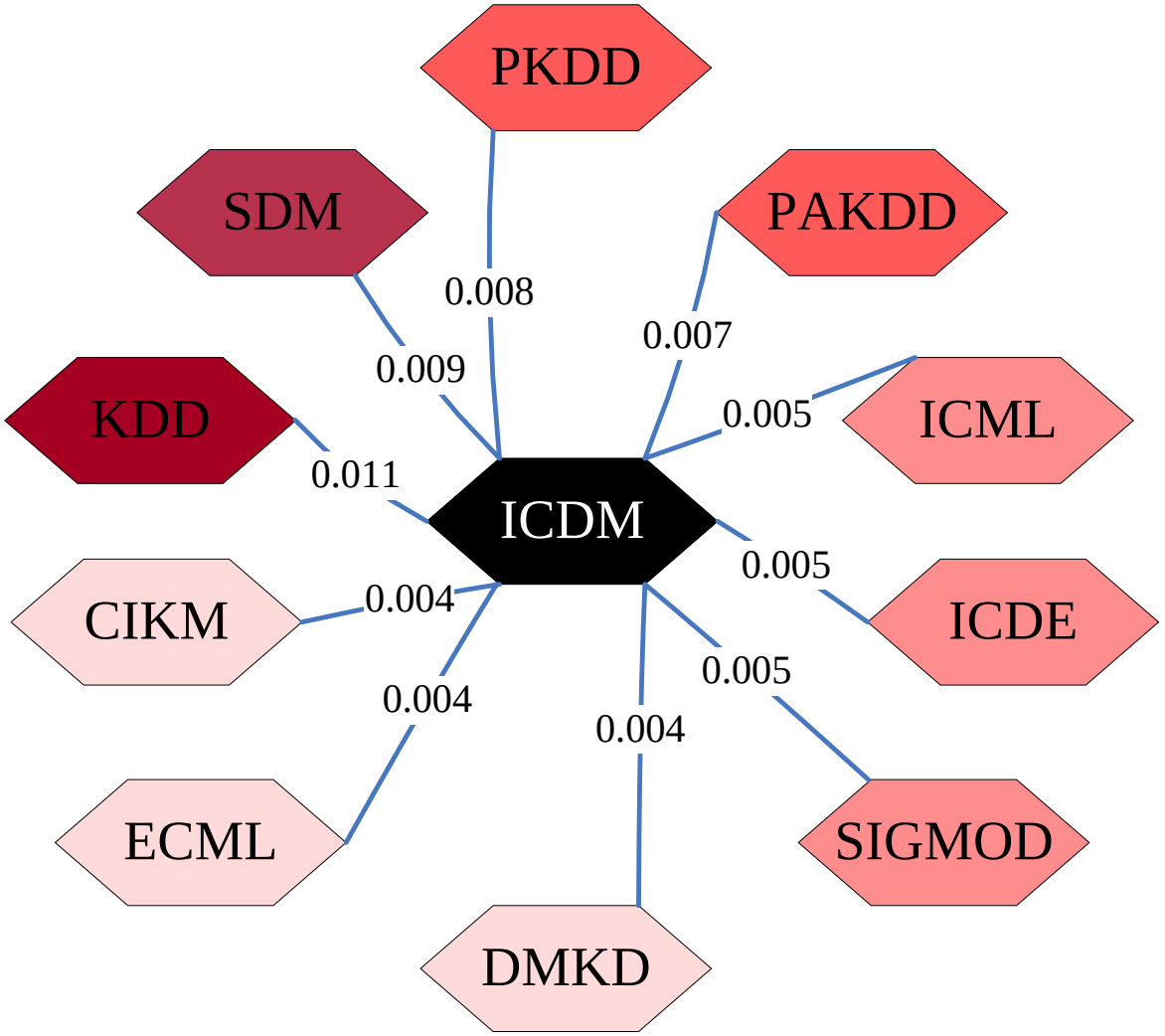
SimRank: Example



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]

BEAR: Block Elimination Approach for Random Walk With Restart on Large Graphs (SIGMOD 2015)

Kijung Shin (CMU)
Jinhong Jung (SNU)
Sael Lee (SUNY Korea)
U Kang (SNU)

Introduction

- ***Random Walk with Restart (RWR)***
  - **Goal:** measures the relevance between two nodes
  - **Properties**: accounts for the global network structure and the multi-faceted relationship between nodes
  - **Applications:** ranking, community detection, link prediction, and anomaly detection
- **Question: How can we compute RWR on large graphs fast, efficiently, and accurately?**

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

Problem Definition

- **Given**: a graph G, a seed node s, and restarting probability c
- **Goal**: find RWR score vector $\vec{\boldsymbol{r}}$ satisfying

$$\vec{\boldsymbol{r}} = (1 - c)\widetilde{\boldsymbol{A}}^{\boldsymbol{T}}\vec{\boldsymbol{r}} + c\vec{\boldsymbol{q}}$$

Input:

- $\widetilde{\boldsymbol{A}} \in \mathbb{R}^{\boldsymbol{n}}$: row-normalized adjacency matrix
- $\vec{\boldsymbol{q}} \in \mathbb{R}^{\boldsymbol{n}}$: query vector where $\vec{\boldsymbol{q}}_{\boldsymbol{s}} = 1$ and $\vec{\boldsymbol{q}}_{\boldsymbol{i}} = 0, \forall i \neq s$
- $c \in \mathbb{R}$: restarting probability

Output:

- $\vec{\boldsymbol{r}} \in \mathbb{R}^{\boldsymbol{n}}$: RWR score vector with regard to node s

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

Previous Method: Inversion (1)

- **Background:** computing RWR boils down to solving a linear system

$$\vec{\boldsymbol{r}} = (1-c)\widetilde{\boldsymbol{A}}^{\boldsymbol{T}}\vec{\boldsymbol{r}} + c\vec{\boldsymbol{q}}$$

$$\Leftrightarrow \left(\boldsymbol{I} - (1-c)\widetilde{\boldsymbol{A}}^{\boldsymbol{T}}\right)\vec{\boldsymbol{r}} = c\vec{\boldsymbol{q}}$$

$$\Leftrightarrow \boldsymbol{H}\vec{\boldsymbol{r}} = c\vec{\boldsymbol{q}}$$

where $\boldsymbol{H} = \boldsymbol{I} - (1-c)\widetilde{\boldsymbol{A}}^{\boldsymbol{T}}$

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

Previous Method: Inversion (2)

- **Preprocess phase** (one-time cost): compute $\boldsymbol{H^{-1}}$
- **Query phase** (repetitive cost): compute $\vec{\boldsymbol{r}}$

$$\vec{\boldsymbol{r}} = \boldsymbol{H^{-1}}(c\vec{\boldsymbol{q}})$$

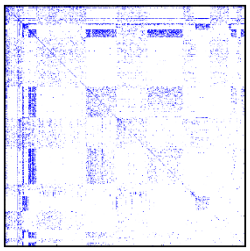
- **Advantages:**
  - Fast query speed (one matrix-vector multiplication)
- **Disadvantages:**
  - Inverting $\boldsymbol{H}$ takes too long
  - $\boldsymbol{H^{-1}}$ is usually too dense to fit in memory

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

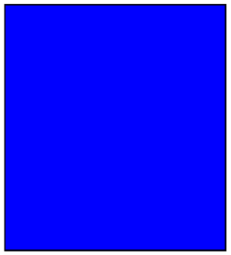
Other Preprocessing methods (1)

- Replace $\boldsymbol{H^{-1}}$ with sparser matrices by reordering and decomposing $\boldsymbol{H}$
- Still expensive in terms of space and/or inaccurate

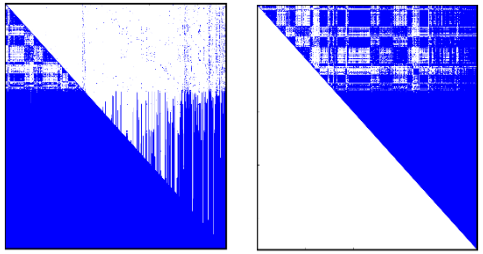
Input graph	(1) Inversion	(2) QR (Fujiwara et al. 12)
#nz=0.1M	Exact, #nz=527M	Exact, #nz=428M



H



H^{-1}



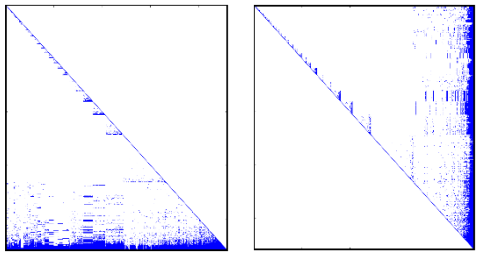
$Q^{-1}(= Q^T)$ R^{-1}

Sparsity pattern of preprocessed matrices on the Routing dataset

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

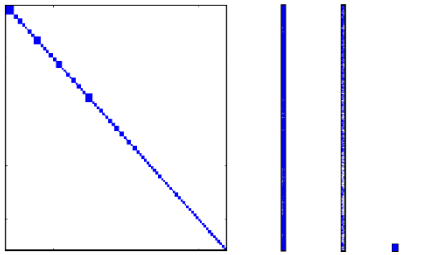
Other Preprocessing methods (2)

(3) LU (Fujiwara et al. 12)
Exact, #nz=10M



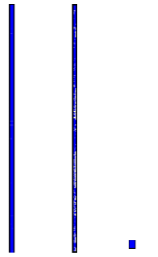
L^{-1} U^{-1}

(4) B_LIN (Tong et al. 07)
Approx, #nz=8M



A_1^{-1} U V $\tilde{\Lambda}$

(5) NB_LIN
Approx, #nz=3M



U V $\tilde{\Lambda}$

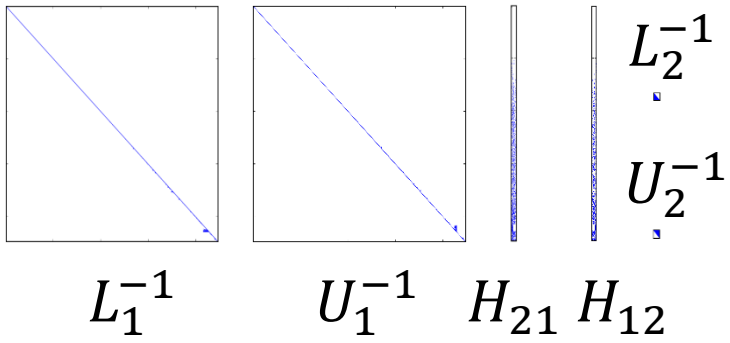
Sparsity pattern of preprocessed matrices on the Routing dataset

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

Proposed Method: BEAR (1)

- We propose ***BEAR***, a fast, space-efficient, and accurate RWR computation method

(6) BEAR-Exact (Proposed)
Exact, #nz=0.4M



Sparsity pattern of preprocessed matrices on the Routing dataset

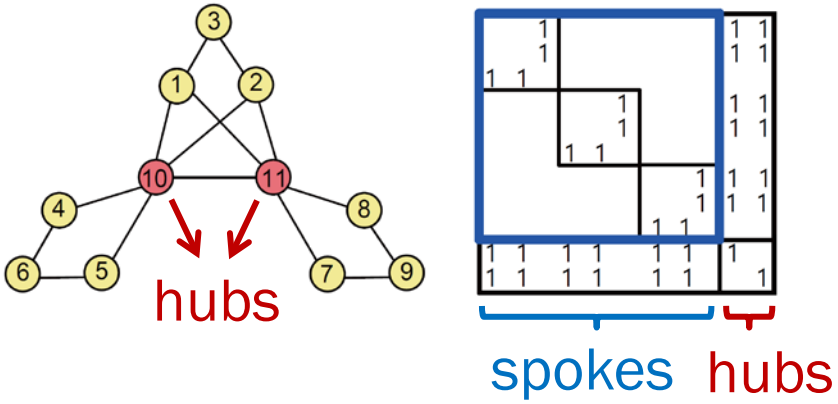
Proposed Method: BEAR (2)

- ***BEAR*** offers two versions
  - ***BEAR-Exact***: guarantees accuracy
  - ***BEAR-Approx***: fast and space-efficient but allows small error
- ***BEAR*** consists of the two phases
  - **Preprocessing phase** (one-time cost): partitions the adjacency matrix into submatrices and precomputes several matrices using the submatrices
  - **Query phase** (repetitive cost): compute RWR scores accurately from precomputed matrices

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

Preprocessing Phase

1. Reordering



2. Partitioning

$$\begin{bmatrix} H \\ (=I-(1-c)\tilde{A}^T) \end{bmatrix} \Rightarrow \begin{bmatrix} {}_1H_{11} & & & H_{12} \\ & {}_2H_{11} & & \\ & & {}_3H_{11} & \\ H_{21} & & & H_{22} \end{bmatrix}$$

3. Schur Complement

$$\begin{bmatrix} S \end{bmatrix} \Leftarrow \begin{bmatrix} H_{22} \end{bmatrix} - \begin{bmatrix} H_{21} \end{bmatrix} \begin{bmatrix} H_{11}^{-1} \end{bmatrix} \begin{bmatrix} H_{12} \end{bmatrix}$$

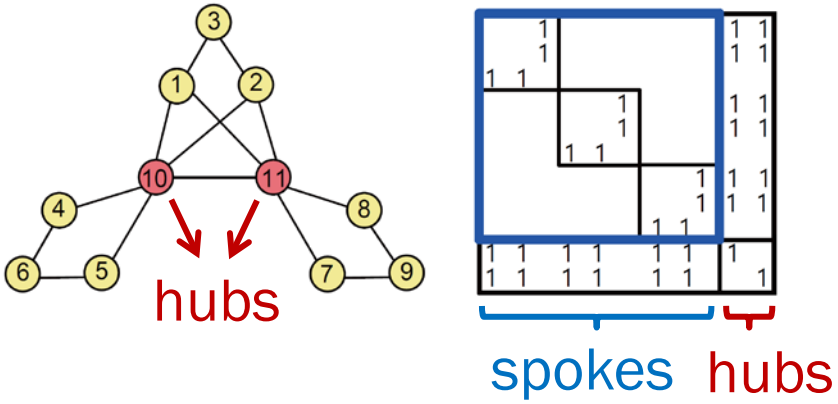
4. Inverting

$$\begin{bmatrix} {}_1H_{11}^{-1} & & & \\ & {}_2H_{11}^{-1} & & \\ & & {}_3H_{11}^{-1} & \end{bmatrix} \begin{bmatrix} S^{-1} \end{bmatrix}$$

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

Preprocessing Phase

1. Reordering



2. Partitioning

$$\begin{bmatrix} H \\ (=I-(1-c)\tilde{A}^T) \end{bmatrix} \Rightarrow \begin{bmatrix} {}_1H_{11} & & & H_{12} \\ & {}_2H_{11} & & \\ & & {}_3H_{11} & \\ H_{21} & & & H_{22} \end{bmatrix}$$

3. Schur Complement

$$\begin{bmatrix} S \end{bmatrix} \Leftarrow \begin{bmatrix} H_{22} \end{bmatrix} - \begin{bmatrix} H_{21} \end{bmatrix} \begin{bmatrix} H_{11}^{-1} \end{bmatrix} \begin{bmatrix} H_{12} \end{bmatrix}$$

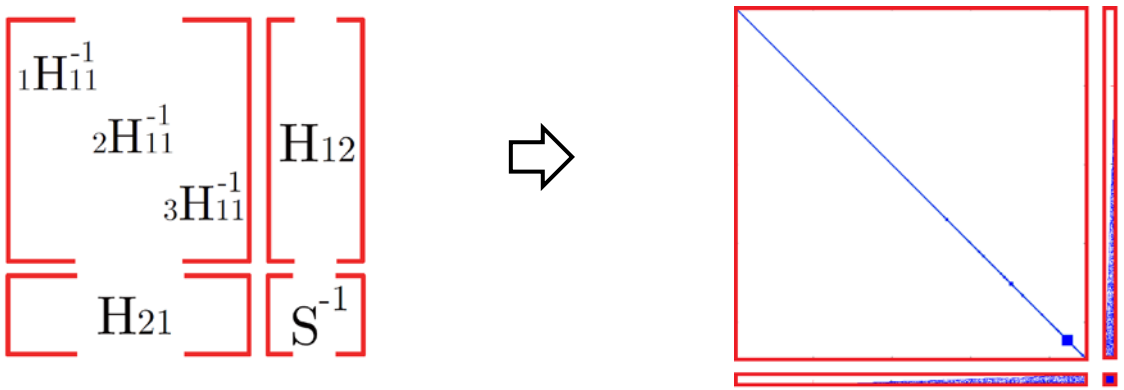
4. Inverting

$$\begin{bmatrix} {}_1H_{11}^{-1} & & & \\ & {}_2H_{11}^{-1} & & \\ & & {}_3H_{11}^{-1} & \end{bmatrix} \begin{bmatrix} S^{-1} \end{bmatrix}$$

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

Preprocessing Phase: Output

- Precomputed matrices are small or composed of small diagonal blocks
- Require little storage



(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

Query Phase

- Given query vector $\vec{q}$, compute RWR score vector $\vec{r}$ using the precomputed matrices
- Theorem (**Block Elimination**): This equation exactly computes RWR scores

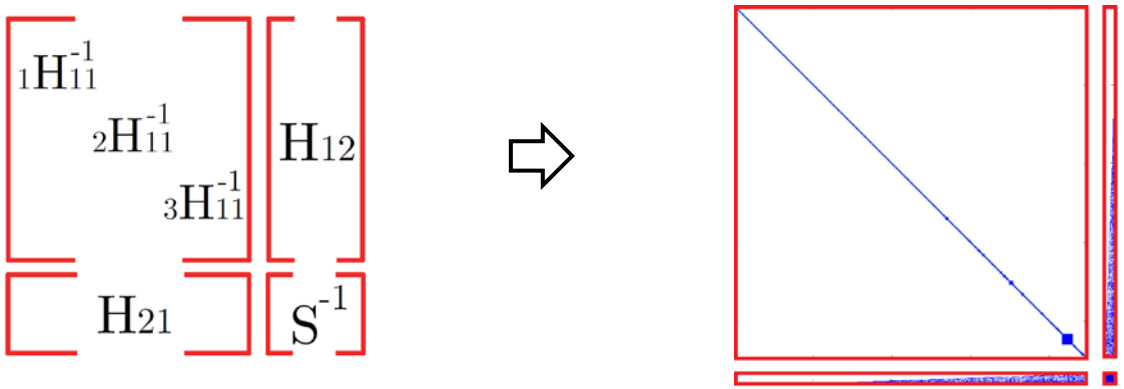
$$\begin{bmatrix} r \end{bmatrix} \Leftarrow \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \Leftarrow \begin{bmatrix} H_{11}^{-1} \end{bmatrix} \left(\begin{bmatrix} q_1 \end{bmatrix} - \begin{bmatrix} H_{12} \end{bmatrix} \begin{bmatrix} r_2 \end{bmatrix} \right)$$

$$\begin{bmatrix} S^{-1} \end{bmatrix} \begin{bmatrix} q_2 \end{bmatrix} - \begin{bmatrix} H_{21} \end{bmatrix} \begin{bmatrix} H_{11}^{-1} \end{bmatrix} \begin{bmatrix} q_1 \end{bmatrix}$$

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

BEAR-Approx

- Remove small entries in precomputed matrices
- Fast and space-efficient but allows small error



Experimental Settings

- **Machine**: single PC with with a 4-core CPU and 16GB memory
- **Datasets**: large-scale real-world network data

dataset	*#nodes*	*#edges*
Routing	22,963	48,436
Co-author	31,163	120,029
Trust	131,828	841,372
Email	265,214	420,045
Web-Stan	281,903	2,312,497
Web-Notre	325,729	1,497,134
Web-BS	685,230	7,600,595
Talk	2,394,385	5,021,410
Citation	3,774,768	16,518,948

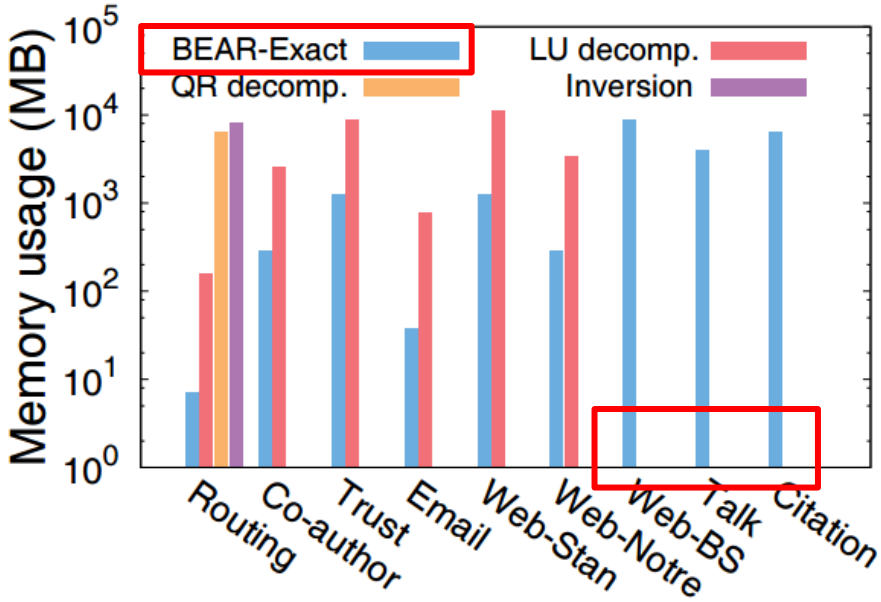
Competitors

- **Exact methods**
  - Inversion
  - Iterative method
  - LU decomp. (Fujiwara et al., 2012)
  - QR decomp. (Fujiwara et al., 2012)
- **Approximate methods**
  - BLIN, NB_LIN (Tong et al., 2008)
  - RPPR, BRPPR (Gleich et al., 2006)

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

Q1. Space Efficiency

- Q1. How much memory space does **BEAR-Exact** require for their precomputed matrices?



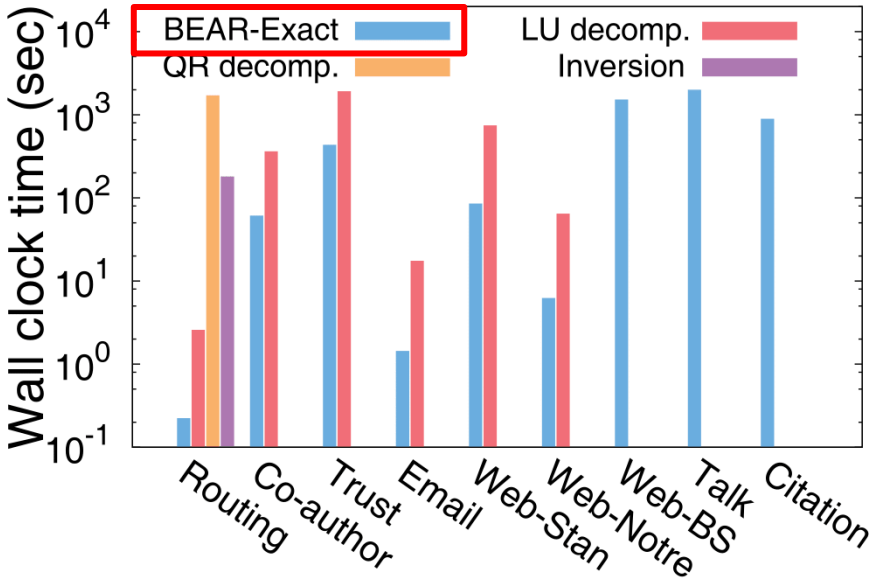
Space for preprocessed data

Up to 22x less memory space than competitors

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

Q2. Preprocessing Time

- How long does the preprocessing phase of **BEAR-Exact** take?



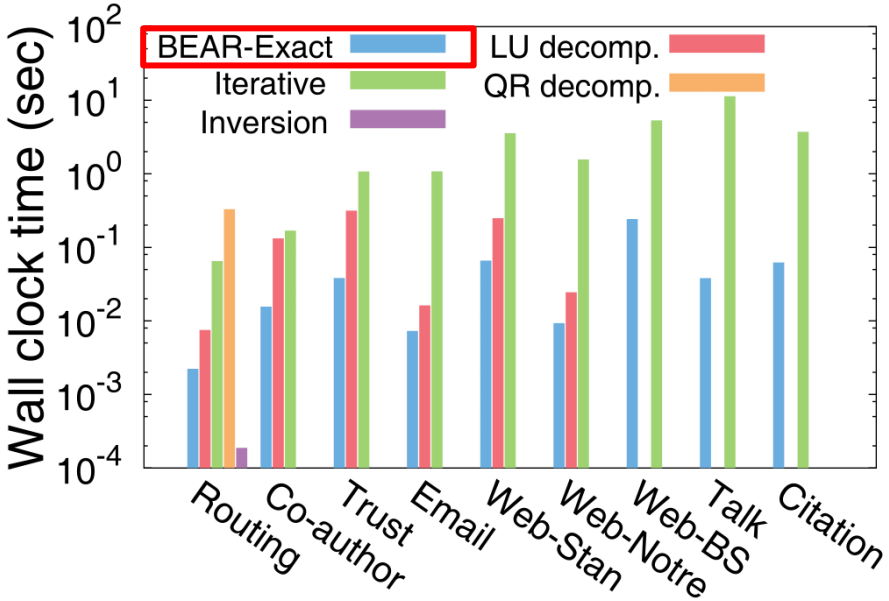
Preprocessing time of exact methods

Up to 12x less preprocessing time than other methods

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

Q3. Query Time

- How long does the query phase of **BEAR-Exact** take?



Query time of exact methods

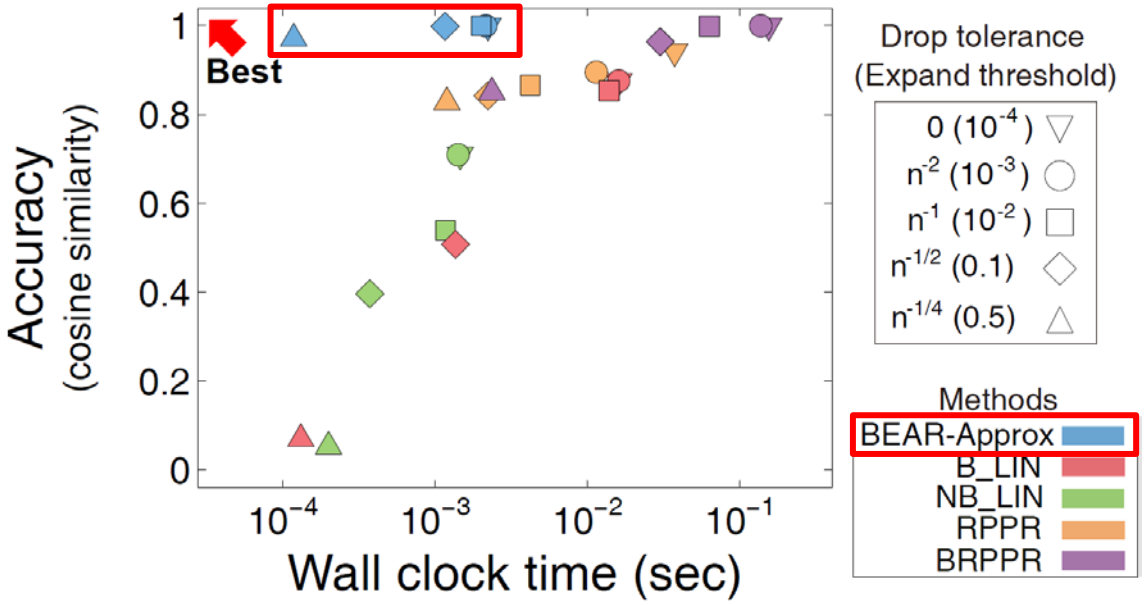
Up to 8x less query time than LU decomp.

Up to 300x less query time than Iterative method

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

Q4. Speed vs Accuracy

- Does **BEAR-Approx** provide a better trade-off between speed and accuracy than other methods?

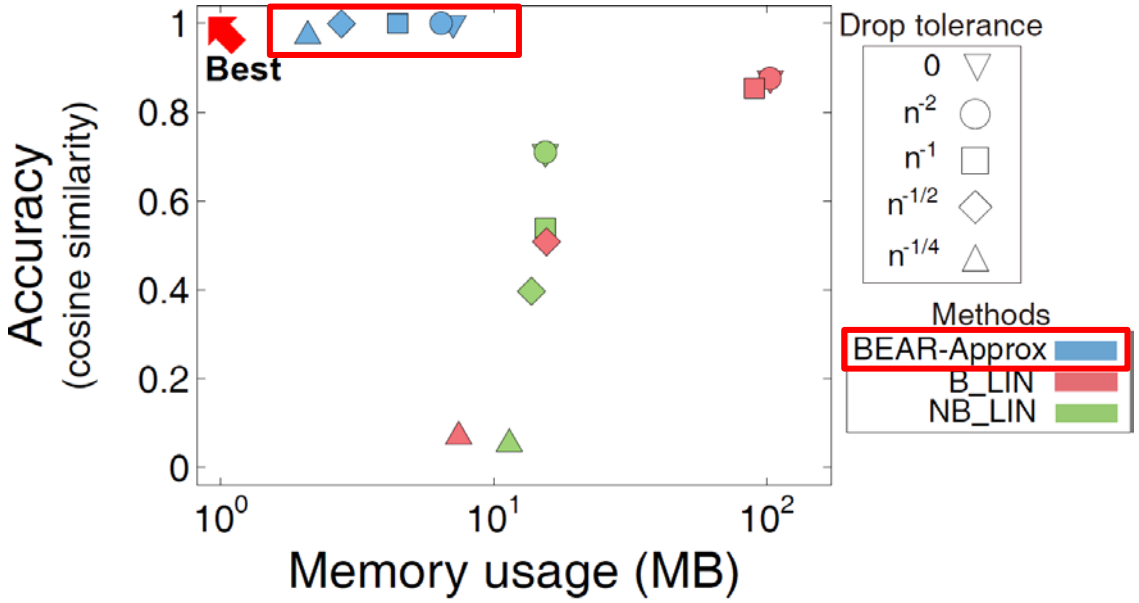


Query speed v.s. Accuracy on the Routing dataset

(Shin, Sael, Kang et al. BEAR SIGMOD 2015)

Q5. Space vs Accuracy

- Does **BEAR-Approx** provide a better trade-off between space and accuracy than other methods?



Space for preprocessed data v.s. Accuracy on the Routing dataset