Introduction to Data Mining

Lecture #10: Link Analysis-2

U Kang
Seoul National University

Outline

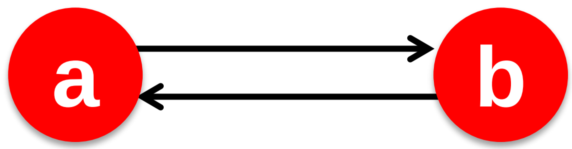
- ☐ **PageRank: Google Formulation**
- ☐ Computing PageRank
- ☐ Topic-Specific PageRank
- ☐ Measuring Proximity in Graphs

PageRank: Three Questions

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

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

Does this converge?



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

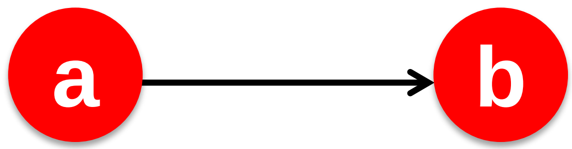
- **Example:**

=

r_a	1	0	1	0
r_b	0	1	0	1

Iteration 0, 1, 2, …

Does it converge to what we want?



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

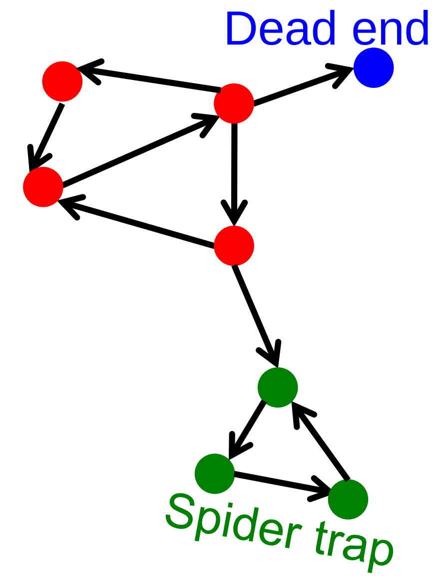
- **Example:**

=

r_a	1	0	0	0
r_b	0	1	0	0

Iteration 0, 1, 2, …

PageRank: Problems



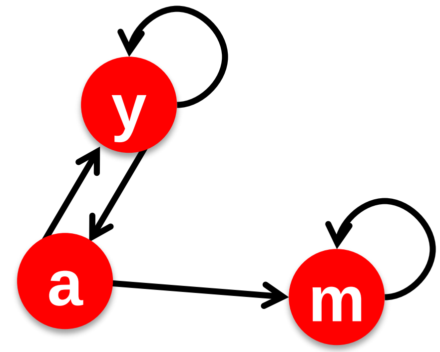
2 problems:

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

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

Problem: Spider Traps

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



m is a spider trap

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	1

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

- **Example:**

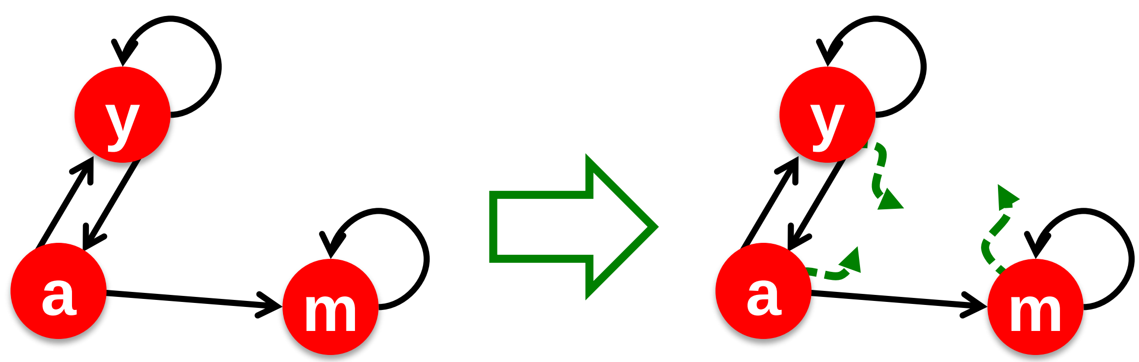
Iteration 0, 1, 2, …

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

All the PageRank score gets “trapped” in node m.

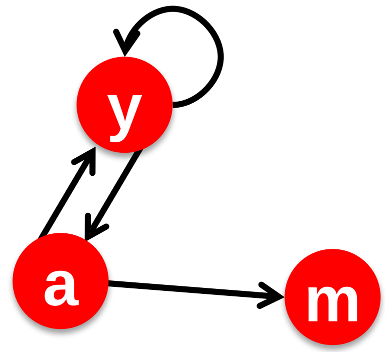
Solution: Teleports!

- **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:**

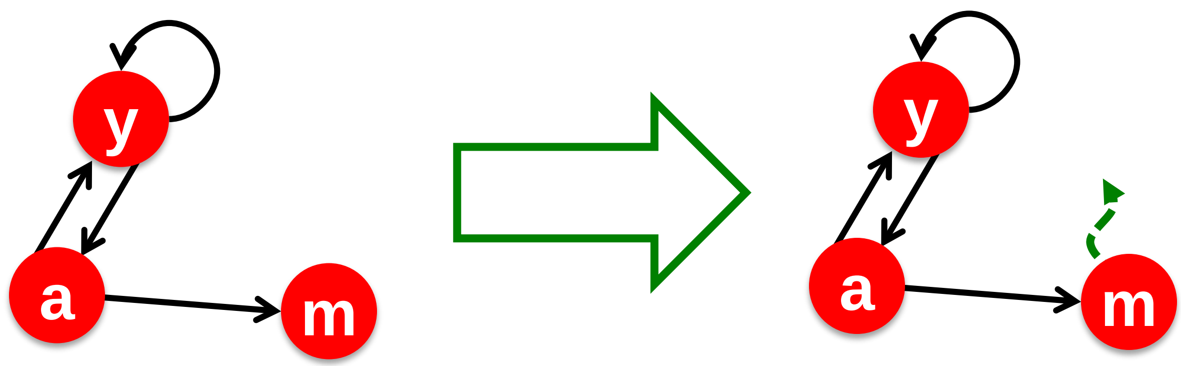
Iteration 0, 1, 2, …

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

Here the PageRank “leaks” out since the matrix is not column 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 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]

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

d_i … out-degree of node i

This formulation assumes that M has no dead ends. We can either preprocess matrix M to remove all dead ends or explicitly follow random teleport links with probability 1.0 from dead-ends.

The Google Matrix

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

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

- **In matrix form:**
  - $r \quad = \beta M r + (1-\beta)[\frac{1}{N}]_{N\times N} r$

$$= \underline{\{\beta M + (1-\beta)[\frac{1}{N}]_{N\times N}\}} r$$

This is called the "Google Matrix"

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

1/3	1/3	1/3
1/3	1/3	1/3
1/3	1/3	1/3

E.g., for N=3

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*:**

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

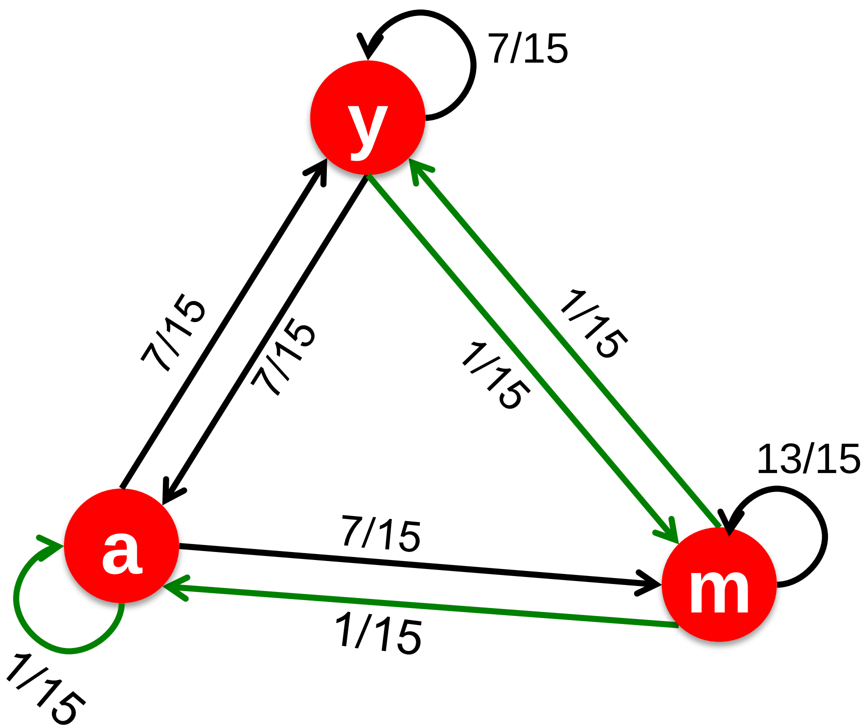
$$A = \beta\, M + (1-\beta)\left[\frac{1}{N}\right]_{N \times N}$$

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

  And the Power method still works!

- **What is β ?**
  - In practice $\beta = 0.8, 0.9$ (make ~5 steps on avg., jump)

Random Teleports (β = 0.8)



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

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

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

Outline

- ☑ PageRank: Google Formulation
- ☐ **Computing PageRank**
- ☐ Topic-Specific PageRank
- ☐ Measuring Proximity in Graphs

Computing Page Rank

- **Key step is matrix-vector multiplication**
  - $\boldsymbol{r}^{new} = \boldsymbol{A} \cdot \boldsymbol{r}^{old}$
- Easy if we have enough main memory to hold **A**, $\mathbf{r}^{old}$, $\mathbf{r}^{new}$
- **Say N = 1 billion pages**
  - We need 4 bytes for each entry (say)
  - Total 2 billion entries for 2 vectors($\mathbf{r}^{old}$, $\mathbf{r}^{new}$): ~ 8GB
  - Matrix **A** has N^2 entries
    - $N^2 = 10^{18}$ (1000 Peta) is a large number!
    - We need to exploit sparsity of M

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

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

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

Sparse Matrix Formulation

- $\boldsymbol{r} = \boldsymbol{A} \cdot \boldsymbol{r},$ **where** $A = \beta\, M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$
- **Main idea: do not construct A explicitly**
- **Specifically:**
  - $\boldsymbol{r} = \boldsymbol{\beta}\, \boldsymbol{M} \cdot \boldsymbol{r} + (\boldsymbol{1} - \boldsymbol{\beta})\left[\frac{\boldsymbol{1}}{\boldsymbol{N}}\right]_{\boldsymbol{N}\times\boldsymbol{N}} \boldsymbol{r}$
  - $= \boldsymbol{\beta}\, \boldsymbol{M} \cdot \boldsymbol{r} + (\boldsymbol{1} - \boldsymbol{\beta})\left[\frac{\boldsymbol{1}}{\boldsymbol{N}}\right]_{\boldsymbol{N}}$

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

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

Sparse Matrix Formulation

- 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!**
  - 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-β)/N** to each entry in r^{new}
    - **Note if M contains dead-ends then $\sum_j r_j^{new} < 1$ and we also have to renormalize r^{new} so that it sums to 1**

PageRank: The Complete Algorithm

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

      $\forall \boldsymbol{j}$: $\boldsymbol{r}_j^{new} = \boldsymbol{r'}_j^{new} + \frac{\mathbf{1-S}}{\boldsymbol{N}}$ **where:** $S = \sum_j r'^{new}_j$
    - $\boldsymbol{r}^{old} = \boldsymbol{r}^{new}$

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

- **Encode sparse matrix using only nonzero entries**
  - Space proportional roughly to number of links
  - Assuming N = 1 billion,
    10N edges would require 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:**

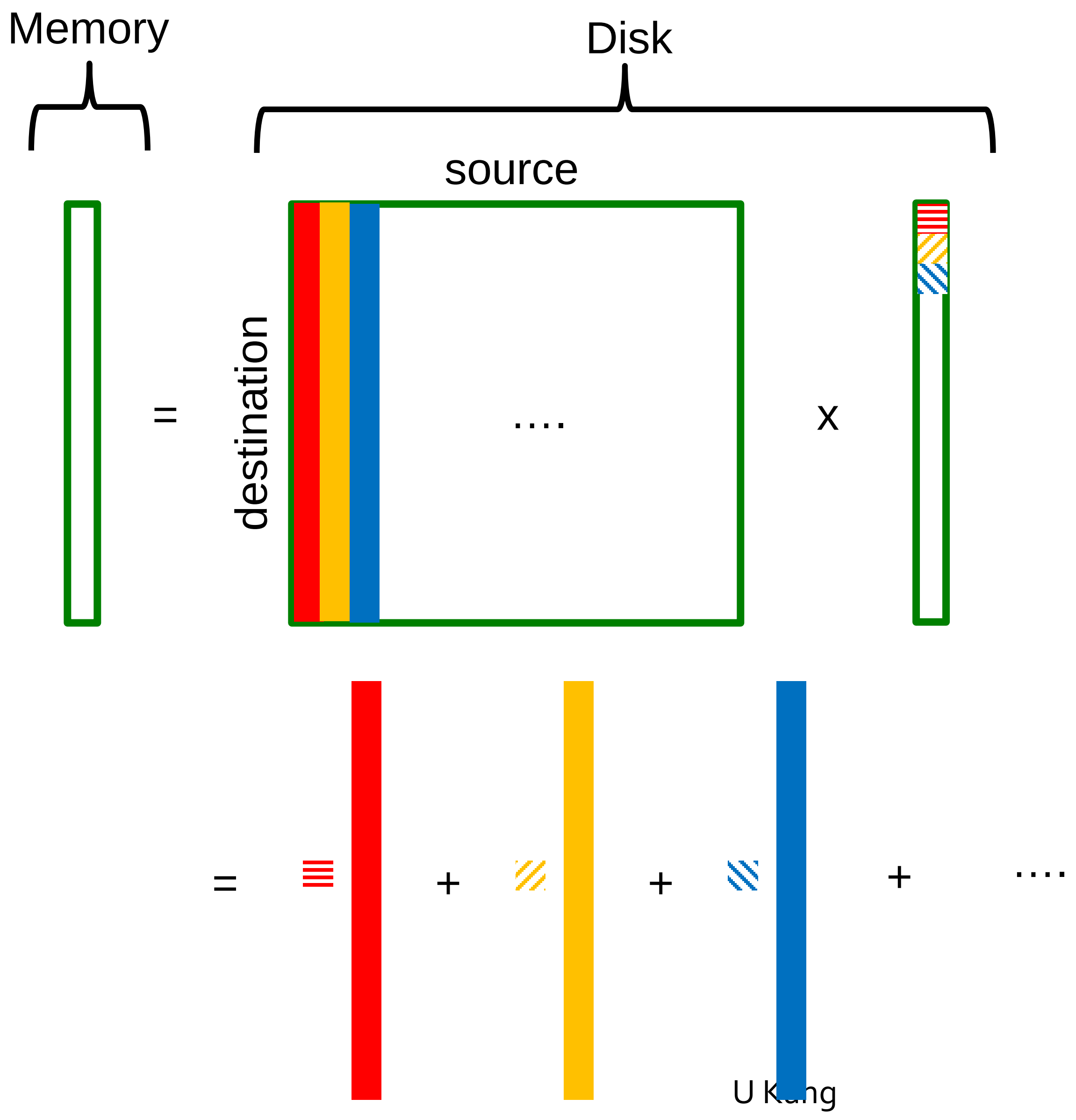
> **Initialize** all entries of $\mathbf{r}^{new} = \mathbf{(1-\beta) / N}$
> For each page i (of out-degree d_i):
>   Read into memory: i, d_i, $dest_1, \ldots, dest_{d_i}, r^{old}(i)$
>   For $\mathbf{j = 1 \ldots d_i}$
> $\mathbf{r^{new}(dest_j) \mathrel{+}= \beta\, r^{old}(i) / d_i}$

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

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

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

Basic Algorithm: Update Step



Analysis

- **Assume enough RAM to fit $\boldsymbol{r}^{new}$ into memory**
  - Store $\boldsymbol{r}^{old}$ and matrix $\boldsymbol{M}$ on disk
- **In each iteration, we have to:**
  - Read $\boldsymbol{r}^{old}$ and $\boldsymbol{M}$
  - Write $\boldsymbol{r}^{new}$ back to disk
  - **Cost (disk I/O) per iteration of Power method:**
    $= 2|\boldsymbol{r}| + |\boldsymbol{M}|$

- **Question:**
  - What if we could not even fit $\boldsymbol{r}^{new}$ in memory?

Block-based Update Algorithm

r^{new}

0
1

2
3

4
5

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

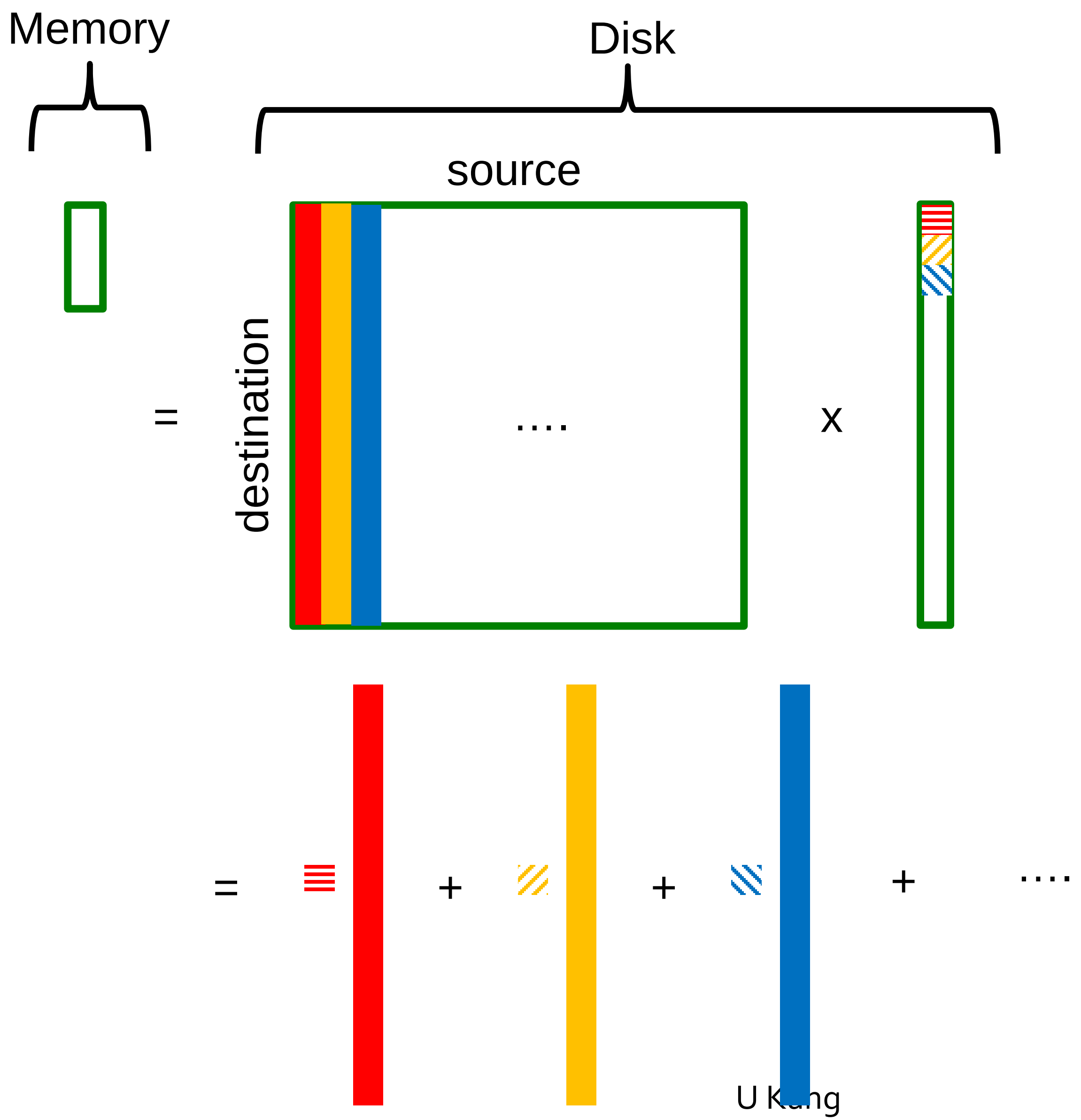
M

r^{old}

0
1
2
3
4
5

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

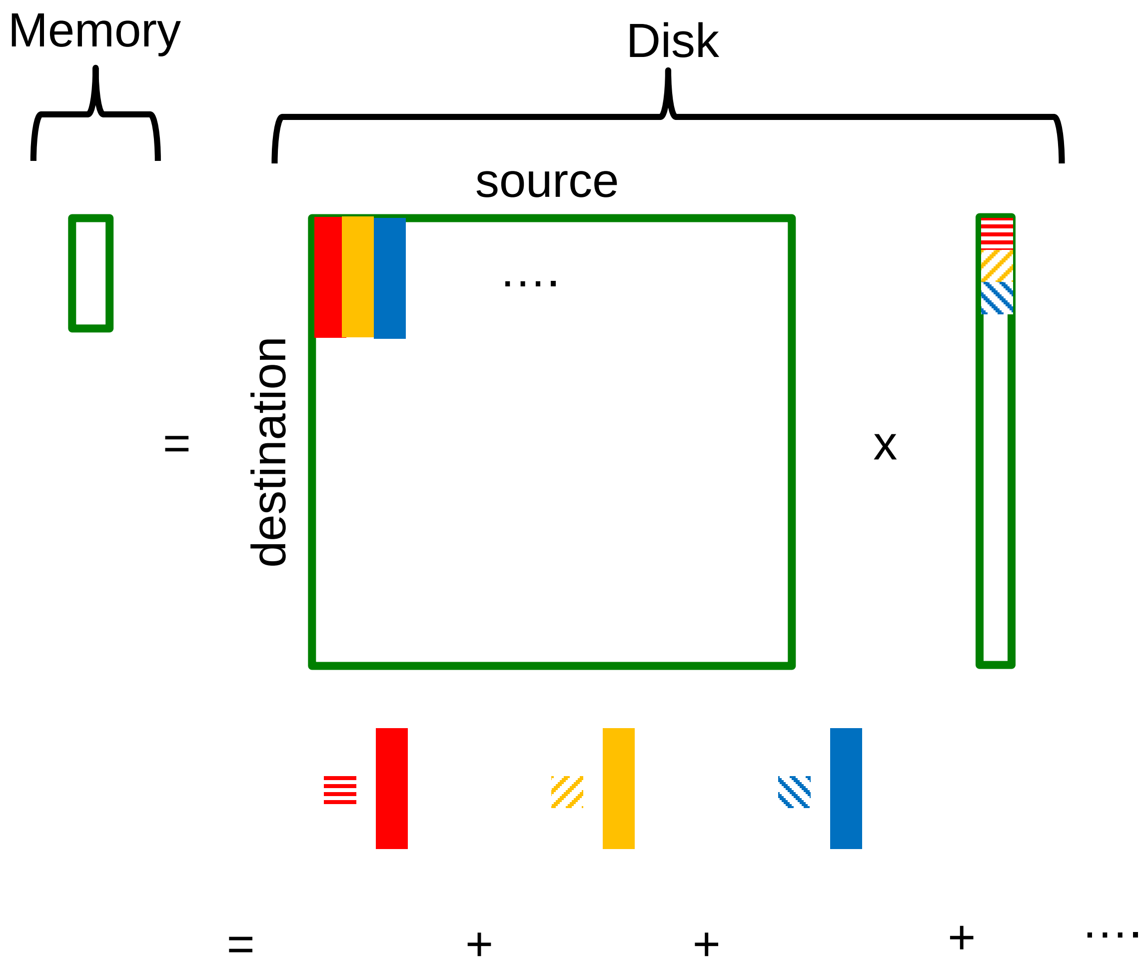
Block-based Update Algorithm



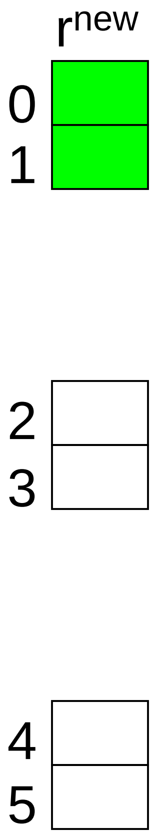
Analysis of Block Update

- **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



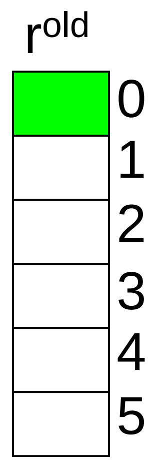
Block-Stripe Update Algorithm



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 Analysis

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

Limitations in Page Rank

- **Measures generic popularity of a page**
  - Biased against 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

Outline

- ☑ PageRank: Google Formulation
- ☑ Computing PageRank
- ☐ **Topic-Specific PageRank**
- ☐ Measuring Proximity in Graphs

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 “Jaguar” wants different pages depending on whether you are interested in animal, car, or operating system

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 picks a page from a set $\boldsymbol{S}$
  - $\boldsymbol{S}$ contains only pages that are relevant to the topic
    - E.g., Yahoo or DMOZ pages for a given topic/query
  - For each teleport set $\boldsymbol{S}$, we get a different vector $\boldsymbol{r_S}$

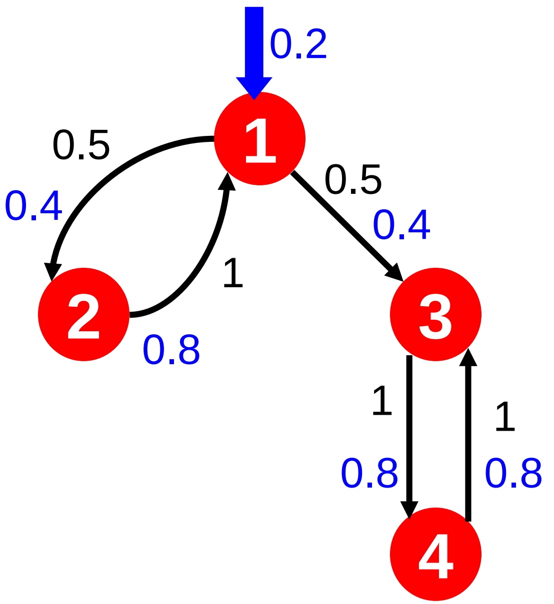
Matrix Formulation

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

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

  - ***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}**, ***β*** **= 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]

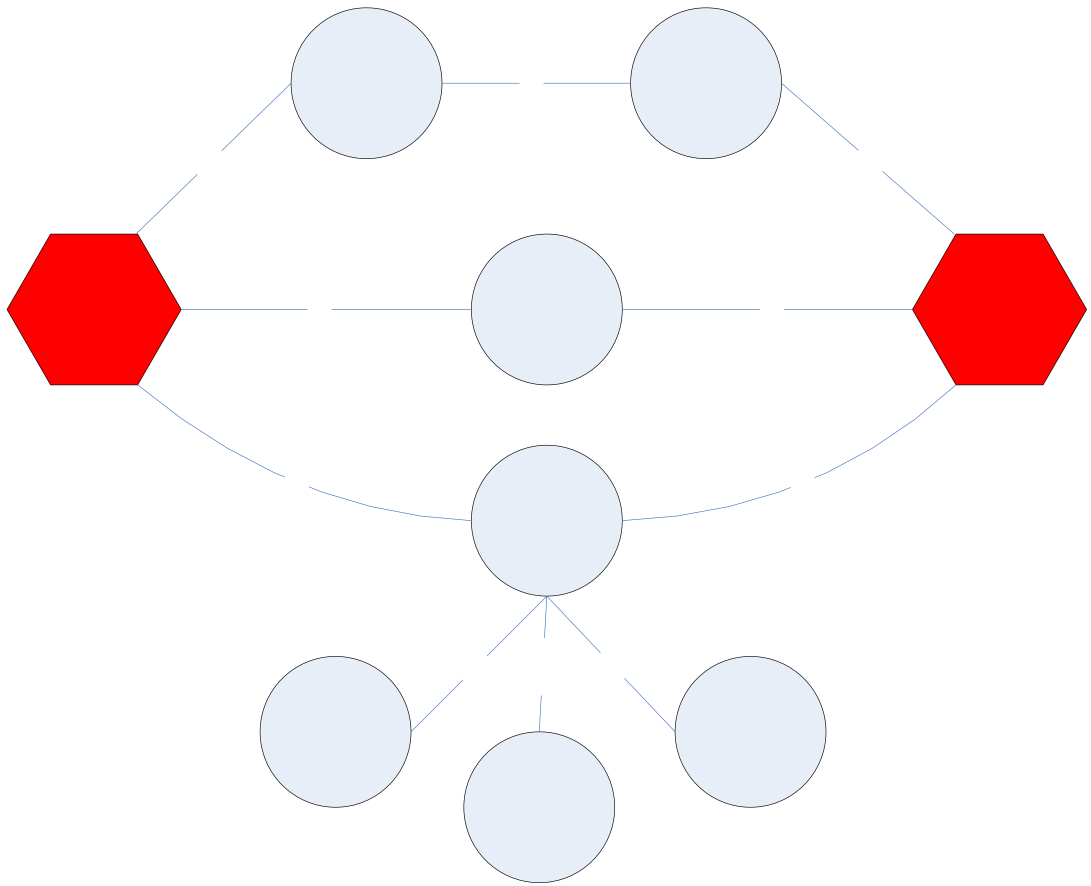
Discovering the Topic Vector S

- **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, …

Outline

- ☑ PageRank: Google Formulation
- ☑ Computing PageRank
- ☑ Topic-Specific PageRank
- ☐ **Measuring Proximity in Graphs**

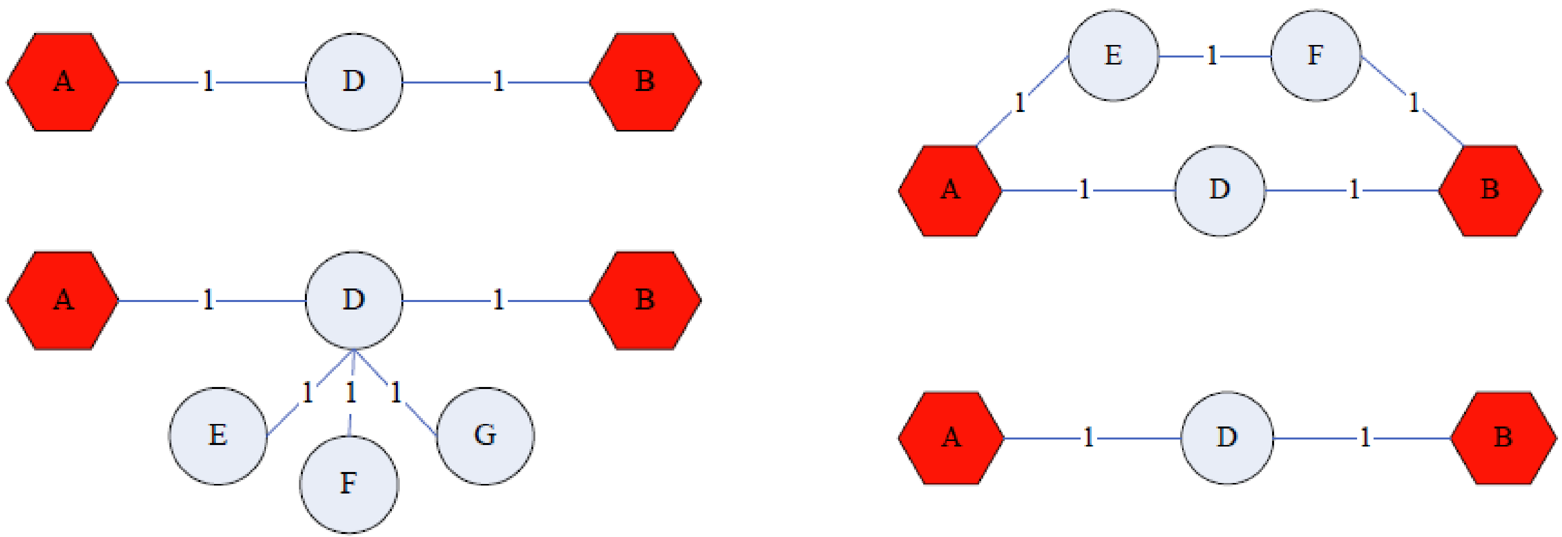
Proximity on Graphs



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

Good proximity measure?

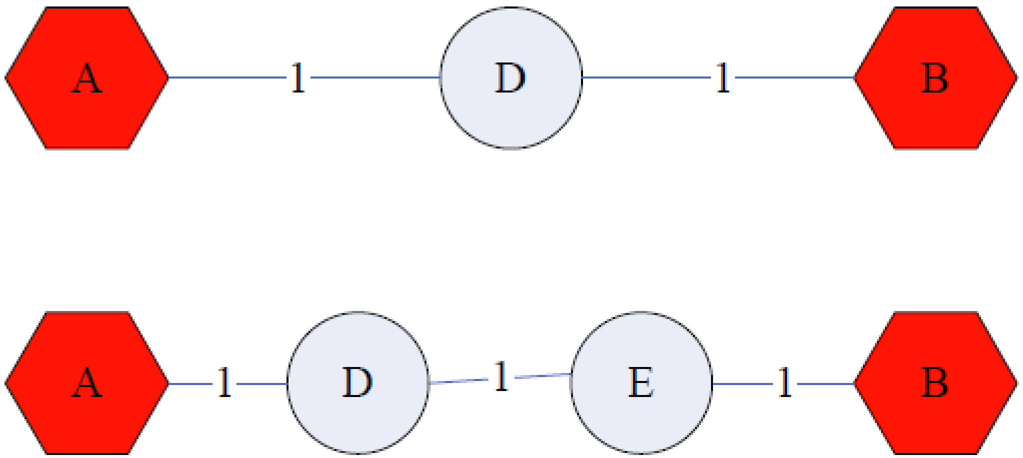
- **Shortest path is not good:**



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

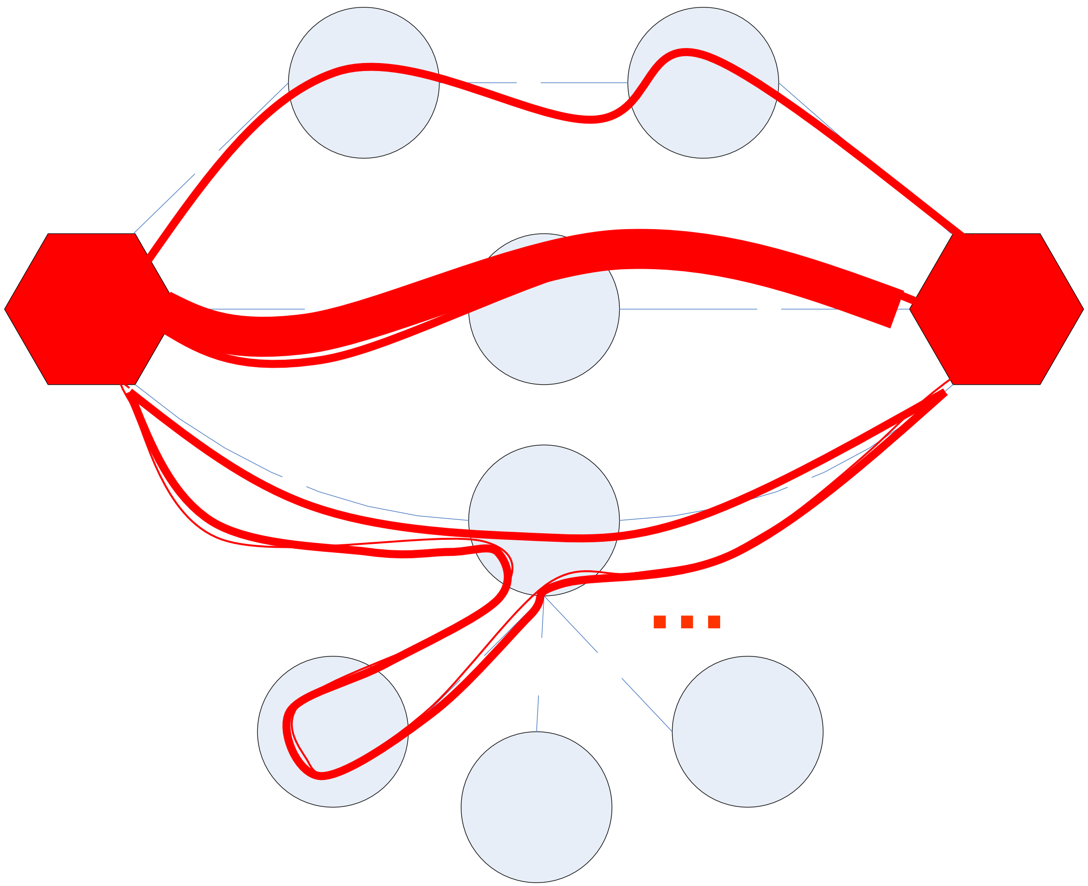
Good proximity measure?

- **Network flow is not good:**



- **Does not punish long paths**

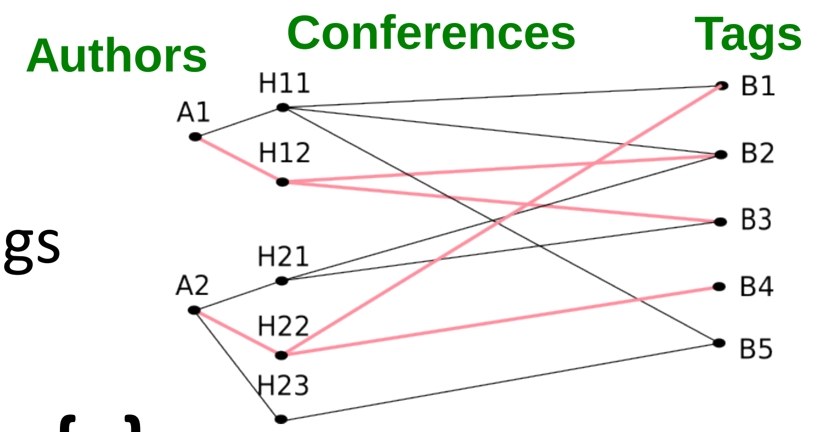
What is good notion of proximity?



- Multiple connections
- Quality of connection
  - Length, Degree, Weight…

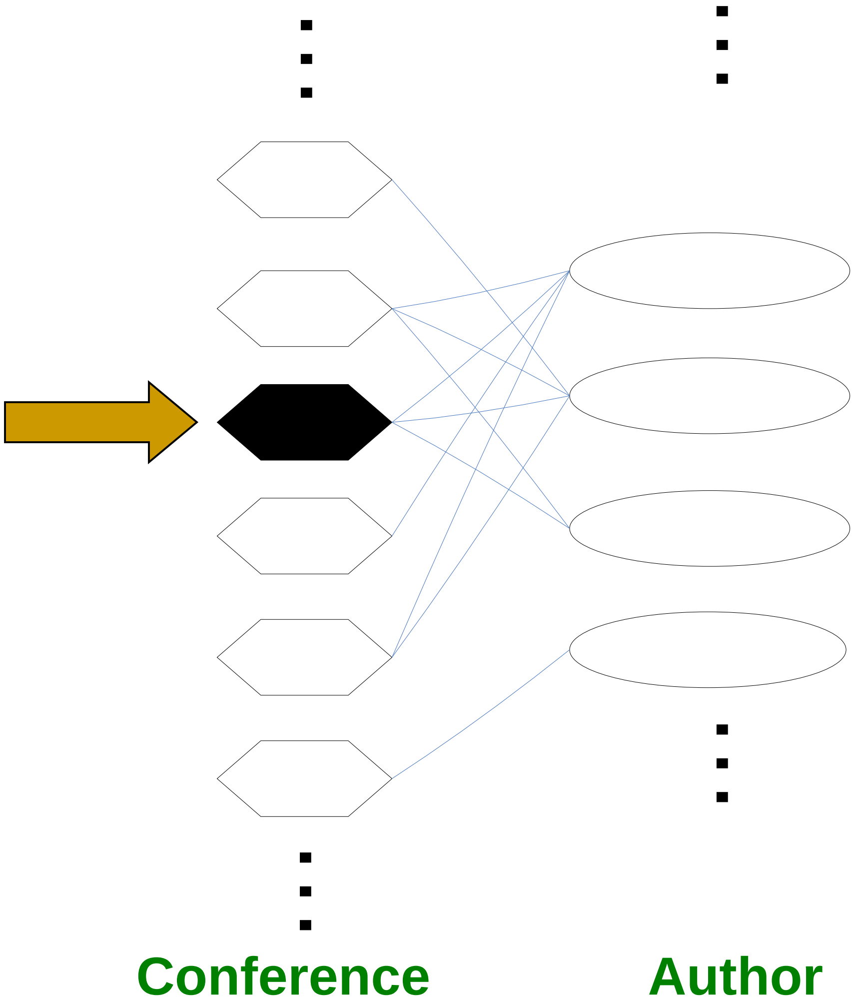
Random Walk with Restart: Idea

- **RWR:** Random walks from a **fixed node**
- **E.g.,** ***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

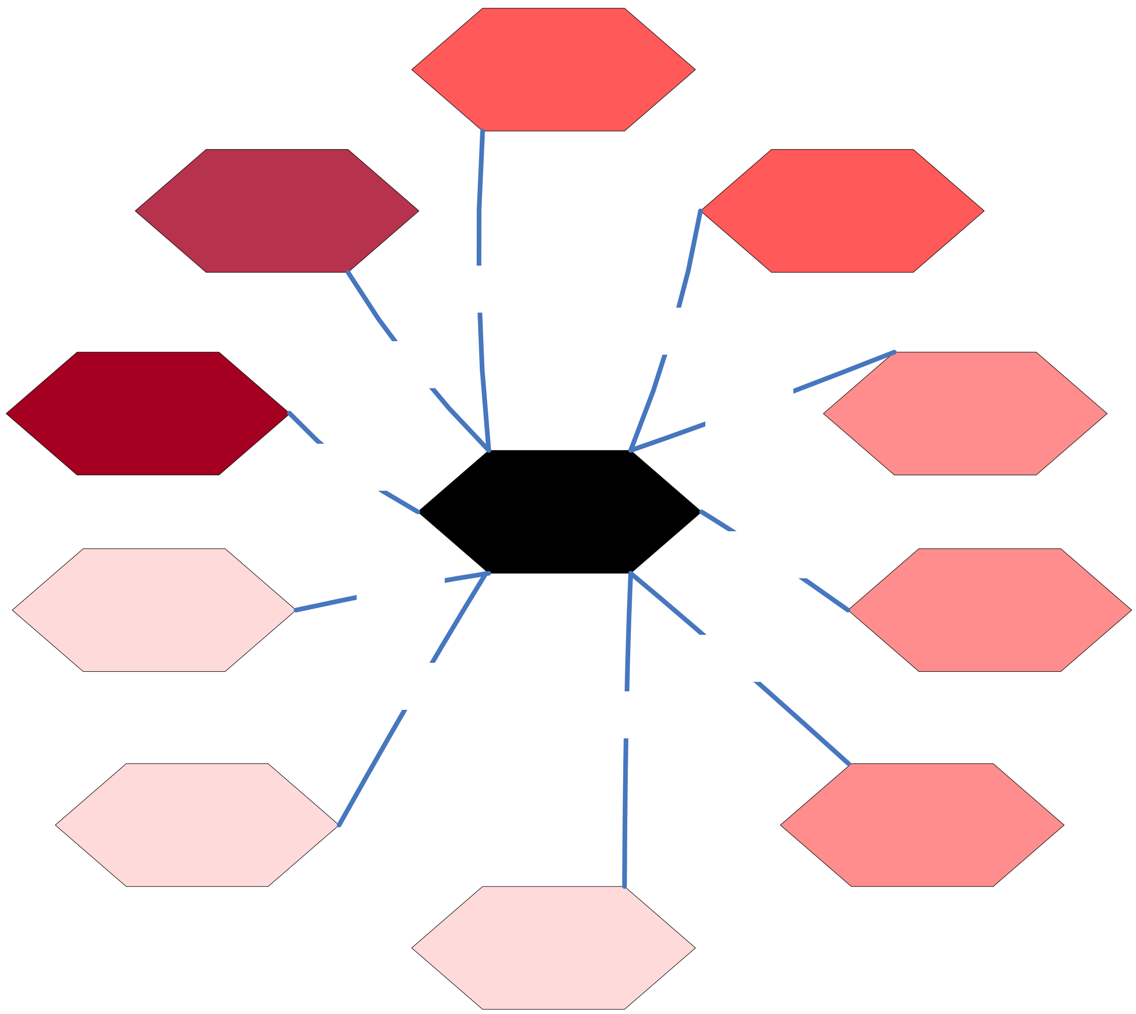
RWR: Example



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

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

RWR: 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]

Questions?