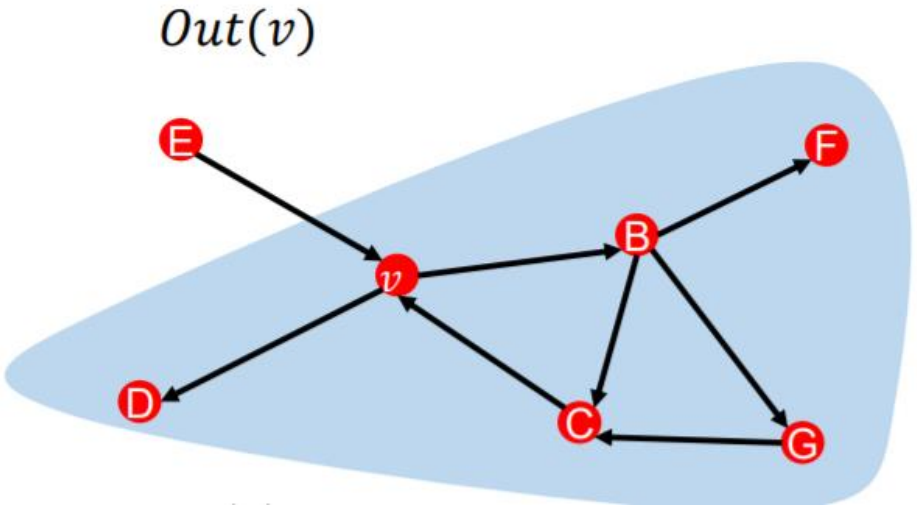
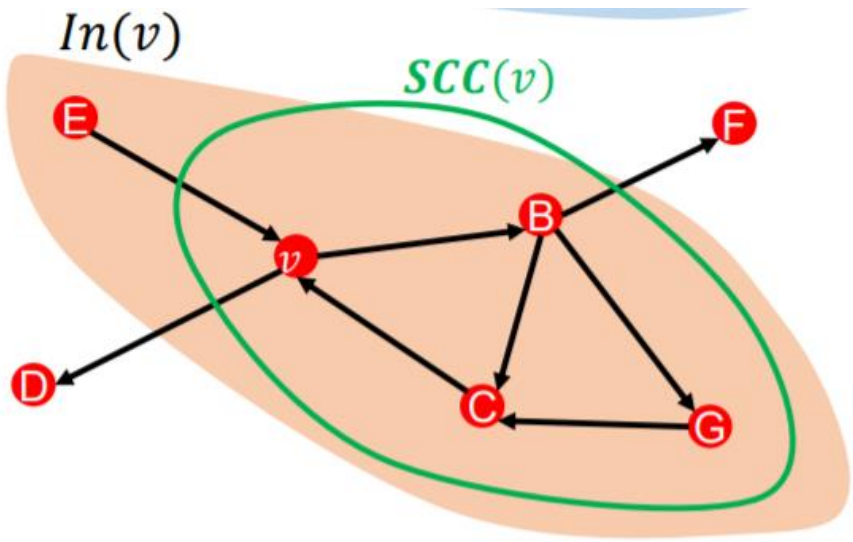
Summary Questions of the lecture

- Define the SCC in a graph and present a computation issue to find a SCC containing a node v.

→ An SCC in a directed graph is the largest possible set of nodes where every pair of nodes in the set can reach each other. The SCC containing a specific node v can be found by intersecting the set of nodes that can reach v and the set of nodes that v can reach.

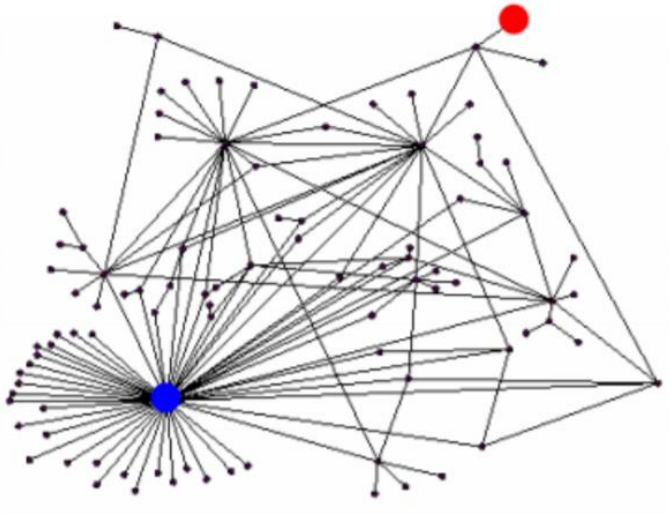




Summary Questions of the lecture

- Discuss the conceptual picture of the Web graph obtained from the result in page 13 of lecture note 14.

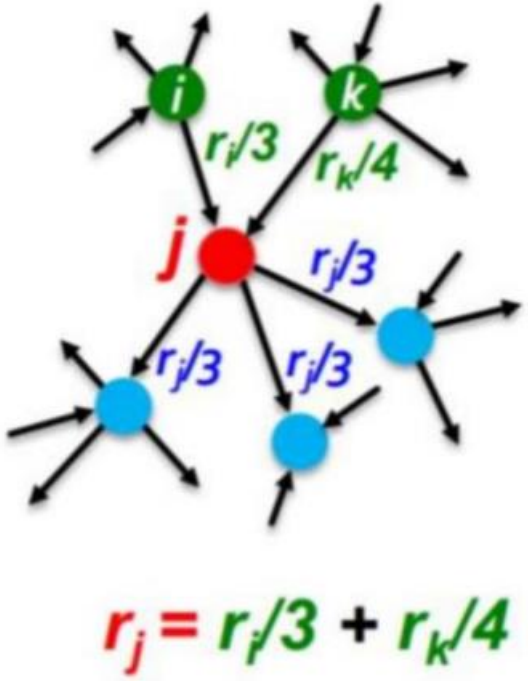
→ In the observation, a randomly chosen node either visits a lot of pages or very few nodes. Then, in average, around half of all pages can reach a randomly chosen page v and can be reached from the page v. Thus, we can say that all web pages are not equally “important”.



Summary Questions of the lecture

- Explain the “Flow’ model for PageRank and discuss its validity on why it is worth.

→“Flow’ model is formulated to estimate the importance of a page by voting the incoming links, where links incoming from important pages count more. The importance of each page are equally divided by the number of its outgoing neighbors and propagated to the outgoing neighbors. Then the importance of each page is defined by the sum of all incoming importance from its incoming neighbors. The flow model effectively captures the intuition that links from important pages are worth more than links from those that are not.



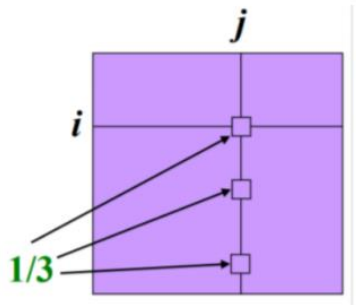
Summary Questions of the lecture

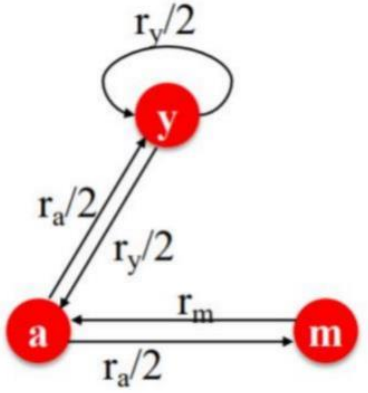
- Present a random work interpretation of ‘power iteration’ method of eigenvector formulation for PageRank.

→ The ‘power iteration’ is a method to approximately calculate the rank vector $\boldsymbol{r}$ by repeatedly multiplying the stochastic adjacency matrix M to the previous rank vector $\boldsymbol{r}$. Since each iterarion is equivalent to a web surfer randomly taking an outgoing link in the current page at each time step, the ‘power iteration’ can be interpreted as a random walk having a transition probability of M and $\boldsymbol{r}$ is interpreted as a vector of the probability that the surfer stays at each page.

- Initialize: $\boldsymbol{r}^{(0)} = \left[\frac{1}{N}, \ldots, \frac{1}{N}\right]^T$.
- Iterate: $\boldsymbol{r}^{(t+1)} = M\boldsymbol{r}^{(t)}$
- Stop when $\left\|\boldsymbol{r}^{(t+1)} - \boldsymbol{r}^{(t)}\right\|_1 < \epsilon$

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





Link Analysis

How do we actually compute the PageRank

Outline of Lecture (4)

- Spatial GCN
  - Spatial View of Simplified ChebNet
  - GraphSage (Hamilton et al. NIPS 2017)
  - GAT : Graph Attention (Veličković et al. ICLR 2018)
  - MPNN: Message Passing (Glimer et al. ICML 2017)
  - **gPool:** Graph U-Nets **(**Gao et al. ICML 2019)
  - **DiffPool:** Differentiable Pooling **(**Ying et al. NeurIPS 2018)
  - **EigenPooling:** EigenPooling (Ma et al. KDD 2019)

- Link Analysis
  - Directed Graph
    - Strongly Connected Graph
    - Directed Acyclic Graph
  - Link Analysis Algorithms
    - PageRank (Ranking of Nodes)
    - Random Teleports
    - Google Matrix
    - Sparse Matrix Formulation
    - Personalized PageRank
    - Random Walk with Restart
- Propagation using graph diffusion
  - Predict Then Propagate [ICLR'19]
  - Graph Diffusion-Embedding Networks [CVPR'19]
- Making a new graph
  - Diffusion Improves Graph Learning [NIPS'19]
  - Graph Learning-Convolutional Nets. [CVPR'19]

PageRank: How to solve?

R: Given a web graph with N nodes, where the nodes are pages and edges are hyperlinks

- Initialize: $\boldsymbol{r}^{(0)} = \left[\frac{\mathbf{1}}{\boldsymbol{N}}, \ldots, \frac{\mathbf{1}}{\boldsymbol{N}}\right]^{\boldsymbol{T}}$.
- Iterate: $\boldsymbol{r}^{(t+1)} = \boldsymbol{M}\boldsymbol{r}^{(t)}$ $\leftarrow$ $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
- Stop when $\left\|\boldsymbol{r}^{(t+1)} - \boldsymbol{r}^{(t)}\right\|_1 < \epsilon$

  where $\|\boldsymbol{x}\|_1 = \sum_{1 \le i \le N} |x_i|$ and we can use any other vector norms.

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

PageRank: Problems

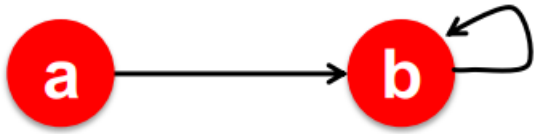
Two problems:

(1) Some pages are dead ends
- Such pages cause importance to “leak out”

(2) Spider traps (all out-links are within the group)
- Eventually spider traps absorb all importance

PageRank: Does this converge?

The “Spider trap” problem:



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

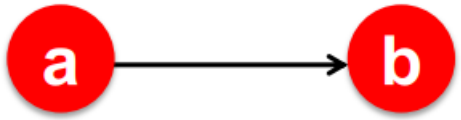
Example:

- Iteration → 0 | 1 | 2 | 3 ……

	0	1	2	3
r_a	1	0	0	0
r_b	0	1	1	1

PageRank: Does it converge to what we want?

The “Dead end” problem:



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

Example:

- Iteration → 0 | 1 | 2 | 3 ……

	0	1	2	3
r_a	1	0	0	0
r_b	0	1	0	0

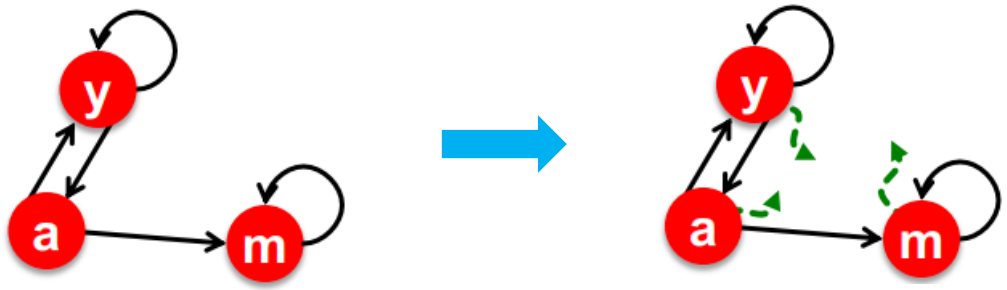
PageRank: Solution to Spider Traps

Google solution for spider traps:

- At each time step, the random surfer has two options
  - With probability β, follow a link at random
  - With probability $1 - \beta$, jump to a random page
  - Common values for β are in the range 0.8 to 0.9

Result:

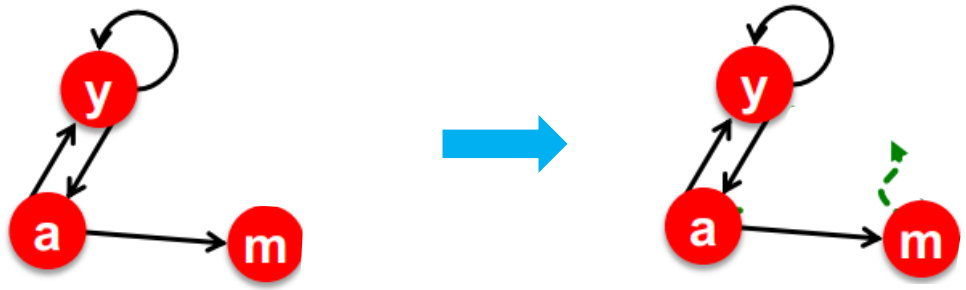
- Surfer will teleport out of spider trap within a few time steps



PageRank: Solution to Dead Ends

Solution for dead ends :

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



	y	a	m
y	$1/2$	$1/2$	0
a	$1/2$	0	0
m	0	$1/2$	0

	y	a	m
y	$1/2$	$1/2$	$1/3$
a	$1/2$	0	$1/3$
m	0	$1/2$	$1/3$

PageRank: Why teleports solve the problem?

Why are dead-ends and spider traps problems 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

PageRank: Random Teleports

Google solution does it all:

- At each time step, the random surfer has two options
  - With probability β, follow a link at random
  - With probability $1-\beta$, jump to a random page

PageRank equation [Brin-Page, 98]

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

Note: 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.

PageRank: Google Matrix

PageRank equation [Brin-Page, 98]

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

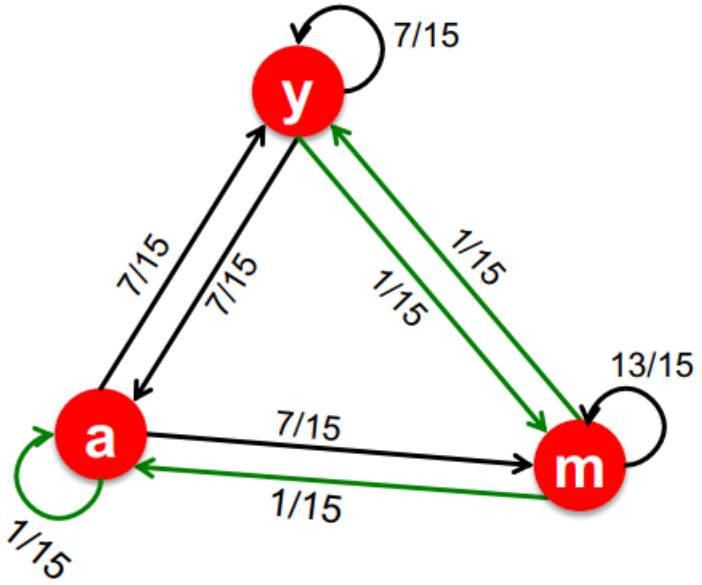
- Google Matrix $\boldsymbol{A}$:

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

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

- We have a recursive problem: $\boldsymbol{r} = \boldsymbol{A}\boldsymbol{r}$, where the power method still works!
- What is β?
  - In practice β =0.8, 0.9 (make 5 steps on average jump)

PageRank: Radom Teleports by Google Matrix ($\beta = 0.8$)



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

M $\qquad$ $[1/N]_{N\times N}$

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

$$\begin{array}{cccccc} y & 1/3 & 0.33 & 0.25 & 0.26 & \cdots & 7/33 \\ a & = 1/3 & 0.20 & 0.22 & 0.18 & \cdots & 5/33 \\ m & 1/3 & 0.47 & 0.53 & 0.56 & & 21/33 \end{array}$$

PageRank: Computing PageRank

Key step is matrix-vector multiplication

$$r^{new} = Ar^{old}$$

$$\boldsymbol{A} = \beta \boldsymbol{M} + (1-\beta)\left[\frac{1}{N}\right]_{N\times N} \leftarrow r_j = \beta \sum_{i\to j} \frac{r_i}{d_i} + \sum_i (1-\beta)\frac{r_i}{N}$$

Easy if we have enough main memory to hold $\boldsymbol{A}$, $\boldsymbol{r^{new}}, \boldsymbol{r^{old}}$

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!

$\boldsymbol{M}$

$1/2$	$1/2$	0
$1/2$	0	0
0	$1/2$	1

0.8 + 0.2

$[1/N]_{N\times N}$

$1/3$	$1/3$	$1/3$
$1/3$	$1/3$	$1/3$
$1/3$	$1/3$	$1/3$

$\boldsymbol{A}$

$7/15$	$7/15$	$1/15$
$7/15$	$1/15$	$1/15$
$1/15$	$7/15$	$13/15$

PageRank: Sparse Matrix Formulation

- We can rearrange the PageRank equation

$$\boldsymbol{r} = \beta \boldsymbol{M}\boldsymbol{r} + [(1-\beta)/N]_N \underbrace{\sum_i r_i}_{=1} \leftarrow \boldsymbol{r} = \boldsymbol{A}\boldsymbol{r}, \boldsymbol{A} = \beta \boldsymbol{M} + (1-\beta)[1/N]_{N\times N}$$

where $[(1-\beta)/N]_N$ is a vector with all N entries $(1-\beta)/N$.

- $\boldsymbol{M}$ is a sparse matrix! (with no dead-ends)
  - 10 links per node, approx. 10 $N \ll N^2$ entries
- So in each iteration, we need to:
  - Compute $\boldsymbol{r}^{\boldsymbol{new}} = \beta \boldsymbol{M}\boldsymbol{r}^{\boldsymbol{old}}$
  - Add a constant value$(1-\beta)/N$ to each entry in $\boldsymbol{r}^{\boldsymbol{new}}$
  - Note if $\boldsymbol{M}$ contains dead-ends then $\sum_i r_i^{new} < 1$ and we also have to renormalize $\boldsymbol{r}^{\boldsymbol{new}}$ so that it sums to 1 (usually, we add self-link to dead end nodes)

PageRank: The Complete Algorithm

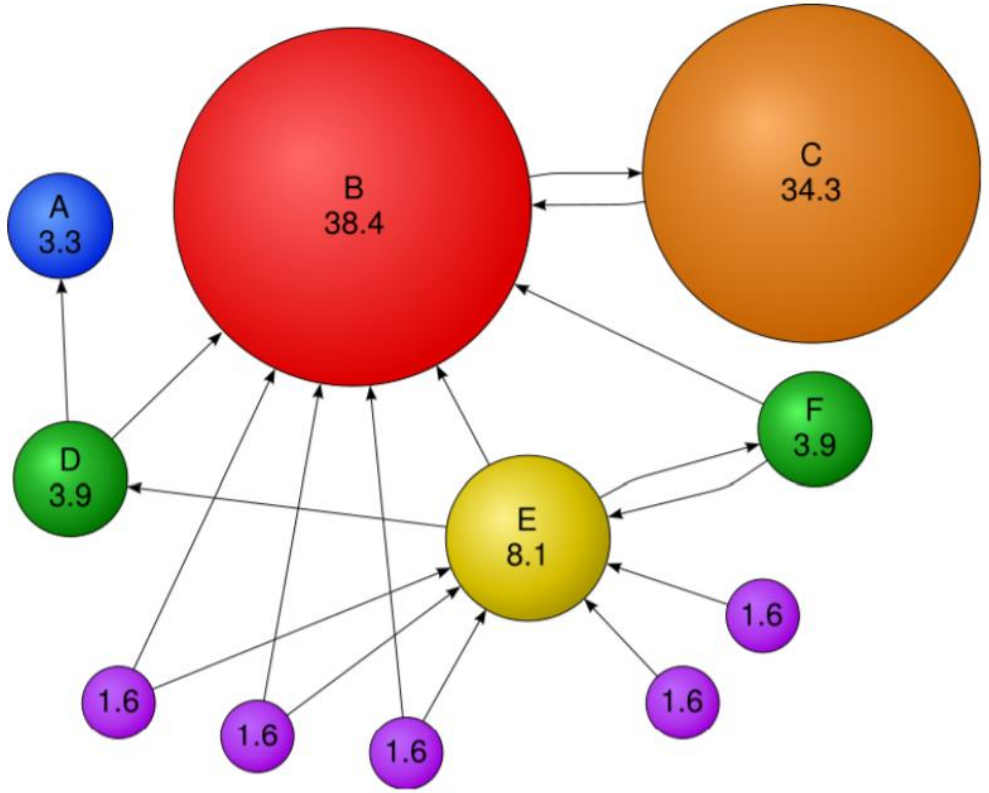
PageRank Algorithm

01: **Input**
02: G: Directed graph with adding self-link to dead end nodes
03: β: Parameter
04: **Output**
05: $\boldsymbol{r}^{new}$: PageRank vector
06: **Initialization**
07: $r_j^{old} = 1/N$
08: **while** $\sum_j \left| r_j^{new} - r_j^{old} \right| \geq \epsilon$ **do**
09: $\forall j,\ r_j^{new} = \beta \sum_{i \to j} \frac{r_i^{old}}{d_i} + \frac{(1-\beta)}{N}$
10: $\boldsymbol{r}^{old} = \boldsymbol{r}^{new}$
11: **end while**

PageRank: The Complete Algorithm

PageRank Algorithm

01: **Input**
02: G: Directed graph with adding self-link to dead end nodes
03: β: Parameter
04: **Output**
05: $\boldsymbol{r}^{new}$: PageRank vector
06: **Initialization**
07: $r_j^{old} = 1/N$
08: **while** $\sum_j \left| r_j^{new} - r_j^{old} \right| \geq \epsilon$ **do**
09: $\forall j,\ r_j^{new} = \beta \sum_{i \to j} \frac{r_i^{old}}{d_i} + \frac{(1-\beta)}{N}$
10: $\boldsymbol{r}^{old} = \boldsymbol{r}^{new}$
11: **end while**



Link Analysis

Random Walk with Restarts and
Personalized PageRank

Personalized PageRank: Example

Graph Search

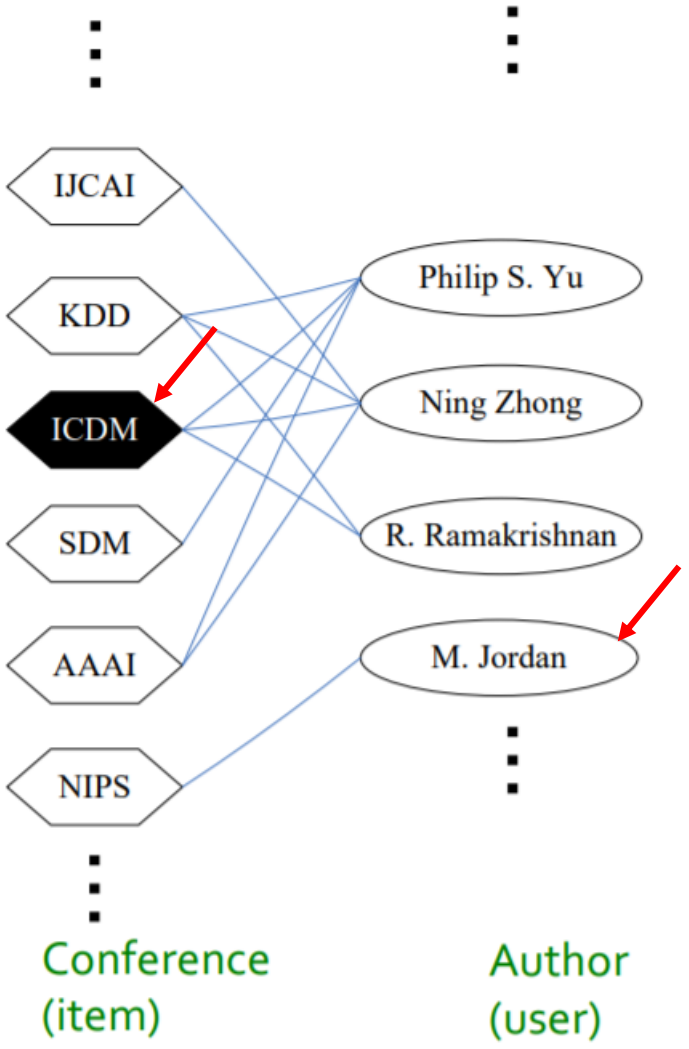
- Given:

  Conferences-to-authors graph

- Goal:

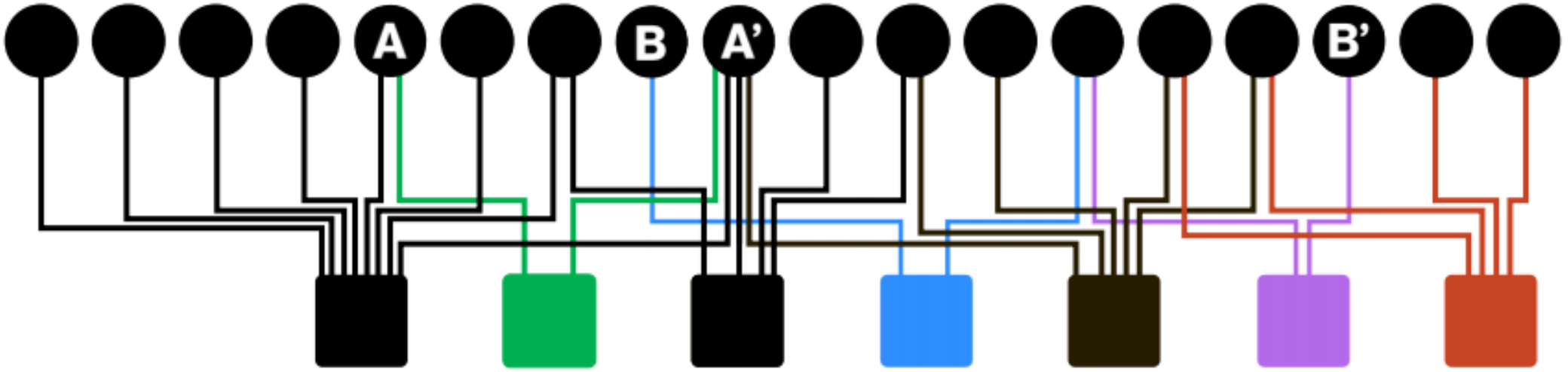
  Proximity on graphs

  - What is most related conference to ICDM?
  - What conferences should we recommend to M. Jordan to attend?



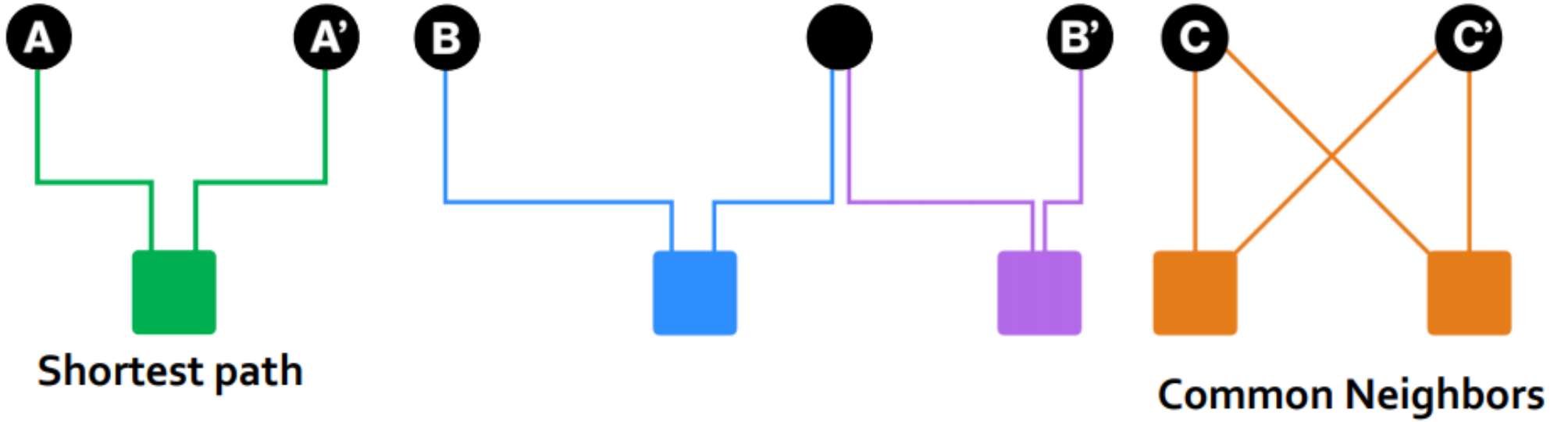
Personalized PageRank: Bipartite User-to-Item Graph

Which is more related A,A' or B,B'?



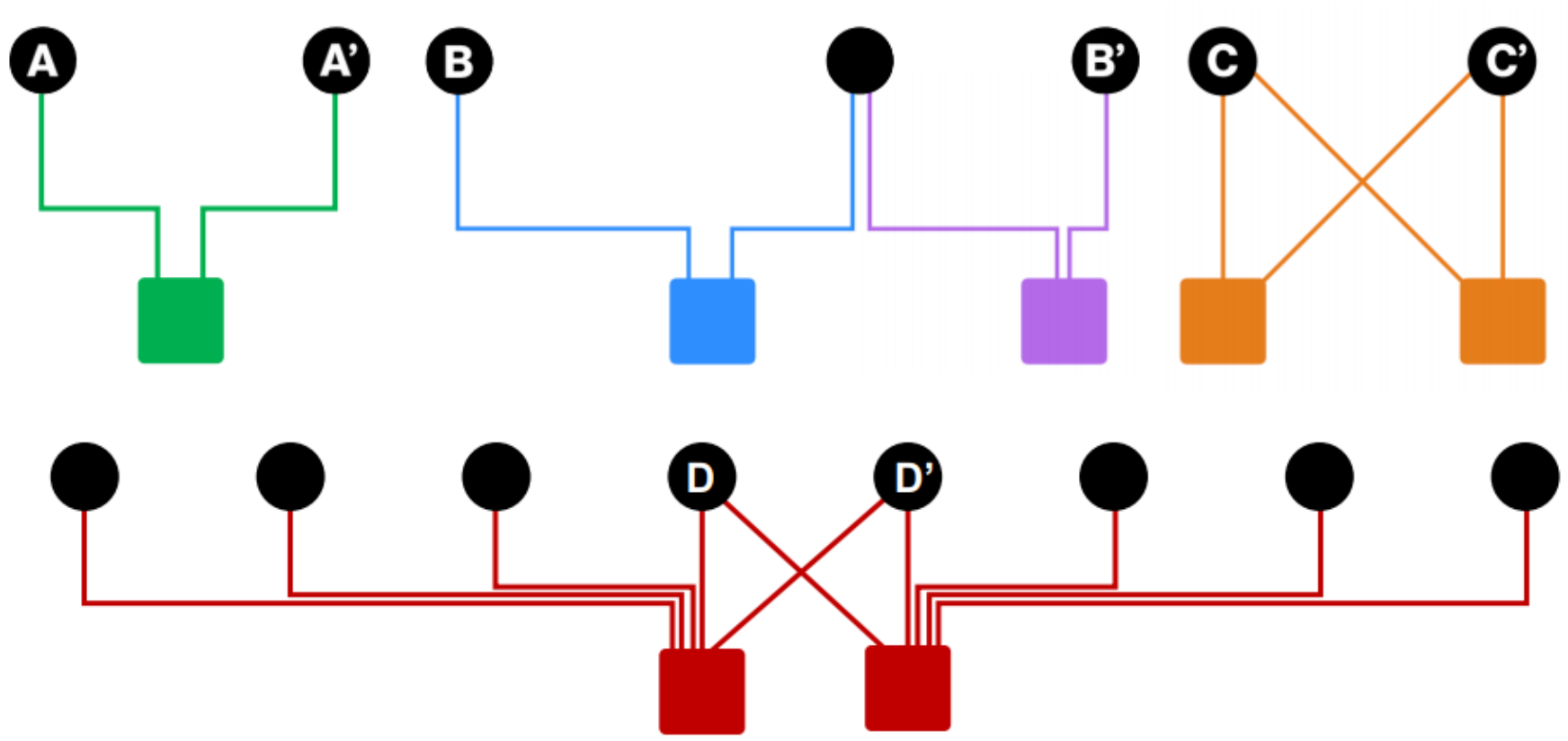
Personalized PageRank: Bipartite User-to-Item Graph

Which is more related A,A', B,B' or C,C'?



Personalized PageRank: Bipartite User-to-Item Graph

Which is more related A,A', B,B', C,C', or D, D'?



Personalized Page Rank/Random Walk with Restarts

Personalized PageRank, Proximity on Graph

Graphs and web search:

- Ranks nodes by "importance"

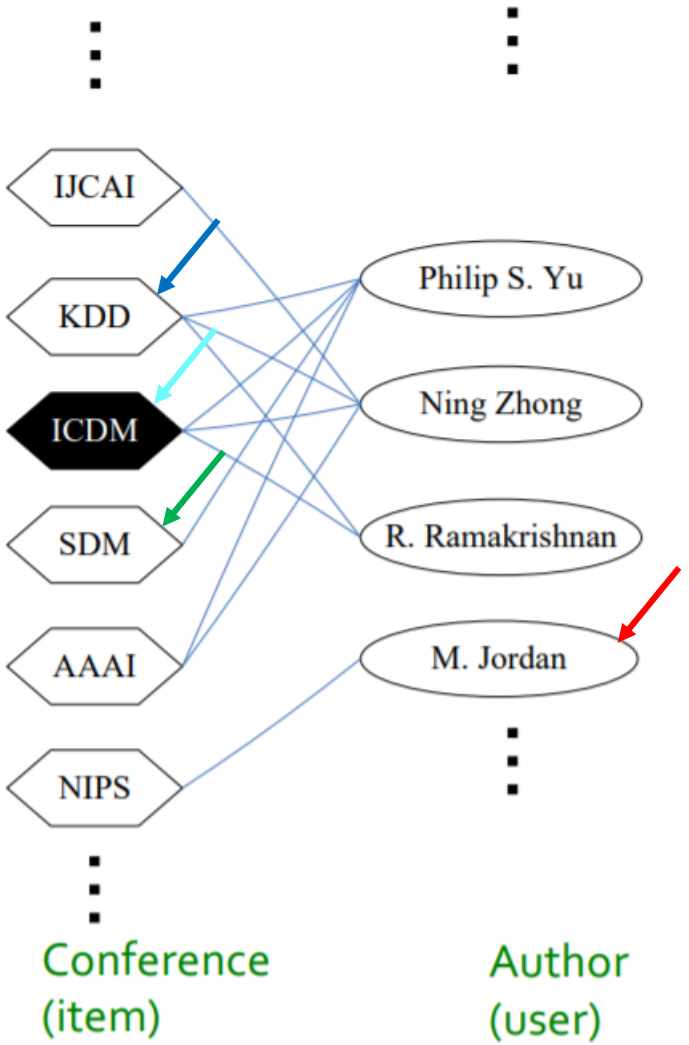
Personalized PageRank:

- Ranks proximity of nodes to the query(teleport) nodes $\mathcal{T} = \{..., N_i, ...\}$

Proximity on graphs:

- Q: What is most related conference to ICDM?
- Random Walks with Restarts:

→Teleport back to the starting query node:
$\mathcal{T}$ = { single node }



Personalized PageRank: Teleports

Random Walks

- Every node has some importance
- Importance gets evenly split among all edges and pushed to the neighbors, based on a graph of relationships between nodes:

Random Walks with Personalized Teleports:

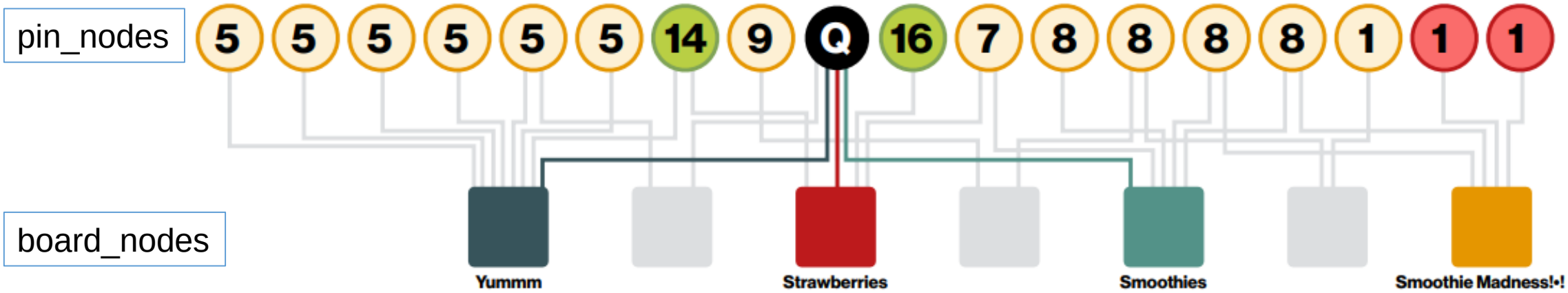
- Given a set of query nodes, we simulate a random walk:
- Make a step to a random neighbor and record the visit (visit count, voting)
- With probability α, restart the walk at one of the query nodes
- The nodes with the highest visit count have highest proximity to the query nodes

Personalized PageRank: proximity to nodes

Query nodes $= \{\dots, Q, \dots\}$

$\alpha = 0.5$

```
pin_nodes = query_nodes.sample_by_weight( )
For i = 1: M
    board_node = pin_node.get_random_neighbor( )
    pin_nodes = board_nodes.get_tandom_neighbor( )
    pin_nodes.visit_count +=1
    if random( ) < α,
        pin_nodes = query_nodes.sample_by_weight( )
    end
end
```



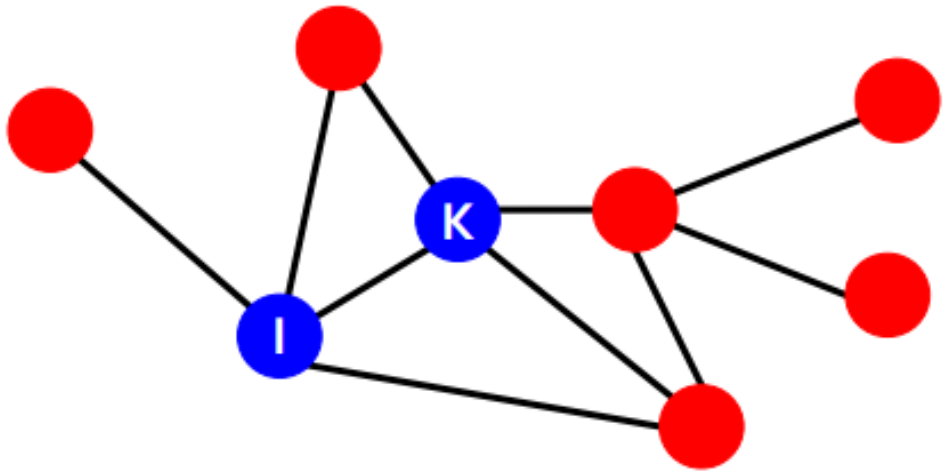
Personalized PageRank: benefits

- Why is this great?
- It considers:
  - Multiple connections (hops)
  - Multiple paths
  - Direct and indirect connections
  - Degree of the node

Personalized PageRank: Non-bipartite Graphs

Q: Which conferences are closest to KDD & ICDM?

A: Personalized PageRank with teleport set S={KDD, ICDM}

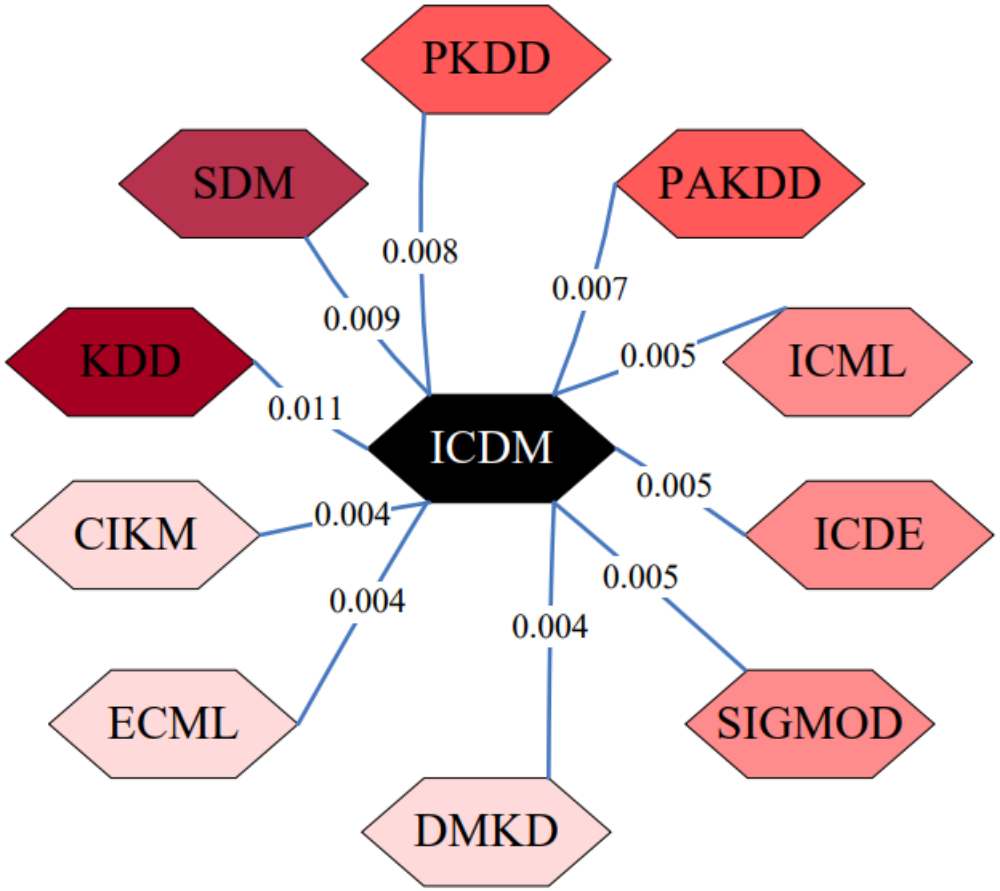


Graph of CS conferences

Personalized PageRank: Non-bipartite Graphs

Q: Which conferences is the most related to ICDM?

A: Random walks with restart at S={ICDM}



PageRank: Summary

- "Normal" PageRank:
  - Teleports uniformly at random to any node
  - All nodes have the same probability of surfer landing there
  - $\mathcal{T}^T$ = [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
  - $\mathcal{T}^T$= [0.1, 0, 0, 0.2, 0, 0, 0.5, 0, 0, 0.2]
- PageRank with "Restarts":
  - Topic-Specific PageRank where teleport is always to the same node
  - $\mathcal{T}^T$=[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

$$\boldsymbol{r} = (1-\alpha)\boldsymbol{M}\boldsymbol{r} + \alpha\mathcal{T}$$

Summary Questions of the lecture

- Why are dead-ends and spider traps problems and why do teleports solve these problems?
- Describe the PageRank algorithm with teleports based on sparse matrix formulation.
- Describe the random walks with personalized teleports.