Fast Random Walk with Restart: Algorithms and Applications

U Kang
Dept. of CSE
Seoul National University

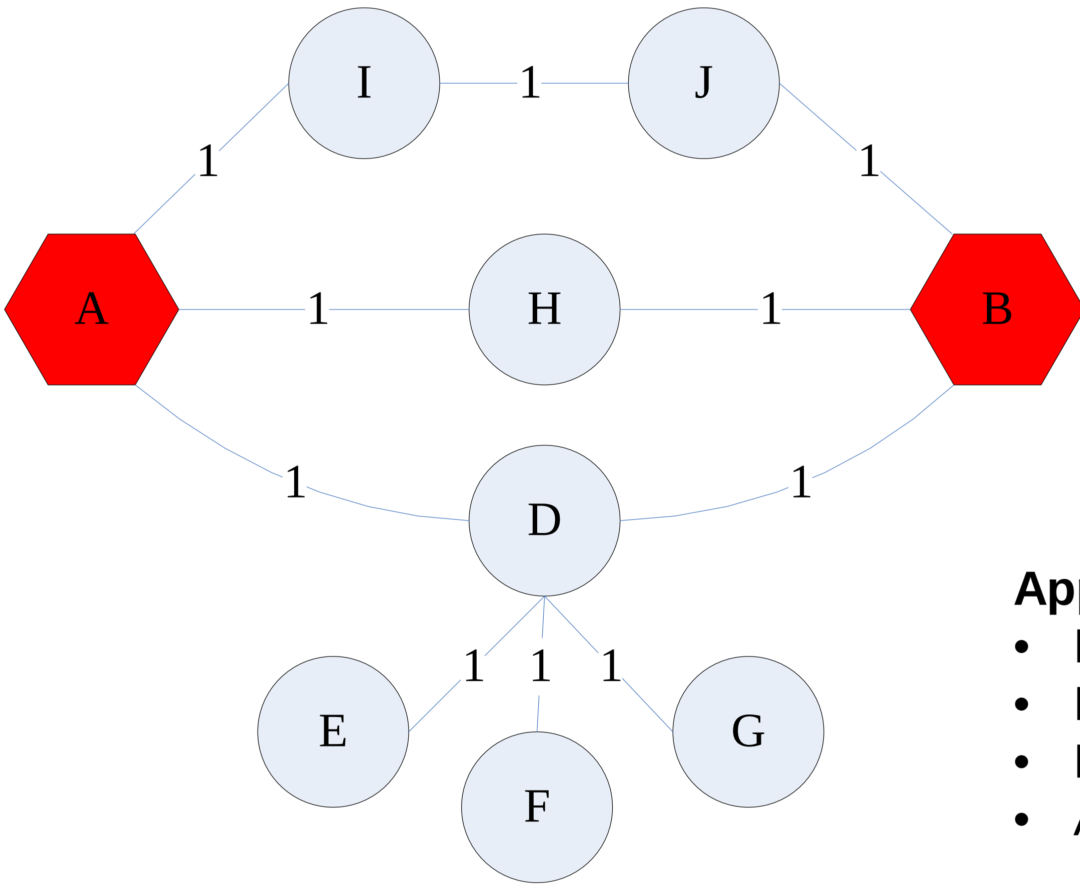
Today's Talk

- RWR for ranking in graphs: important problem with many real world applications
  - Web search, friend recommendation, product recommendation, …
- BePI: state-of-the-art method for *exact* RWR
  - Linear algebra + Graph theory + Real World Graph Analysis
- TPA and OSP: those for *approximate* RWR

Outline

- **Random Walk with Restart (RWR)**
- Fast Exact RWR
- Fast Approximate RWR
- Conclusions

Proximity on Graphs



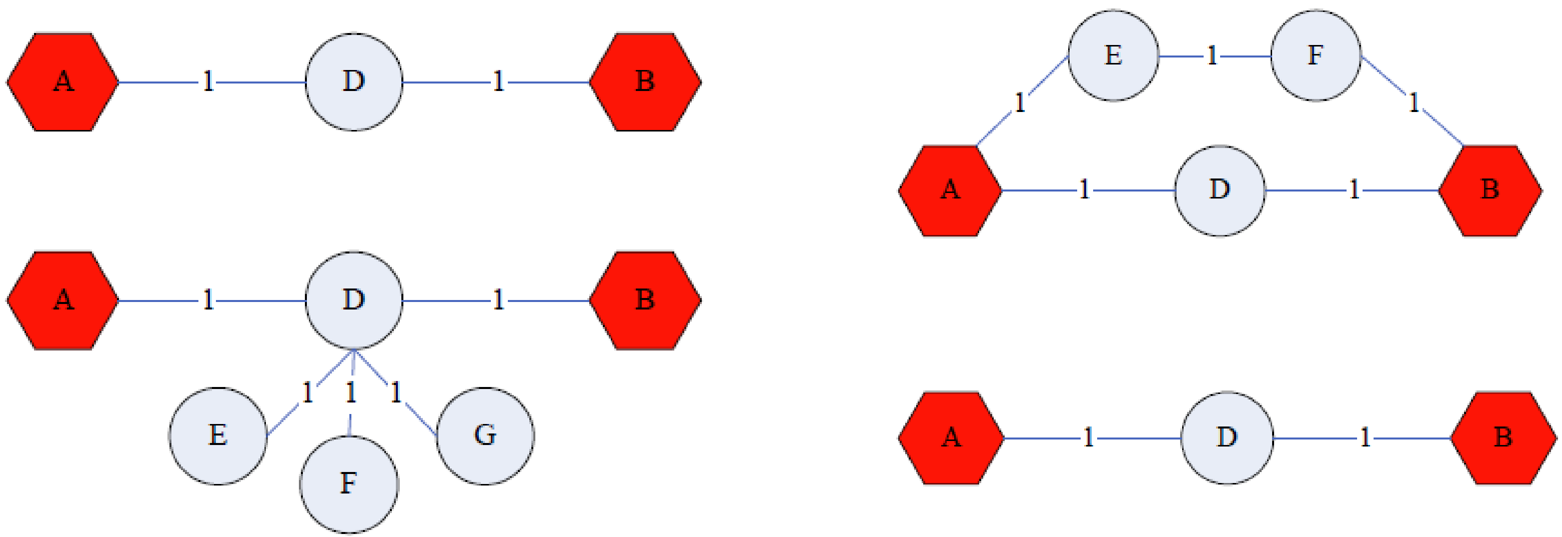
Application

- Recommendation
- Ranking
- Link Prediction
- Anomaly Detection

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

Good proximity measure?

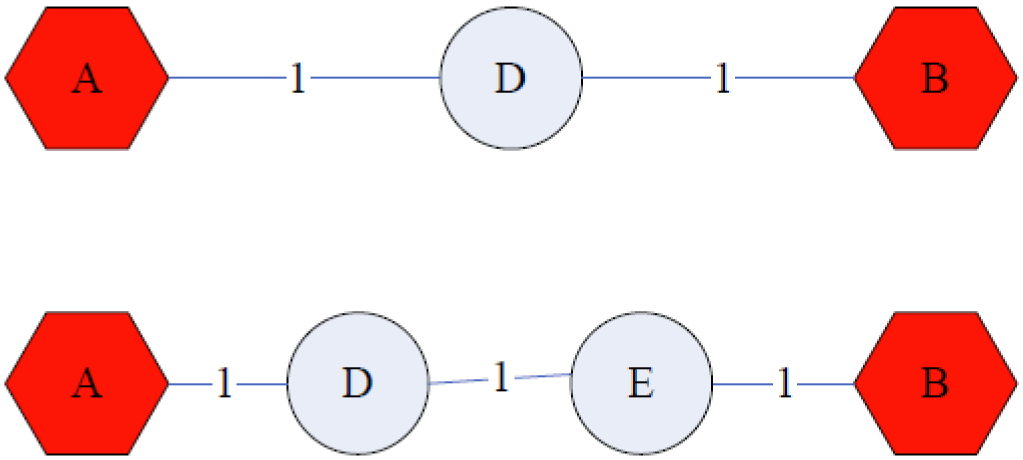
- **Shortest path is not good:**



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

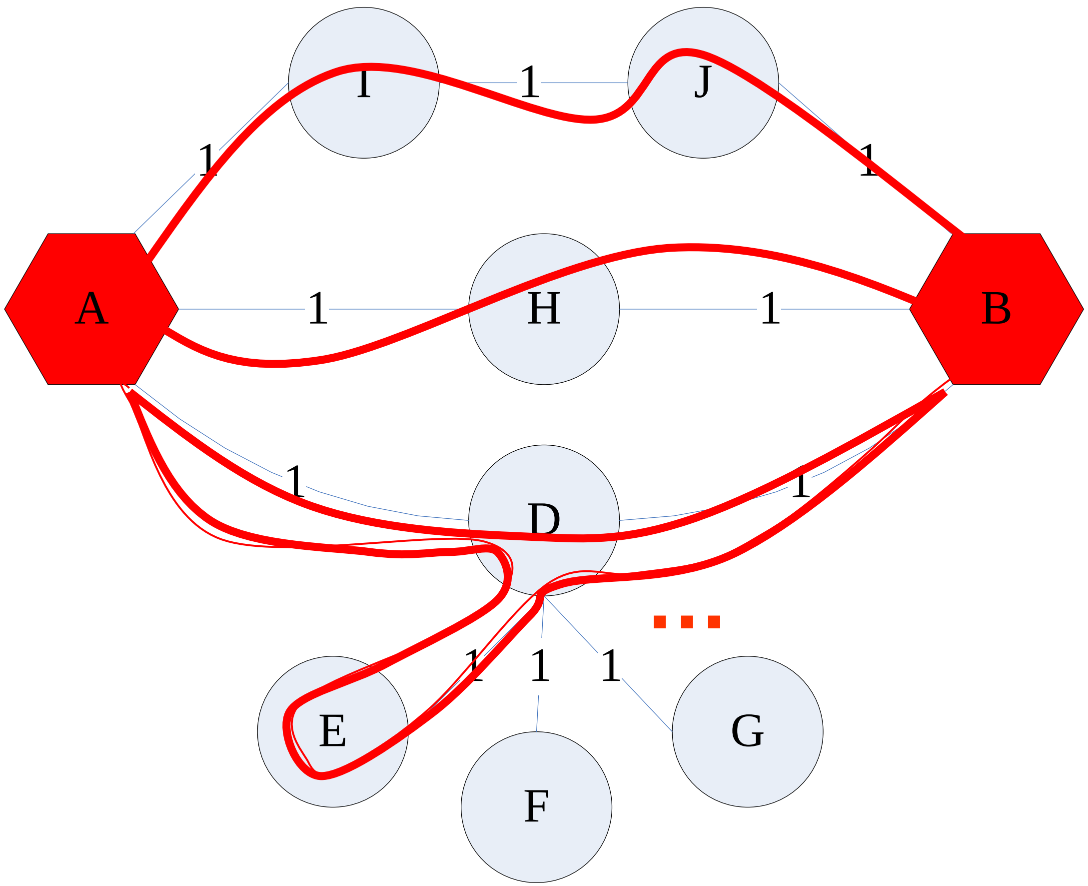
Good proximity measure?

- **Network flow is not good:**



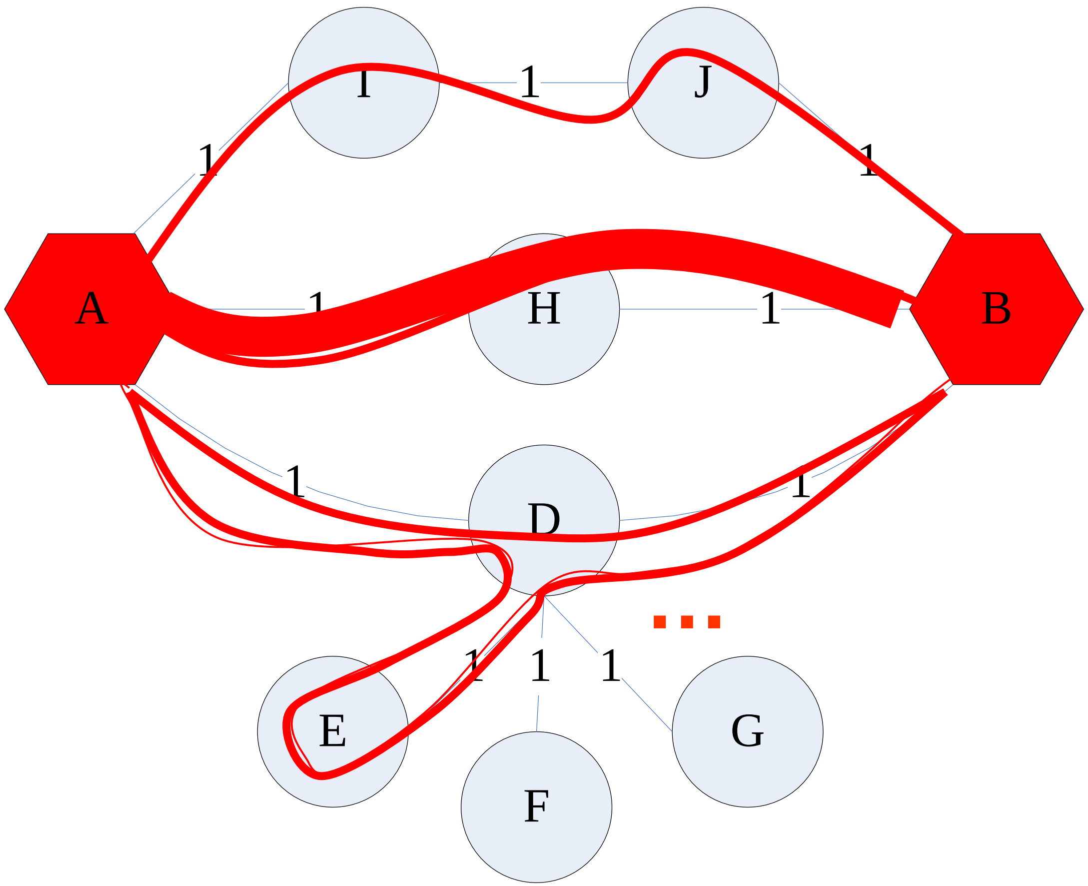
- **Does not punish long paths**

What is good notion of proximity?



- **Multiple connections**

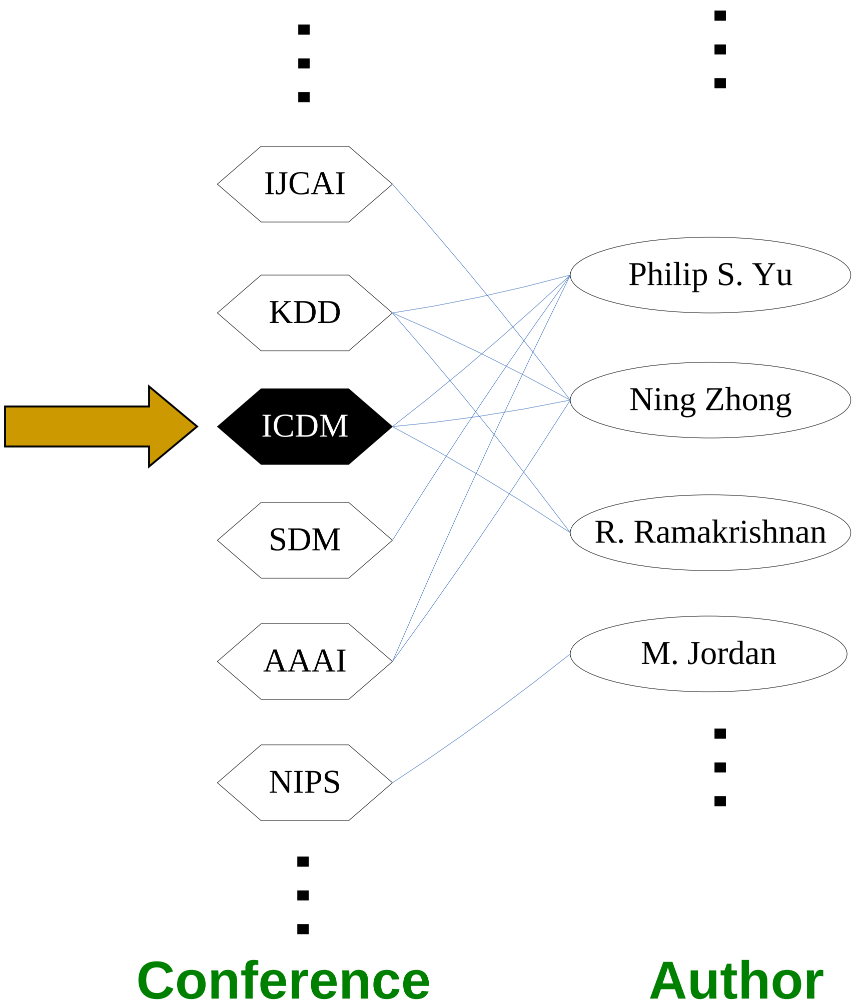
What is good notion of proximity?



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

•Answer: RWR !

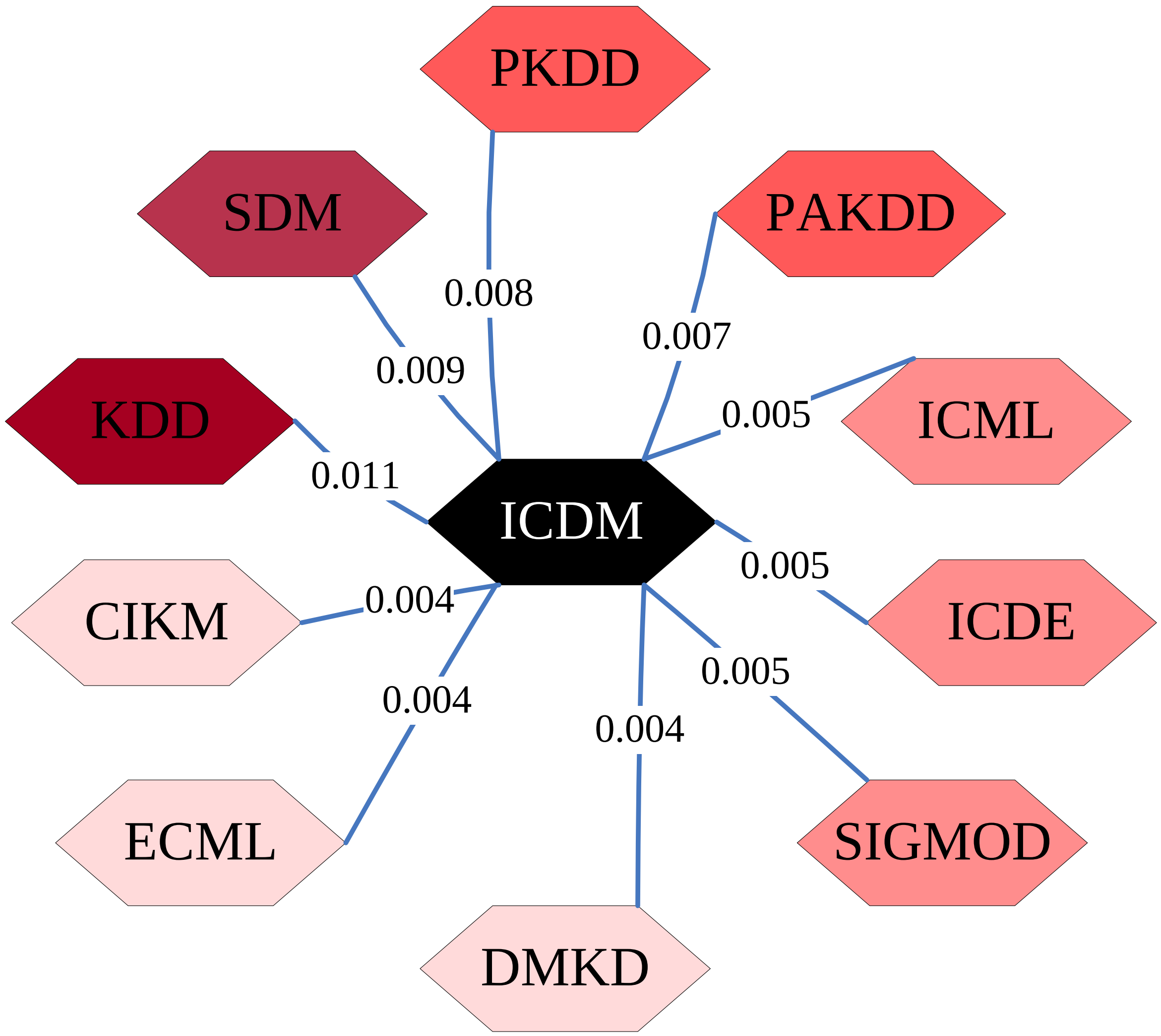
RWR: Example



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

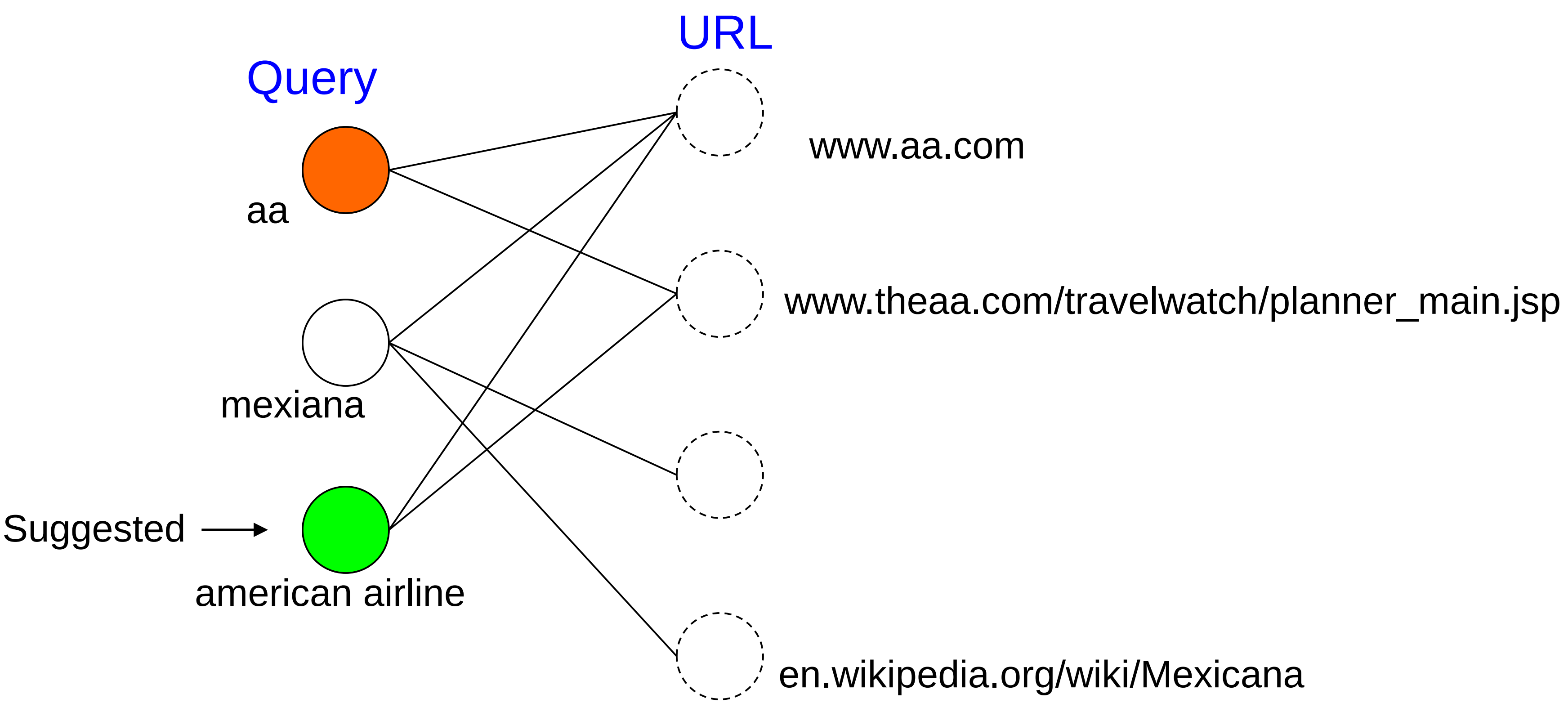
A: Random Walk With Restart from S={ICDM}

RWR: Example


PKDD
SDM
PAKDD
KDD
ICML
ICDM
CIKM
ICDE
ECML
SIGMOD
DMKD
0.008
0.009
0.007
0.005
0.011
0.005
0.004
0.004
0.005
0.004

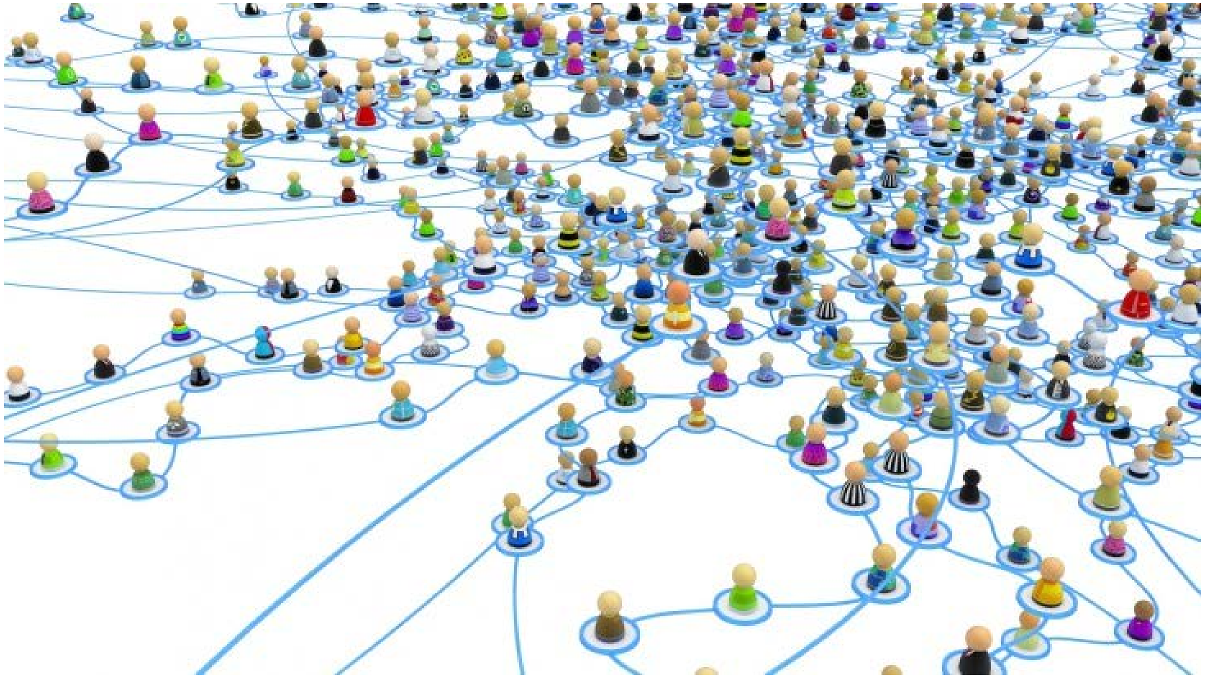
RWR: Applications

- Web Search: Query Suggestion



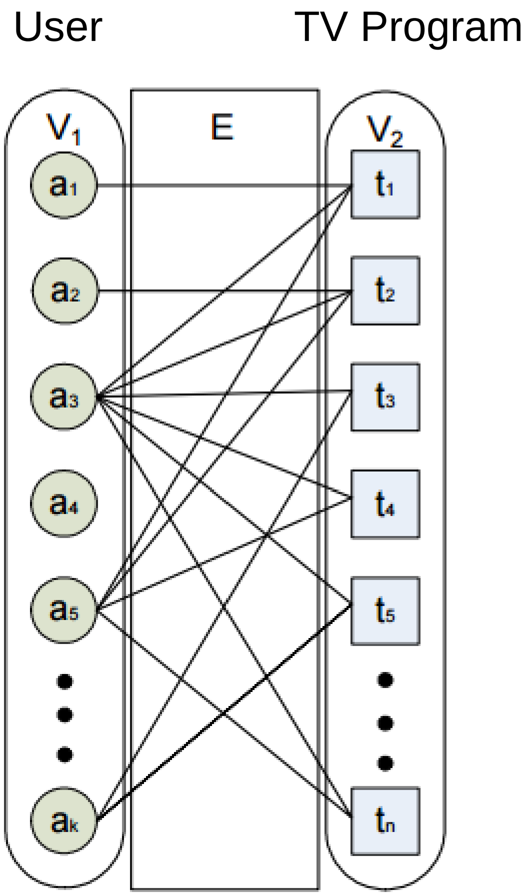
RWR: Applications

- Friend Recommendation



RWR: Applications

- TV Program Recommendation

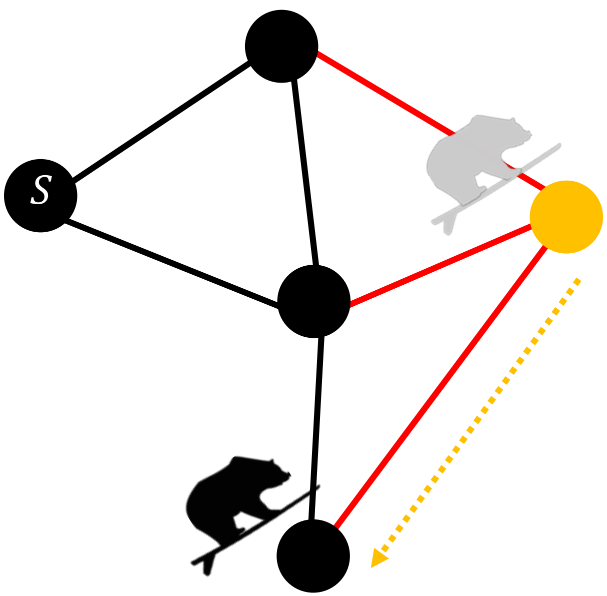


Random Walk with Restart (1)

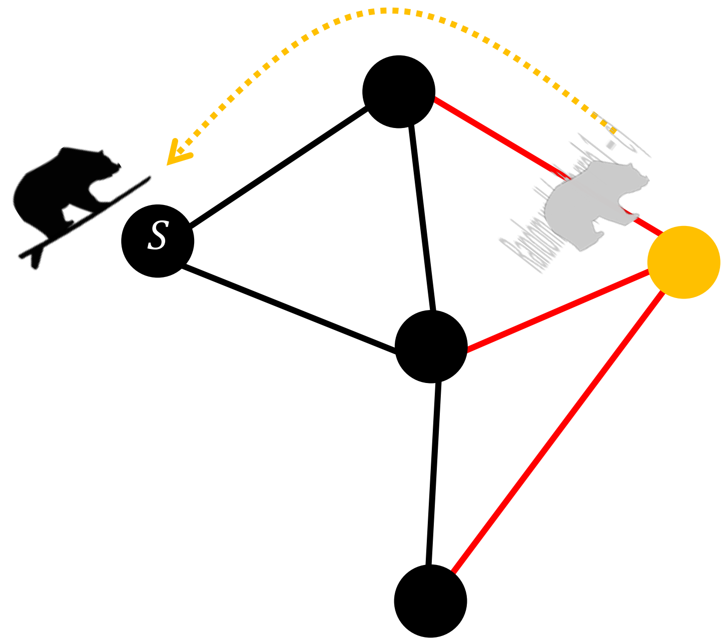
- Given a query node, compute proximities of other nodes to the query node
- A random surfer moves to one of its outgoing neighbor with prob. 1-c, and jumps to the query node with prob. c
  - After many moves, RWR score of a node is proportional to # of times the node is visited
- Also called Personalized PageRank
  - Similar to PageRank, but the random surfer jumps only to the query nodes

Random Walk with Restart (2)

- RWR assumes a random surfer on a graph



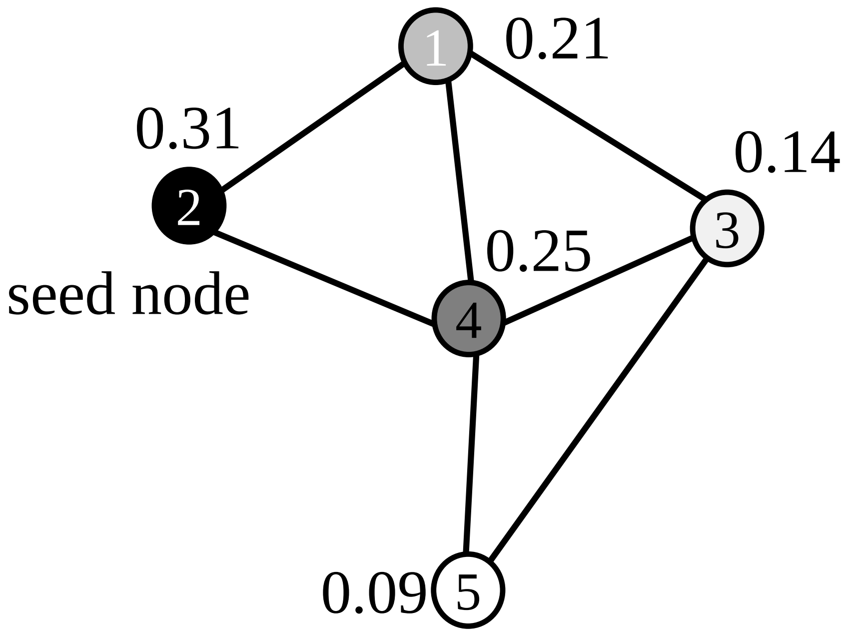
Random walk (with prob $1 - c$)



Restart (with prob c)

Random Walk with Restart (3)

- RWR computes the stationary probability that the surfer stays at each node



Node	RWR Score (relevance with node 2)
1	0.21
2	0.31
3	0.14
4	0.25
5	0.09

Restarting probability $c = 0.2$

Conclusion: RWR

- Random Walk with Restart
  - Personalized PageRank to compute node proximity

- Widely used for measuring proximities of nodes in graphs
  - Applications: Web search, friend recommendation, product recommendation, …

Outline

- ☑ Random Walk with Restart (RWR)
- ☐ **Fast Exact RWR**
- ☐ Fast Approximate RWR
- ☐ Conclusions

Overview

- I will describe two state-of-the-art exact RWR algorithms
  - BEAR (SIGMOD 2015)
  - BePI (SIGMOD 2017)

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

http://datalab.snu.ac.kr/bear

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?**

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

Previous Methods

- **Background:**
  - RWR score vector $\vec{\boldsymbol{r}}$ has to be computed with regard to many different query vectors $\vec{\boldsymbol{q}}$s
  - Computing $\vec{\boldsymbol{r}}$ from scratch (e.g., the iterative method) takes too long for large graphs
- **Approach:**
  - Preprocessing the graph to speed up the RWR computation
- **Limitations:**
  - Previous preprocessing methods require too much space and/or do not guarantee accuracy of $\vec{\boldsymbol{r}}$

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

Previous Method: Inversion (2)

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

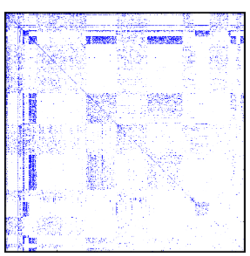
$$\vec{\boldsymbol{r}} = \boldsymbol{H}^{-1}(c\vec{\boldsymbol{q}})$$

- **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

Other Preprocessing methods (1)

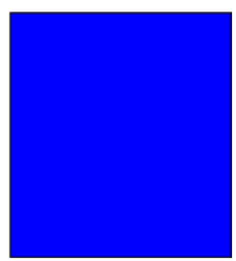
- Replace $\boldsymbol{H}^{-1}$ with sparser matrices by reordering and decomposing $\boldsymbol{H}$
- Still expensive in terms of space and/or inaccurate

Input graph
#nz=0.1M



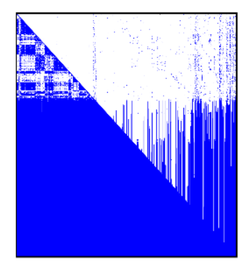
H

(1) Inversion
Exact, #nz=527M

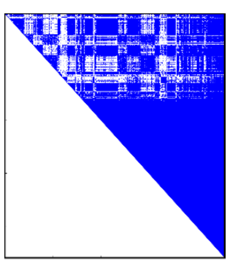


H^{-1}

(2) QR (Fujiwara et al. 12)
Exact, #nz=428M



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

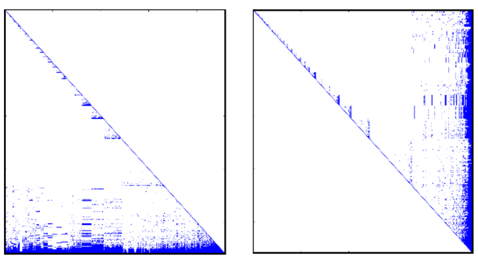


R^{-1}

Sparsity pattern of preprocessed matrices on the Routing dataset

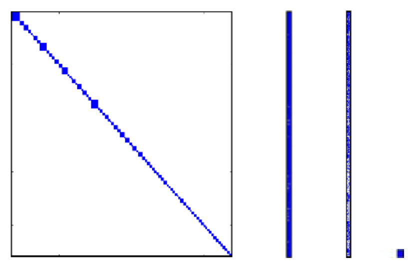
Other Preprocessing methods (2)

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



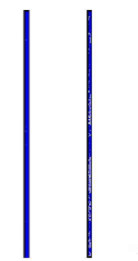
L^{-1} U^{-1}

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



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

(5) NB_LIN
Approx, #nz=3M



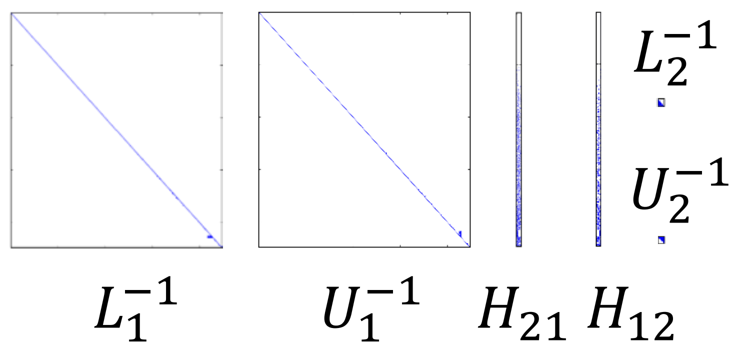
U V $\tilde{\Lambda}$

Sparsity pattern of preprocessed matrices on the Routing dataset

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)

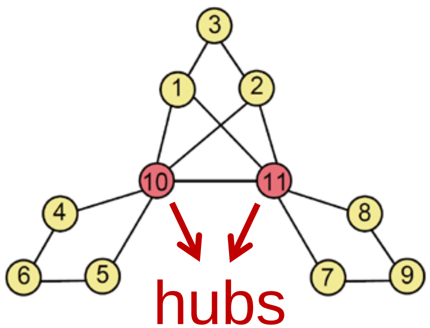
- ***BEAR*** offers two versions
  - ***BEAR-Exact***: guarantees accuracy
  - ***BEAR-Approx***: fast and space-efficient but allows small error
- ***BEAR*** consists of the two phases
  - **Preprocessing phase** (one-time cost): partitions the adjacency matrix into submatrices and precomputes several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

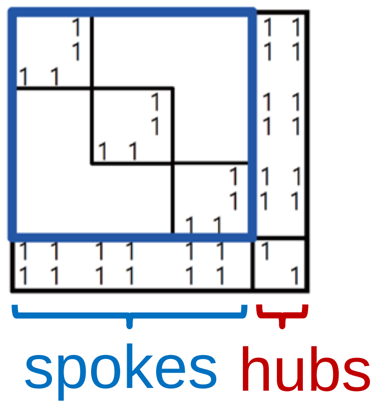
BEAR: Main Idea

- The key issue is inverting a matrix
  - $\vec{\boldsymbol{r}} = \left(\boldsymbol{I} - (1-c)\widetilde{\boldsymbol{A}}^{\boldsymbol{T}}\right)^{-\mathbf{1}} c\vec{\boldsymbol{q}} = \boldsymbol{H}^{-\mathbf{1}} c\vec{\boldsymbol{q}}$
- Use "block elimination" idea
  - If we can invert a submatrix of H easily, then we can invert H easily as well!
- But, the original adjacency matrix is not block elimination-friendly
  - Reorder the graph to easily invert a submatrix!

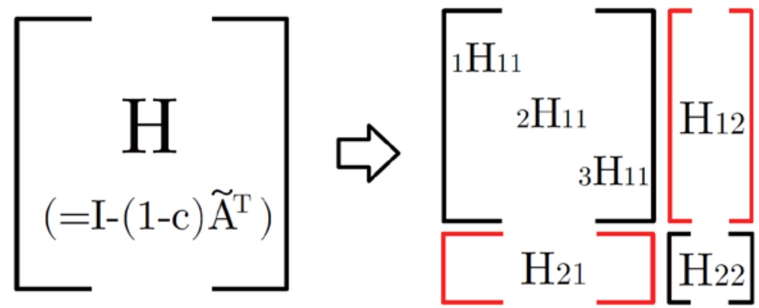
Preprocessing Phase

1. Reordering

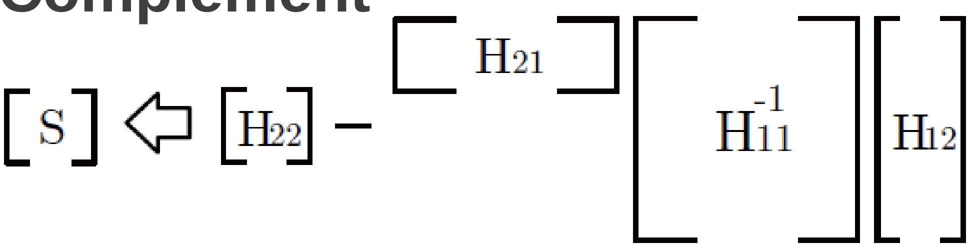




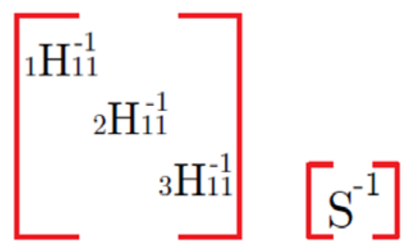
2. Partitioning



3. Schur Complement



4. Inverting



Aside: Graph Reordering

SlashBurn: Graph Compression and Mining beyond Caveman Communities (ICDM 2011, TKDE 2014)

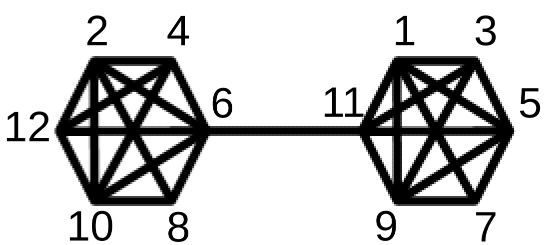
U Kang
(SNU)

Yongsub Lim
(SNU)

Christos Faloutsos
(CMU)

Node Order Matters

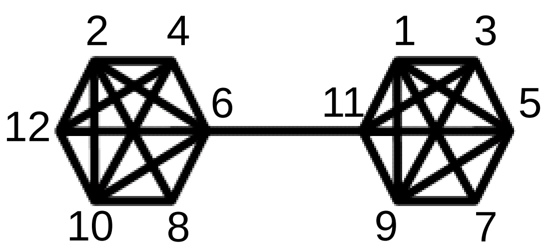
- A graph and the adjacency matrix

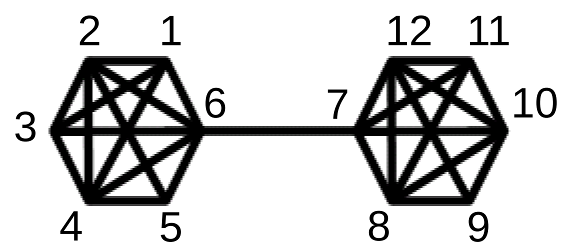


	1	2	3	4	5	6	7	8	9	10	11	12
1			1		1		1		1		1	
2				1		1		1		1		1
3	1				1		1		1		1	
4		1				1		1		1		1
5	1		1				1		1		1	
6		1		1				1		1	1	1
7	1		1		1				1		1	
8		1		1		1				1		1
9	1		1		1		1				1	
10		1		1		1		1				1
11	1		1		1	1	1		1			
12		1		1		1		1		1		

Node Order Matters

- Same graphs with different orderings



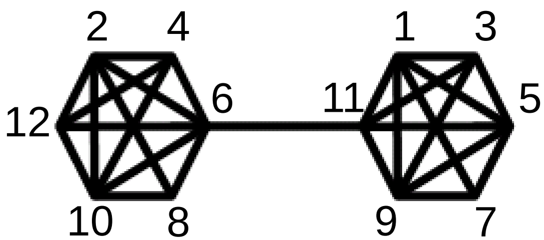


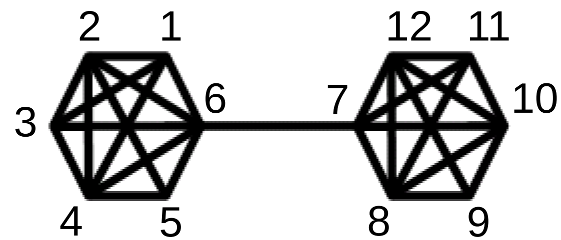
	1	2	3	4	5	6	7	8	9	10	11	12
1			1		1		1		1		1	
2				1		1		1		1		1
3	1				1		1		1		1	
4		1				1		1		1		1
5	1		1				1		1		1	
6		1		1				1		1	1	1
7	1		1		1				1		1	
8		1		1		1				1		1
9	1		1		1		1				1	
10		1		1		1		1				1
11	1		1		1	1	1		1			
12		1		1		1		1		1		

	1	2	3	4	5	6	7	8	9	10	11	12
1		1	1	1	1	1						
2	1		1	1	1	1						
3	1	1		1	1	1						
4	1	1	1		1	1						
5	1	1	1	1		1						
6	1	1	1	1	1		1					
7						1		1	1	1	1	1
8							1		1	1	1	1
9								1		1	1	1
10								1	1		1	1
11								1	1	1		1
12								1	1	1	1	

Good ordering = Good compression

- Same graphs with different orderings





Many sparse blocks

	1	2	3	4	5	6	7	8	9	10	11	12
1			1		1		1		1		1	
2				1		1		1		1		1
3	1				1		1		1		1	
4		1				1		1		1		1
5	1		1				1		1		1	
6		1		1				1		1	1	1
7	1		1		1				1		1	
8		1		1		1				1		1
9	1		1		1		1				1	
10		1		1		1		1				1
11	1		1		1	1	1		1			
12		1		1		1		1		1		

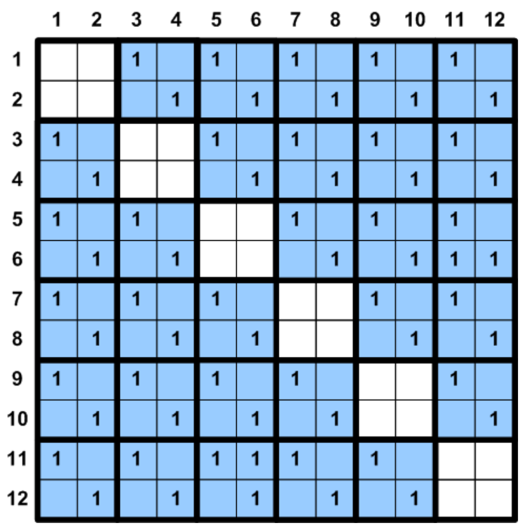
	1	2	3	4	5	6	7	8	9	10	11	12
1		1	1	1	1	1						
2	1		1	1	1	1						
3	1	1		1	1	1						
4	1	1	1		1	1						
5	1	1	1	1		1						
6	1	1	1	1	1		1					
7						1		1	1	1	1	1
8							1		1	1	1	1
9								1		1	1	1
10								1	1		1	1
11								1	1	1		1
12								1	1	1	1	

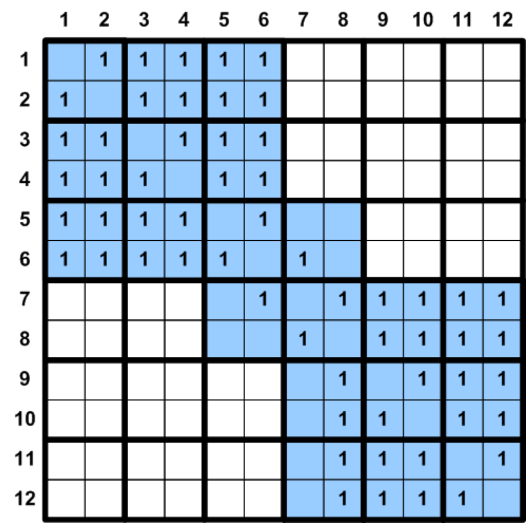
Few dense blocks

Problem Definition

- Given a graph, how can we lay-out its edges so that nonzero elements are well-clustered?
- Better clustering = better compression

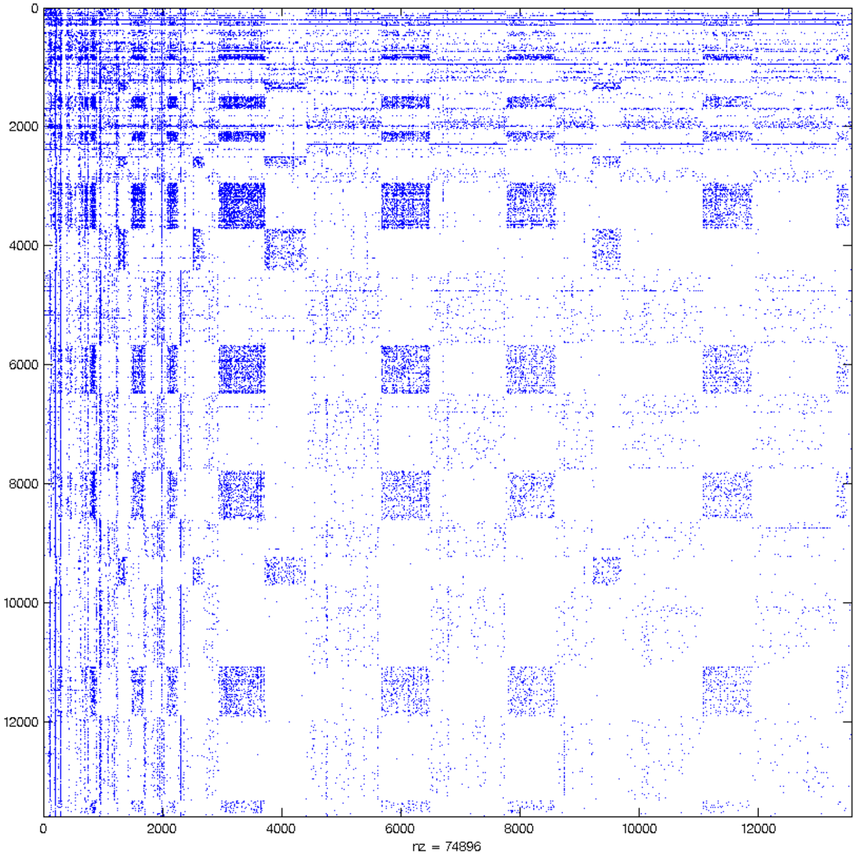
Many sparse blocks

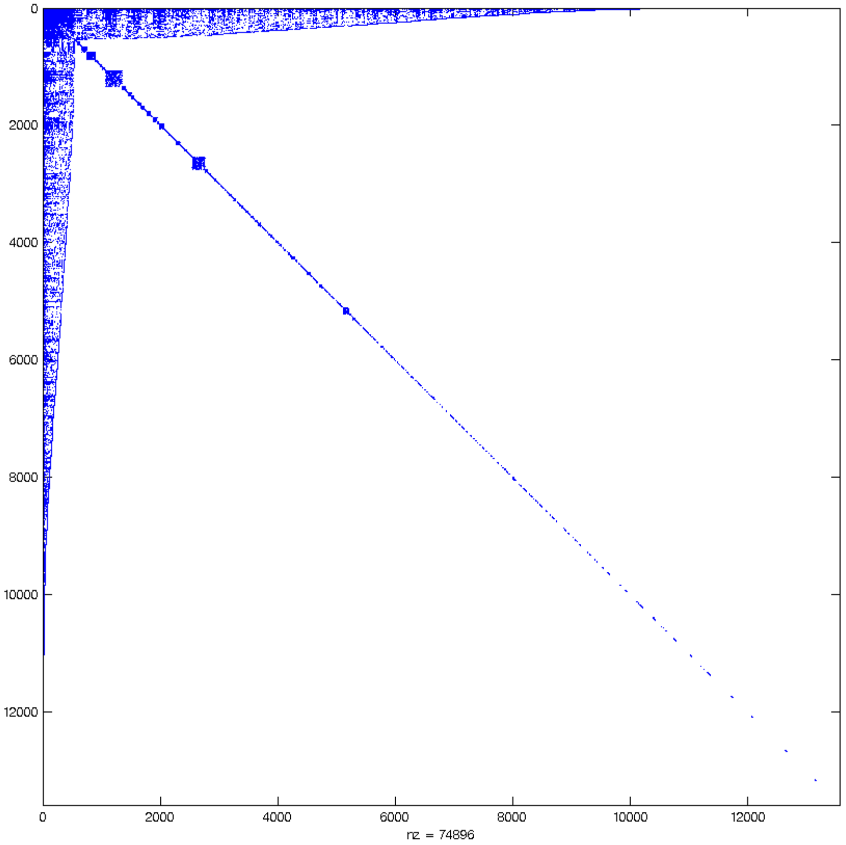




Few dense blocks

Main Result



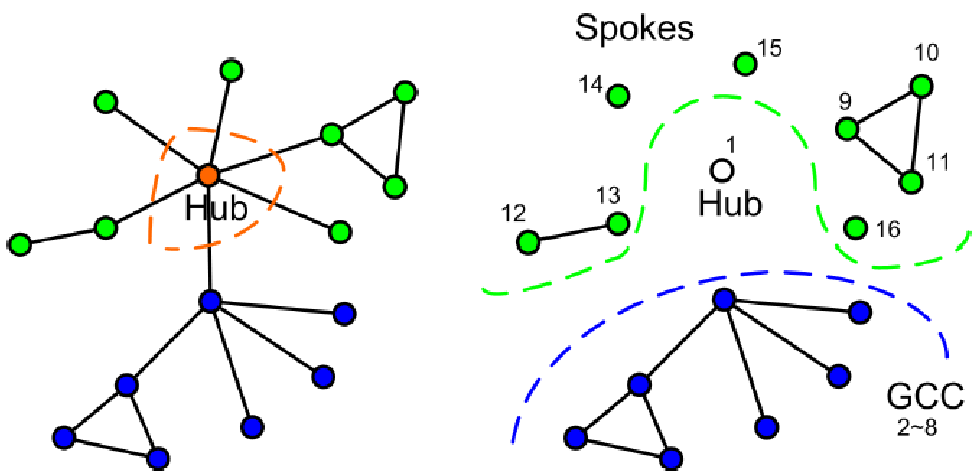


Original **SlashBurn**

Slash-Burn method

Details

- ‘Slash’ the top k hubs, and ‘burn’ the edges
- Move k hubs to the front of the row/column,
  non-GCC to the back of the row/column
- Continue on the remaining GCC

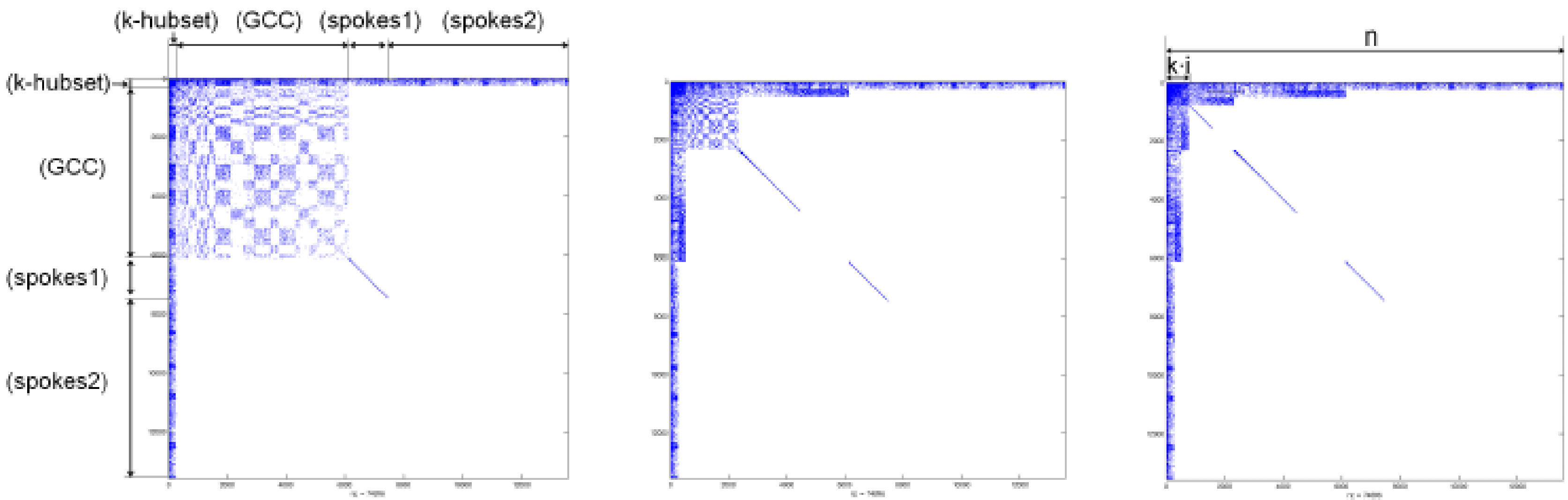


(a) Before SlashBurn (b) After SlashBurn

Slash-Burn method

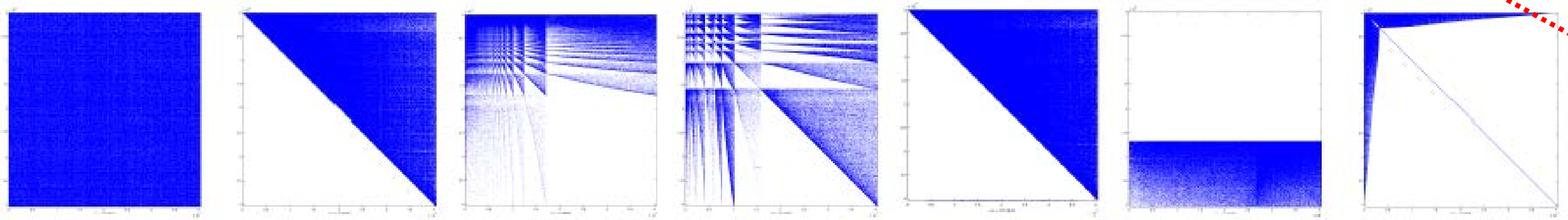
Details

- ‘Slash’ the top k hubs, and ‘burn’ the edges
- Move k hubs to the front of the row/column,
  non-GCC to the back of the row/column
- Continue on the remaining GCC

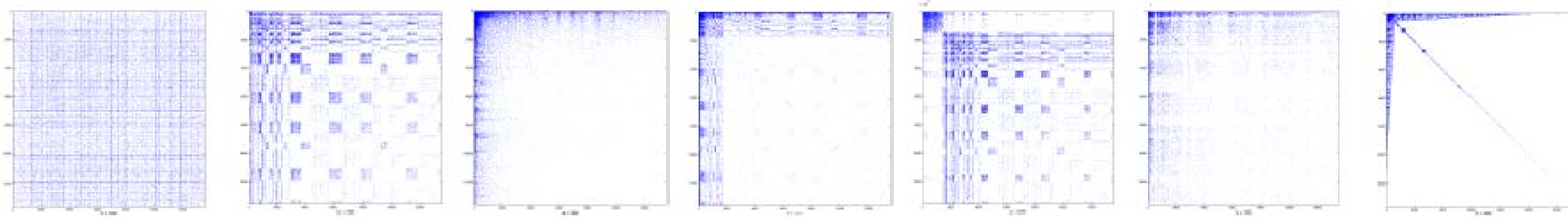


(a) AS-Oregon after 1 iteration (b) .. after 1 more iteration (c) .. after 1 more iteration

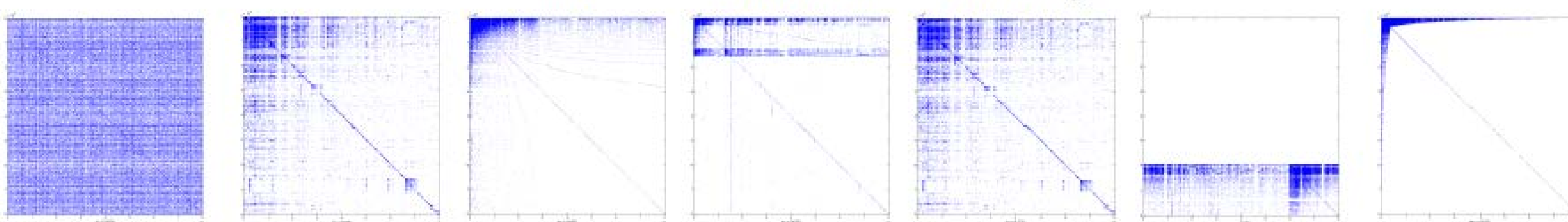
Spyplots



Flickr:
(a) Random (b) Natural (c) Degree Sort (d) Cross Association (e) Spectral Clustering (f) Shingle (g) SLASHBURN



AS-Oregon:
(a) Random (b) Natural (c) Degree Sort (d) Cross Association (e) Spectral Clustering (f) Shingle (g) SLASHBURN

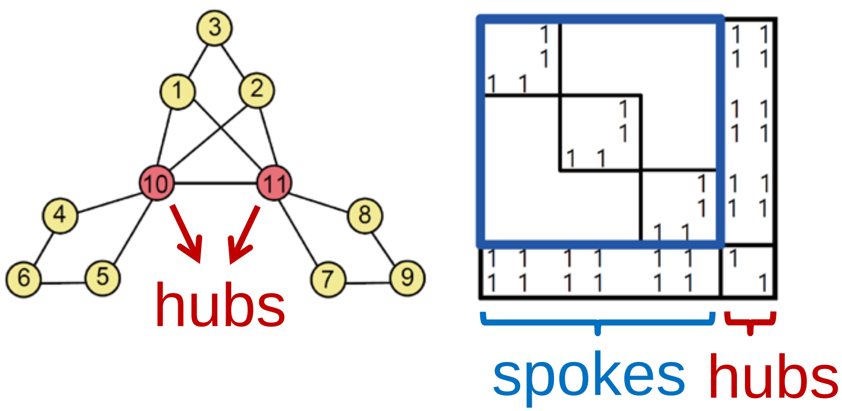


Enron:
(a) Random (b) Natural (c) Degree Sort (d) Cross Association (e) Spectral Clustering (f) Shingle (g) SLASHBURN

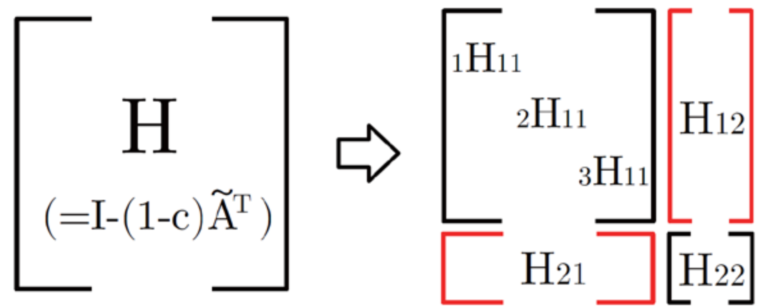
End of Aside

Preprocessing Phase

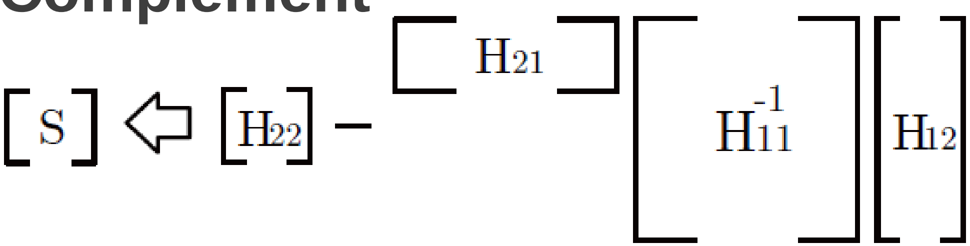
1. Reordering



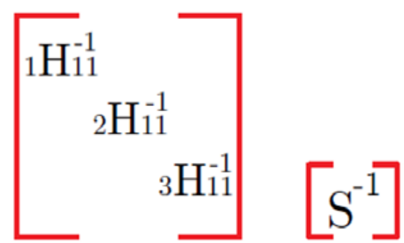
2. Partitioning



3. Schur Complement

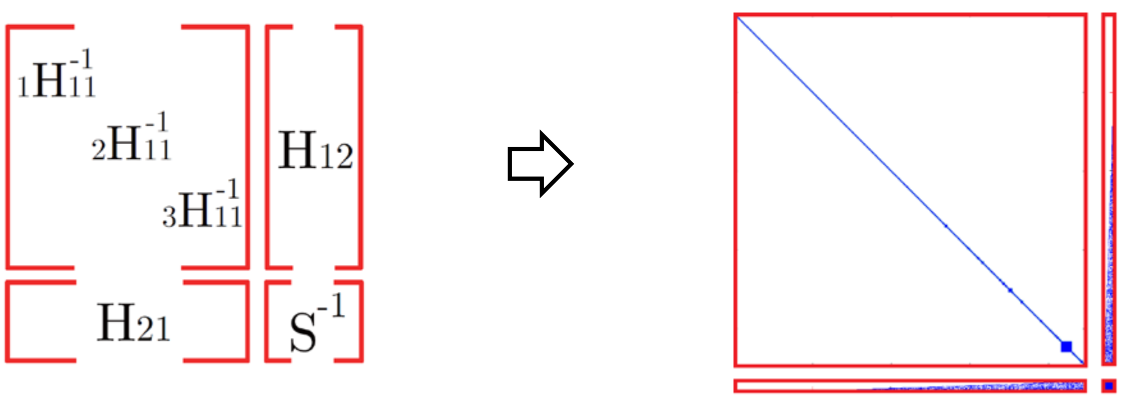


4. Inverting



Preprocessing Phase: Output

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



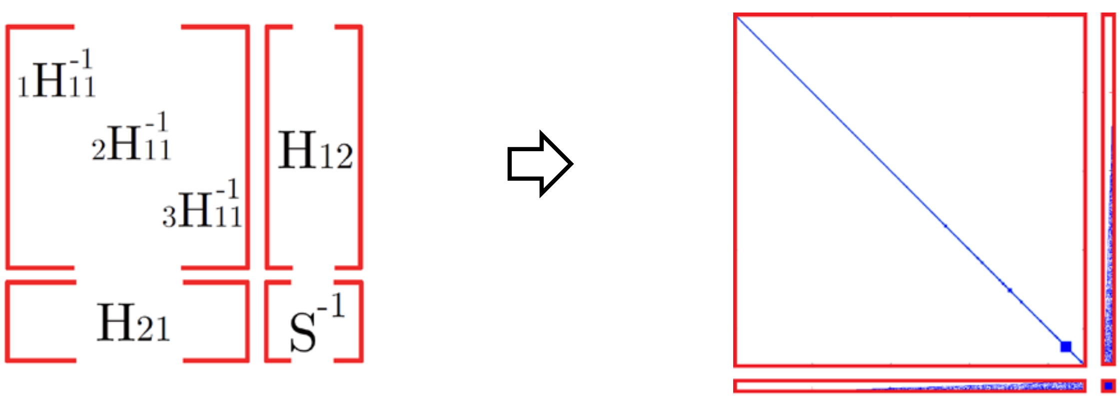
Query Phase

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

$$
\begin{bmatrix} r \end{bmatrix} \Leftarrow \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}, \quad
\begin{bmatrix} r_1 \end{bmatrix} \Leftarrow \begin{bmatrix} H_{11}^{-1} \end{bmatrix} \left(\begin{bmatrix} q_1 \end{bmatrix} - \begin{bmatrix} H_{12} \end{bmatrix} \begin{bmatrix} r_2 \end{bmatrix} \right), \quad
\begin{bmatrix} r_2 \end{bmatrix} \Leftarrow \begin{bmatrix} S^{-1} \end{bmatrix} \begin{bmatrix} q_2 \end{bmatrix} - \begin{bmatrix} H_{21} \end{bmatrix} \begin{bmatrix} H_{11}^{-1} \end{bmatrix} \begin{bmatrix} q_1 \end{bmatrix}
$$

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

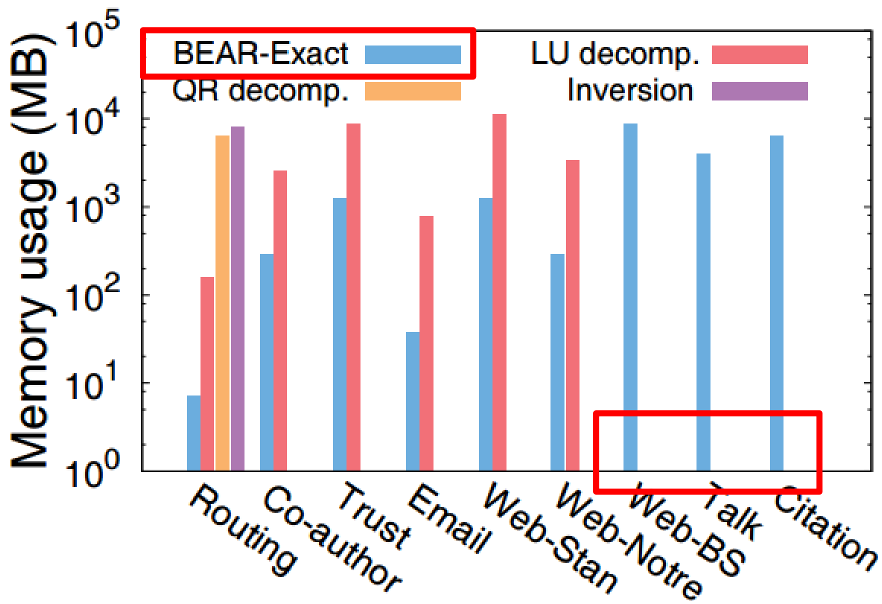
dataset	#nodes	#edges
Routing	22,963	48,436
Co-author	31,163	120,029
Trust	131,828	841,372
Email	265,214	420,045
Web-Stan	281,903	2,312,497
Web-Notre	325,729	1,497,134
Web-BS	685,230	7,600,595
Talk	2,394,385	5,021,410
Citation	3,774,768	16,518,948

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)

Q1. Space Efficiency

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

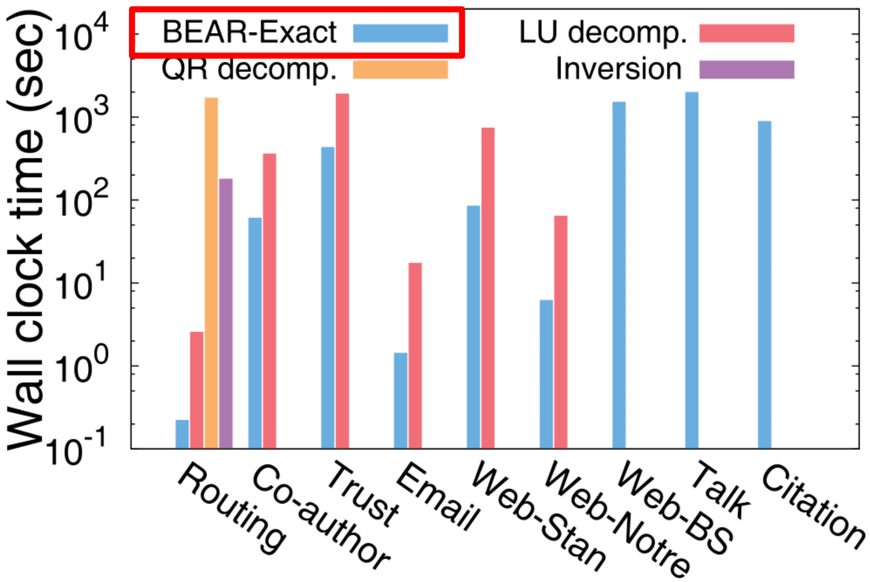


Space for preprocessed data

Up to 22x less memory space than competitors

Q2. Preprocessing Time

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

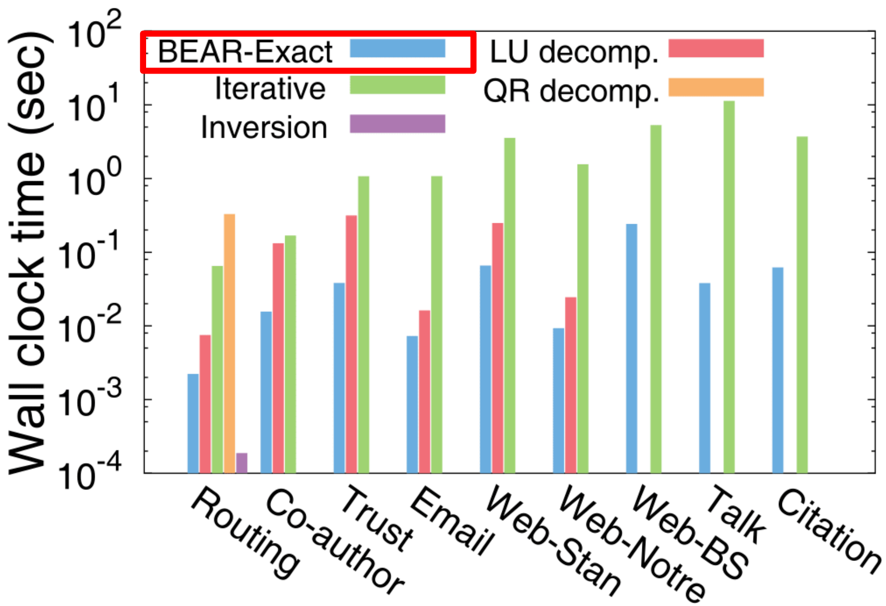


Preprocessing time of exact methods

Up to 12x less preprocessing time than other methods

Q3. Query Time

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



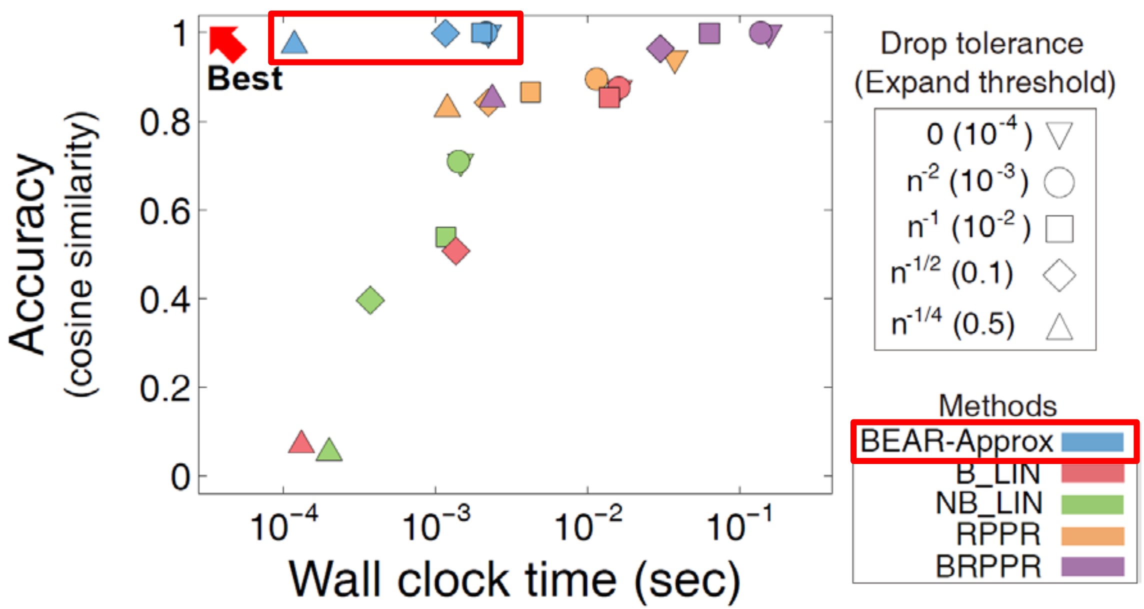
Query time of exact methods

Up to 8x less query time than LU decomp.

Up to 300x less query time than Iterative method

Q4. Speed vs Accuracy

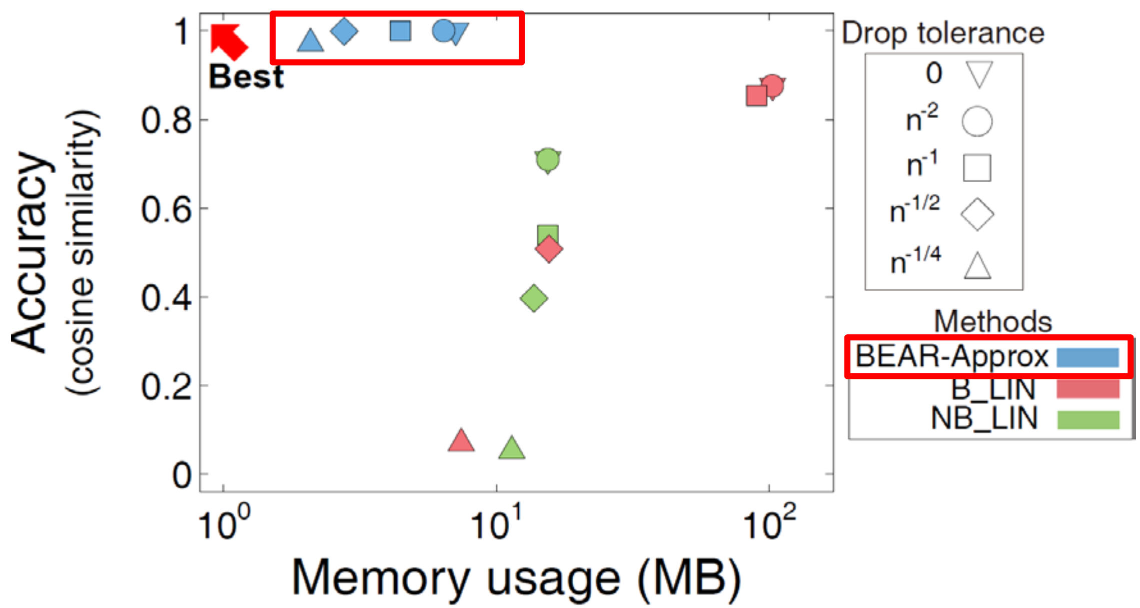
- Does **BEAR-Approx** provide a better trade-off between speed and accuracy than other methods?



Query speed v.s. Accuracy on the Routing dataset

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

Conclusion: BEAR

- **BEAR** (**B**lock **E**limination **A**pproach for **R**WR)
  - partitions the adjacency matrix into small submatrices using the *hub-and-spoke* structure of real-world graphs
  - computes RWR scores accurately from the submatrices using *block elimination*
- **BEAR-Exact**
  - up to 22× less space, 12× less preprocessing time, and 8× less query time than other exact methods
- **BEAR-Approx**
  - better trade-off between time, space, and accuracy than other approximate methods

http://datalab.snu.ac.kr/bear

BePI: Fast and Memory-Efficient Method for Billion-Scale Random Walk with Restart (SIGMOD 2017)

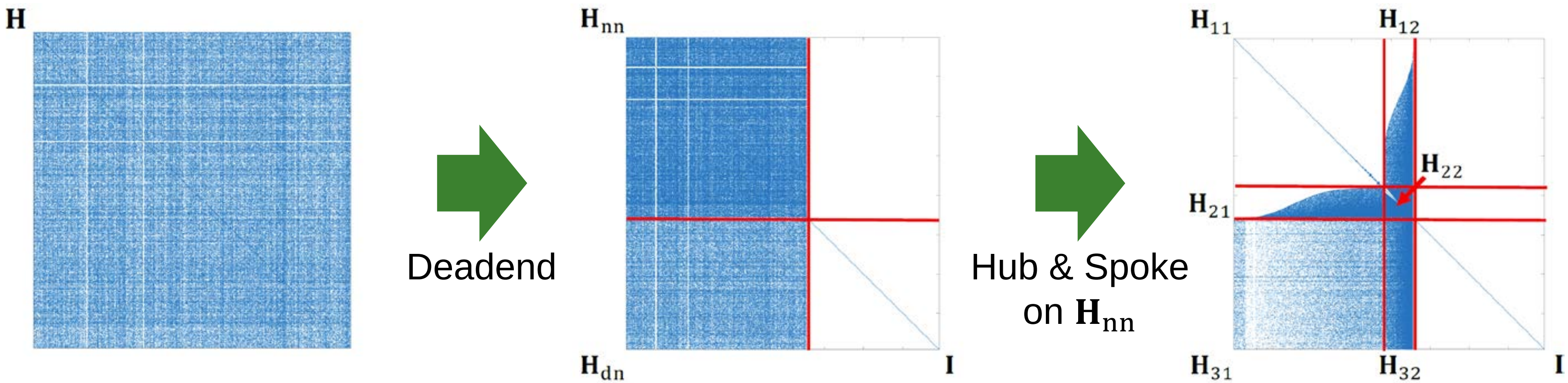
http://datalab.snu.ac.kr/bepi

Proposed Method

- BePI (Best of Preprocessing and Iterative approaches)
  - A fast and scalable method by taking the advantages of both preprocessing and iterative methods
- Key Ideas
  - **Idea 1) Exploit graph characteristics** to adopt a preprocessing approach for fast query speed
  - **Idea 2) Incorporate an iterative method into the preprocessing approach** to increase the scalability
  - **Idea 3) Optimize the performance of the iterative method** to accelerate RWR computation speed
    - (Omitted for brevity; see the paper)

Proposed Method – Idea 1

- **Combine deadend** and hub & spoke reordering



$\mathbf{H}_{11}$ **is a block diagonal matrix!**

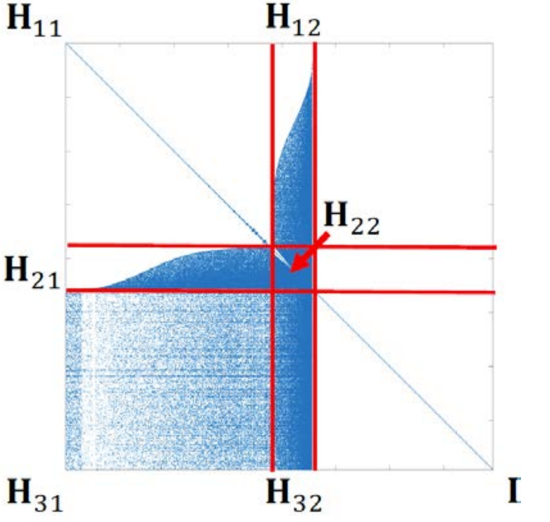
$$\mathbf{H}\mathbf{r} = c\mathbf{q}_s \Leftrightarrow \begin{bmatrix} \boxed{\mathbf{H}_{11}} & \mathbf{H}_{12} & \mathbf{0} \\ \mathbf{H}_{21} & \mathbf{H}_{22} & \mathbf{0} \\ \mathbf{H}_{31} & \mathbf{H}_{32} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \end{bmatrix} = c \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \mathbf{q}_3 \end{bmatrix}$$

Proposed Method – Idea 2

- Incorporate an iterative method into the preprocessing approach
  - Computing $\boldsymbol{H}_{\mathbf{11}}^{-\mathbf{1}}$ is trivial since it is block diagonal
  - But, inverting $\boldsymbol{S}$ is impractical in very large graphs
    - $\dim(\boldsymbol{S})$ = # of hubs > 1 million (10^6) in large graphs
    - e.g., 10 million hubs in the Twitter network

$$\begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{11}^{-1}(c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2) \\ \mathbf{S}^{-1}(c\mathbf{q}_2 - c\mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{q}_1) \\ c\mathbf{q}_3 - \mathbf{H}_{31}\mathbf{r}_1 - \mathbf{H}_{32}\mathbf{r}_2 \end{bmatrix}$$

$$\mathbf{S} = \mathbf{H}_{22} - \mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12}$$



Proposed Method – Idea 2

- Incorporate an iterative method into the preprocessing approach
  - **Solution.** Solve the linear system on ***S*** using ***an iterative linear solver*** such as GMRES [Saad et al., `86]

$$\mathbf{r}_2 = \mathbf{S}^{-1}(c\mathbf{q}_2 - c\mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{q}_1)$$

$$\Leftrightarrow \mathbf{S}\mathbf{r}_2 = c\mathbf{q}_2 - c\mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{q}_1 \triangleq \tilde{\mathbf{q}}_2$$

    - Linear solvers obtain the accurate $\boldsymbol{r}_2$ without inverting $\mathbf{S}$

$$\mathbf{S}\mathbf{r}_2 = \tilde{\mathbf{q}}_2$$

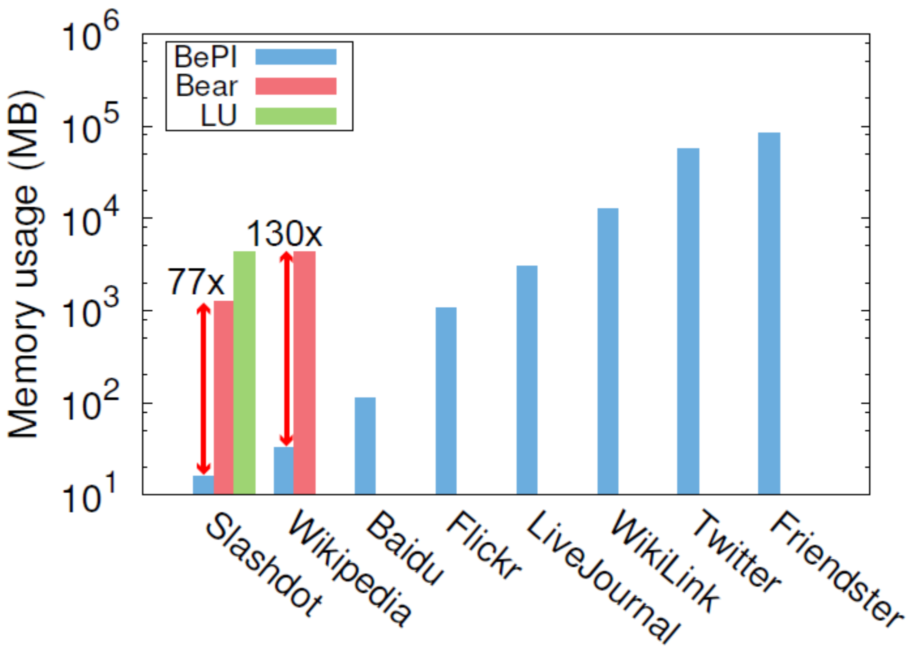
Introducing the linear solver increases the scalability of RWR computation!

Experimental Questions

- Q1. (Space) How much memory space does BePI requires for their preprocessed results?
- Q2. (Prep. Time) How long does the preprocessing phase of BePI take?
- Q3. (Query Time) How quickly does BePI respond to an RWR query?
- Q4. (Scalability) How well does BePI scale up?

Q1. Space Efficiency

- How much memory space does BePI requires for their preprocessed results?

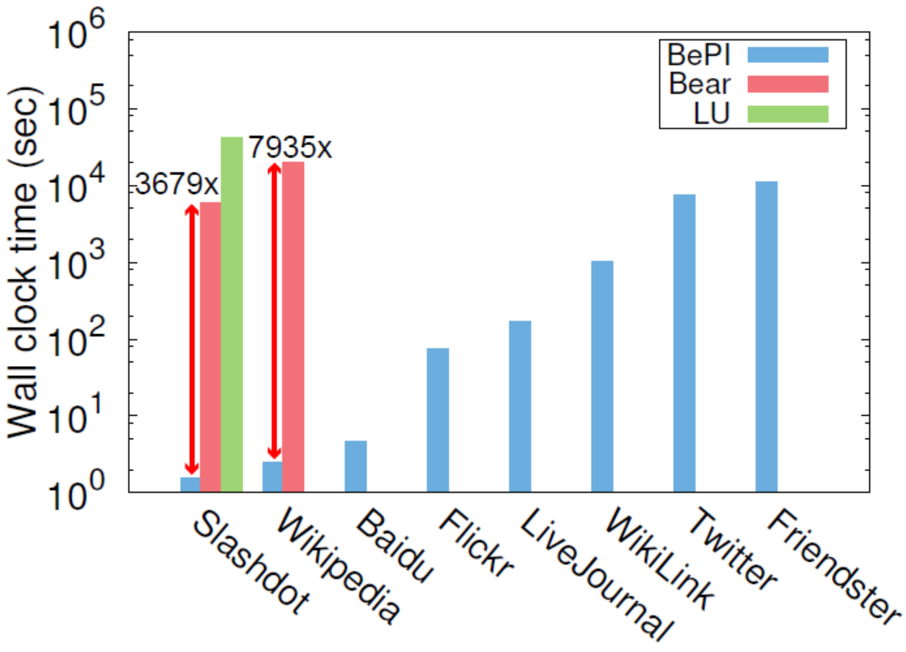


Memory space for preprocessed data

BePI is up to 130 × less memory space than other preprocessing methods!

Q2. Preprocessing Time

- How long does the preprocessing phase of BePI take?

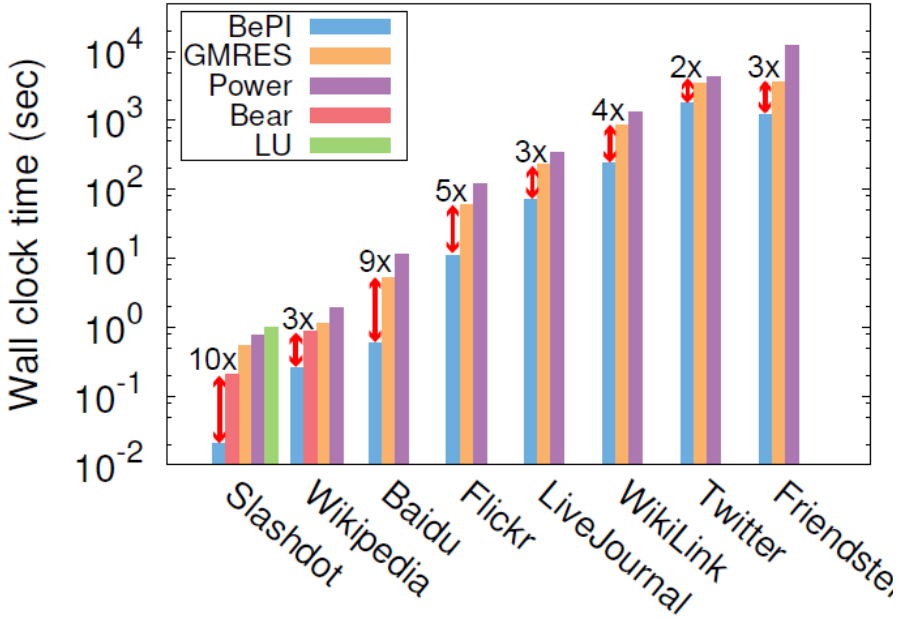


Preprocessing time

BePI is significantly faster than other methods in terms of preprocessing time!

Q3. Query Time

- How quickly does BePI respond to an RWR query?

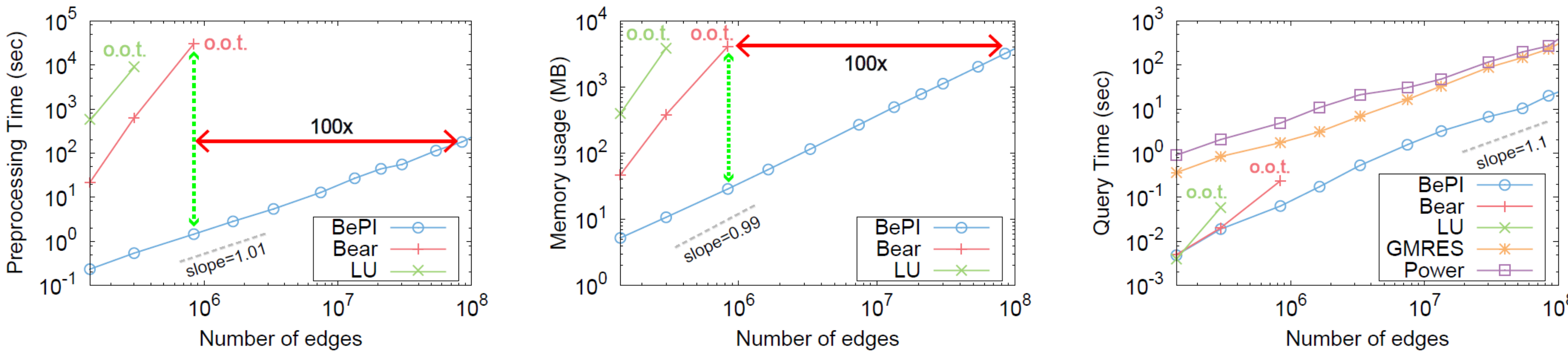


(c) Query time

BePI is up to 9× faster than other competitors in terms of query speed!

Q4. Scalability of BePI

- How well does BePI scale up?
  - Processes **100** × larger graphs than other preprocessing methods
  - Shows the fastest RWR computation speed among others



(a) Preprocessing time (b) Space for preprocessed data (c) Query time

BePI shows the best performance in terms of scalability and running time!

Conclusion: BePI

- BePI (Best of Preprocessing and Iterative approaches)
  - **Idea 1)** Exploit graph characteristics for a prep. method
  - **Idea 2)** Incorporate an iterative method into the prep. method
  - **Idea 3)** Optimize the performance of the iterative method
- Main Results
  - Fast and scalable computation for RWR on billion-scale graphs
  - Requires **130**× less memory space & processes **100** × larger graphs than other preprocessing methods
  - Computes RWR scores **9** × faster than other existing methods

http://datalab.snu.ac.kr/bepi

Outline

- ☑ Random Walk with Restart (RWR)
- ☑ Fast Exact RWR
- ☐ **Fast Approximate RWR**
- ☐ Conclusions

Overview

- I will describe two state-of-the-art approximate RWR algorithms
  - Static method TPA (to appear at ICDE 2018)
  - Dynamic method OSP (to appear at WWW 2018)

TPA: Fast, Scalable, and Accurate Method for Approximate Random Walk with Restart on Billion Scale Graphs (ICDE 2018)

http://datalab.snu.ac.kr/tpa

Problem Definition

- How can we approximately compute RWR quickly, with little loss of accuracy?

CPI: Cumulative Power Iteration

- Exact RWR computation method
- Re-interpretation of RWR
- Propagation of scores across a graph
  1) Score c is generated from the seed node
  2) At each step, scores are divided evenly into out-edges with decaying coefficient $(1 - c)$
  3) Each node accumulates scores they have received
  4) Accumulated scores become RWR score of each node

CPI: Cumulative Power Iteration

$$\mathbf{x}^{(0)} = c\mathbf{q}$$

1) Initial score c at seed node

$$\mathbf{x}^{(i)} = (1-c)\tilde{\mathbf{A}}^\top \mathbf{x}^{(i-1)} = c\left((1-c)\tilde{\mathbf{A}}^\top\right)^i \mathbf{q}$$

2) scores are divided evenly into out-edges with (1-c)

$$\mathbf{r}_{\mathrm{CPI}} = \sum_{i=0}^{\infty} \mathbf{x}^{(i)} = c\sum_{i=0}^{\infty}\left((1-c)\tilde{\mathbf{A}}^\top\right)^i \mathbf{q}$$

3) CPI accumulate interim scores of each node to get final results

- $\mathbf{x}(i) \in \mathbb{R}^{n\times 1}$: interim score vector computed from i th iteration
- Correctness of CPI: Theorem 1
- For PageRank computation, the seed vector $\mathbf{q}$ is set to $\frac{1}{n}\mathbf{1}$

TPA: Two Phase Approximation

- TPA approximates RWR scores with fast speed and high accuracy
  - CPI performs iterations until convergence
  - Divide the whole iterations in CPI into three parts as follows :

$$
\begin{aligned}
&\mathbf{r}_{\text{CPI}} \\
&= \mathbf{r}_{\text{family}} + \mathbf{r}_{\text{neighbor}} + \mathbf{r}_{\text{stranger}} \\
&= \underbrace{\mathbf{x}^{(0)} + \cdots + \mathbf{x}^{(S-1)}}_{\text{family part}} + \underbrace{\mathbf{x}^{(S)} + \cdots + \mathbf{x}^{(T-1)}}_{\text{neighbor part}} + \underbrace{\mathbf{x}^{(T)} + \cdots}_{\text{stranger part}}
\end{aligned}
$$

S : starting iteration of $r_{neighbor}$, T : starting iteration of $r_{stranger}$

TPA: Two Phase Approximation

$$
\begin{aligned}
&\mathbf{r}_{\text{CPI}} \\
&= \mathbf{r}_{\text{family}} + \mathbf{r}_{\text{neighbor}} + \mathbf{r}_{\text{stranger}} \\
&= \underbrace{\mathbf{x}^{(0)} + \cdots + \mathbf{x}^{(S-1)}}_{\text{family part}} + \underbrace{\mathbf{x}^{(S)} + \cdots + \mathbf{x}^{(T-1)}}_{\text{neighbor part}} + \underbrace{\mathbf{x}^{(T)} + \cdots}_{\text{stranger part}}
\end{aligned}
$$

$$
\mathbf{r}_{\text{TPA}} = \mathbf{r}_{\text{family}} + \tilde{\mathbf{r}}_{\text{neighbor}} + \tilde{\mathbf{r}}_{\text{stranger}}
$$

- 1st Phase: Stranger Approximation
  - Approximates $r_{stranger}$ in RWR using PageRank
- 2nd Phase: Neighbor Approximation
  - Approximates $r_{neighbor}$ using r_{family}

Stranger Approximation - Definition

- PageRank score vector p_{CPI} is represented by CPI as follows:

$$\mathbf{x}'^{(0)} = \frac{c}{n}\mathbf{1} \quad \mathbf{x}'^{(i)} = (1-c)\tilde{\mathbf{A}}^\top \mathbf{x}'^{(i-1)}$$

$$\begin{aligned}&\mathbf{p}_{\text{CPI}}\\&= \mathbf{p}_{\text{family}} + \mathbf{p}_{\text{neighbor}} + \mathbf{p}_{\text{stranger}}\\&= \underbrace{\mathbf{x}'^{(0)} + \cdots + \mathbf{x}'^{(S-1)}}_{\text{family part}} + \underbrace{\mathbf{x}'^{(S)} + \cdots + \mathbf{x}'^{(T-1)}}_{\text{neighbor part}} + \underbrace{\mathbf{x}'^{(T)} + \cdots}_{\text{stranger part}}\end{aligned}$$

- $r_{stranger}$ in RWR is approximated by $p_{stranger}$ in PageRank as follows:

$$\tilde{\mathbf{r}}_{\text{stranger}} = \mathbf{p}_{\text{stranger}}$$

Stranger Approximation - Intuition

- The amount of scores propagated into each node

1. # of in-edges
    - Nodes with many in-edges have many sources to receive scores
2. Distance from seed node
    - Scores are decayed by factor $(1-c)$ as iteration progresses
    - Nodes close to the seed node take in high scores

Stranger Approximation - Intuition

- In stranger iterations
  - Scores $(x(T), x(T+1), \cdots)$ are mainly determined by # in-edges
  - **Nodes are already far from seed**
- PageRank is solely determined by arrangement of edges (= # in-edges) !!
  - Motivation of Stranger Approximation
  - Estimate stranger iterations in RWR with those in PageRank
- Precompute $\tilde{r}_{stranger}$ in preprocessing phase

TPA: Two Phase Approximation

$$
\begin{aligned}
&\mathbf{r}_{\text{CPI}} \\
&= \mathbf{r}_{\text{family}} + \mathbf{r}_{\text{neighbor}} + \mathbf{r}_{\text{stranger}} \\
&= \underbrace{\mathbf{x}^{(0)} + \cdots + \mathbf{x}^{(S-1)}}_{\text{family part}} + \underbrace{\mathbf{x}^{(S)} + \cdots + \mathbf{x}^{(T-1)}}_{\text{neighbor part}} + \underbrace{\mathbf{x}^{(T)} + \cdots}_{\text{stranger part}}
\end{aligned}
$$

$$
\mathbf{r}_{\text{TPA}} = \mathbf{r}_{\text{family}} + \tilde{\mathbf{r}}_{\text{neighbor}} + \tilde{\mathbf{r}}_{\text{stranger}}
$$

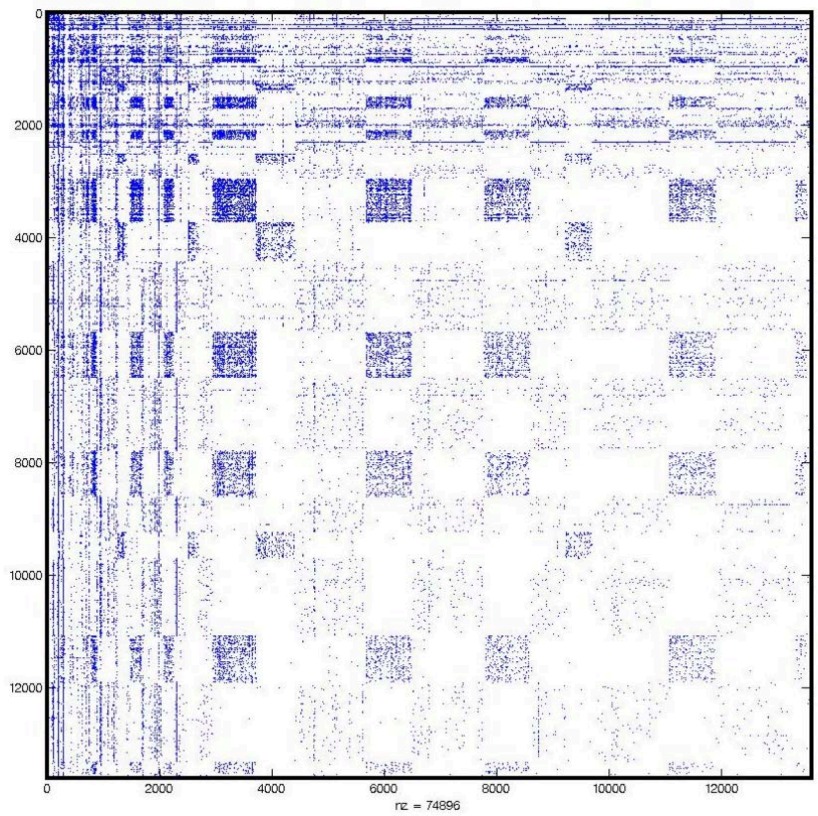
- 1st Phase: Stranger Approximation
  - Approximates $r_{stranger}$ in RWR using PageRank
- 2nd Phase: Neighbor Approximation
  - Approximates $r_{neighbor}$ using r_{family}

Neighbor Approximation - Definition

- The neighbor approximation
  - Limit computation to r_{family}
  - Estimate $r_{neighbor}$ by scaling r_{family} as follows:

$$\tilde{\mathbf{r}}_{\text{neighbor}} = \frac{\|\mathbf{r}_{\text{neighbor}}\|_1}{\|\mathbf{r}_{\text{family}}\|_1}\mathbf{r}_{\text{family}} = \frac{(1-c)^S - (1-c)^T}{1-(1-c)^S}\mathbf{r}_{\text{family}}$$

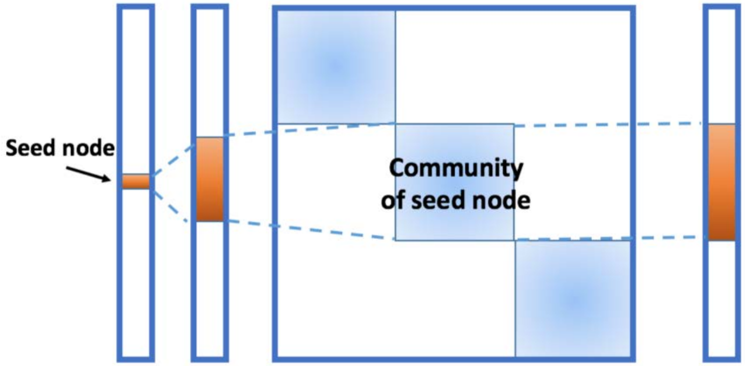
Neighbor Approximation - Intuition



Block-wise, Community-like structure of real-world graphs[1]

[1] U. Kang and C. Faloutsos. Beyond ‘caveman communities’: Hubs and spokes for graph compression and mining. In *ICDM*, 2011

Neighbor Approximation - Intuition



- Nodes which receive scores in the early iterations (family part)
  - Would receive scores again in the following iterations (neighbor part)
- Nodes which have more in-edges thus receive more scores in the early iterations
  - Would receive more scores than other nodes in the following iterations.

TPA: Two Phase Approximation

Exact RWR: $\mathbf{r}_{\text{CPI}} = \mathbf{r}_{\text{family}} + \mathbf{r}_{\text{neighbor}} + \mathbf{r}_{\text{stranger}}$

Preprocessing phase

$\mathbf{p}_{\text{CPI}} = \mathbf{p}_{\text{family}} + \mathbf{p}_{\text{neighbor}} + \mathbf{p}_{\text{stranger}}$

$\tilde{\mathbf{r}}_{\text{stranger}} \leftarrow \mathbf{p}_{\text{stranger}}$ **: Stranger approximation**

Online phase

Compute $\mathbf{r}_{\text{family}}$ **using CPI**

$\tilde{\mathbf{r}}_{\text{neighbor}} \leftarrow \frac{\|\mathbf{r}_{\text{neighbor}}\|_1}{\|\mathbf{r}_{\text{family}}\|_1} \mathbf{r}_{\text{family}}$ **: Neighbor approximation**

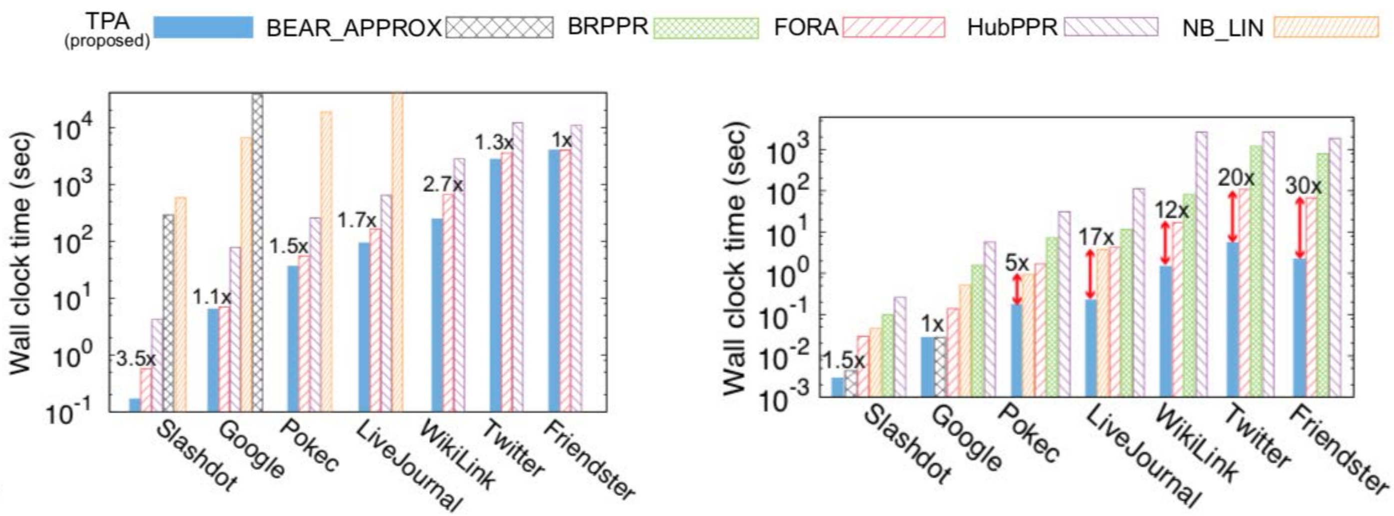
Approximate RWR: $\mathbf{r}_{\text{TPA}} = \mathbf{r}_{\text{family}} + \tilde{\mathbf{r}}_{\text{neighbor}} + \tilde{\mathbf{r}}_{\text{stranger}}$

Experimental Questions

- Q1. Performance
  - How much does TPA enhance the computational efficiency compared with its competitors?
- Q2. Accuracy
  - How much does TPA sacrifice accuracy?

Q1: Performance of TPA- Speed

How long does **TPA** take for its preprocessing phase and online phase, respectively?

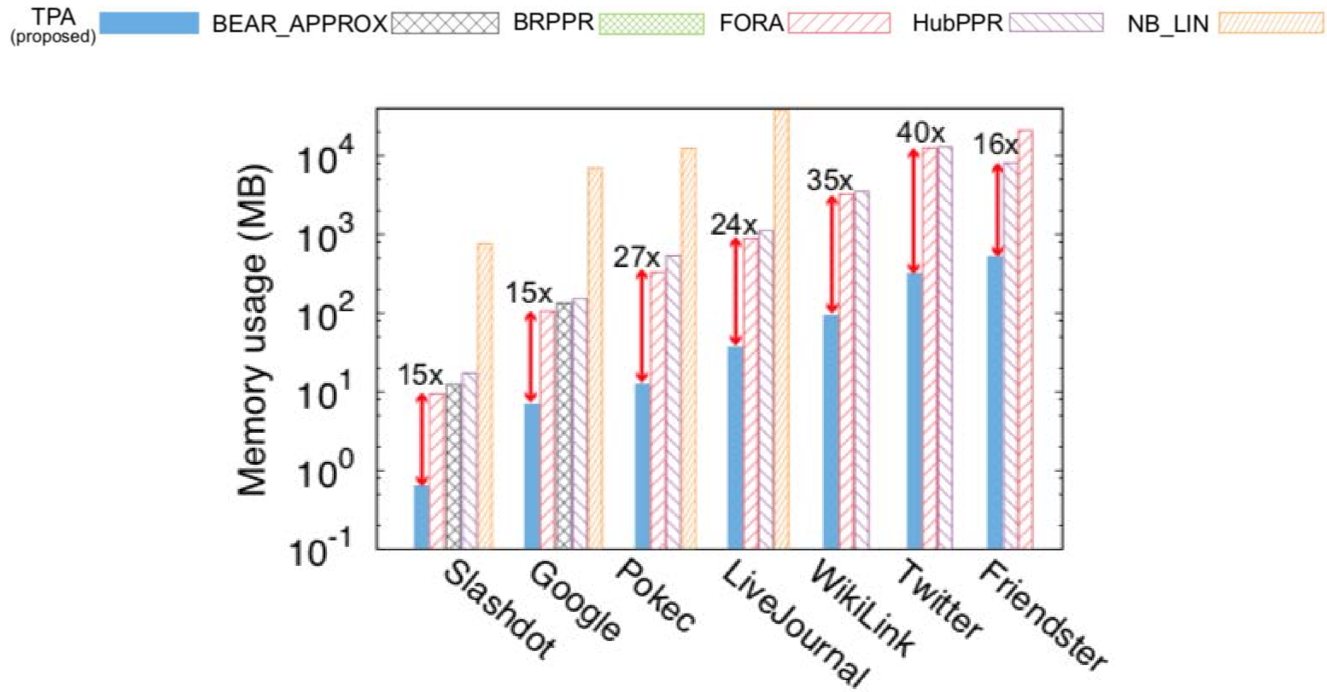


(a) Preprocessing Time **(b) Online Time**

TPA takes smaller running time in both preprocessing and online phases (up to 30x)

Q1: Performance of TPA- Memory

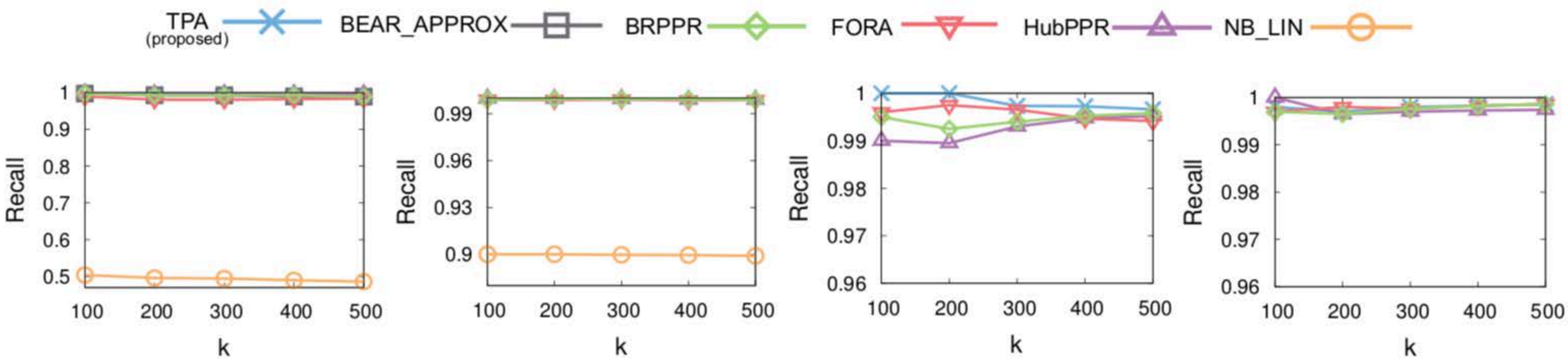
How much memory space does **TPA** requires for preprocessed results?



TPA requires up to 40x smaller memory space than competitors

Q2: Accuracy of TPA

How much does **TPA** sacrify its accuracy?



(a) Slashdot (b) Pokec (c) WikiLink (d) Twitter

TPA provides the best accuracy among competitors!

Conclusion: TPA

- TPA (Two Phase Approximation)
  - Neighbor Approximation
    - block-wise structure of real-world graphs
  - Stranger Approximation
    - PageRank
- Main Results
  - Requires 40x memory space & preprocesses 3.5x time than other preprocessing methods
  - Computes RWR scores 30x faster than other existing methods in online phase
  - Maintaining high accuracy

http://datalab.snu.ac.kr/tpa

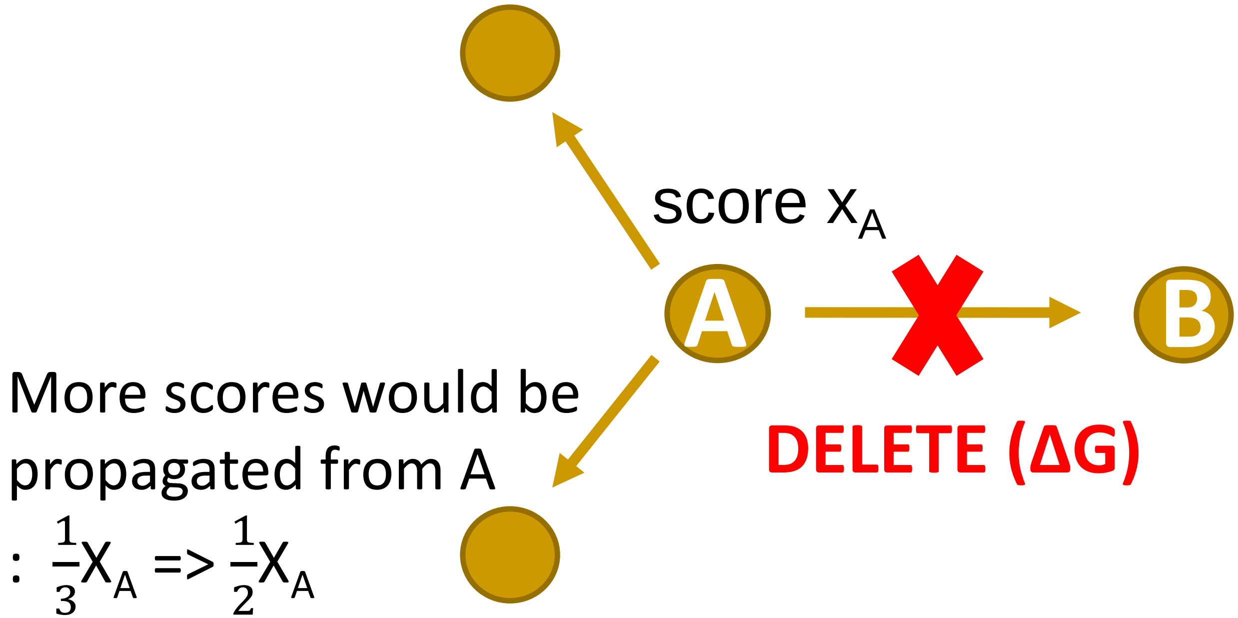
Fast and Accurate Random Walk with Restart on Dynamic Graphs with Guarantees
(WWW 2018)

http://datalab.snu.ac.kr/osp

Problem Definition

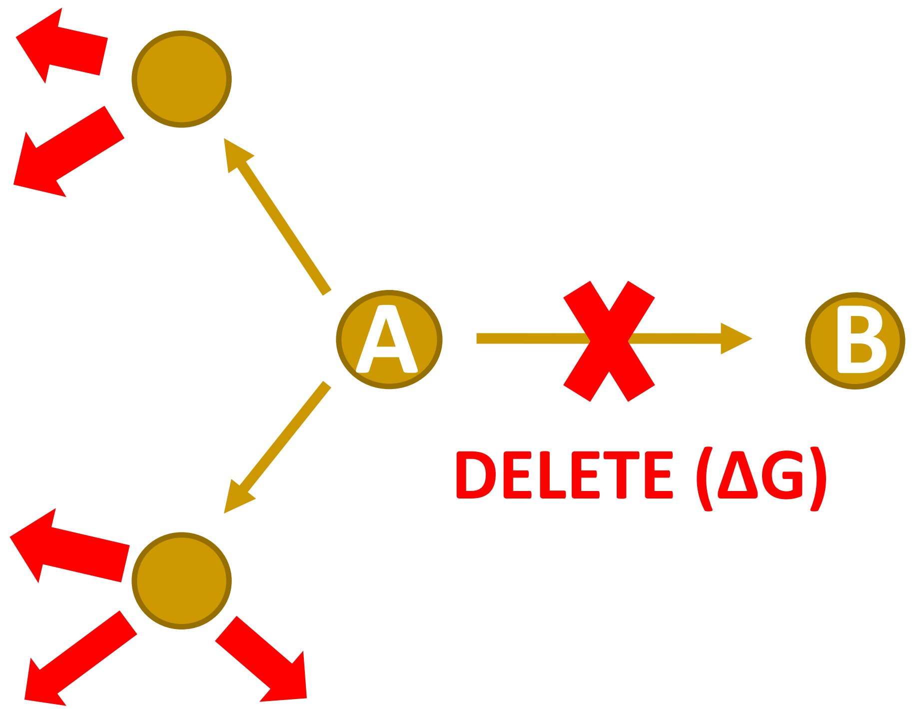
- How can we approximately compute RWR quickly, for dynamic graphs?
  - Dynamic graphs: nodes/edges are added/removed continuously
  - We want to update RWR scores quickly, without computing it from scratch for graph update

Score Propagation on dynamic graph



- RWR scores of nodes are determined by arrangement of edges
  1. When the graph G is updated with ΔG
  2. Propagation of scores around ΔG is changed

Score Propagation on dynamic graph



3. These small changes are propagated
4. Affect previous propagation pattern across whole graph
5. Finally lead to $\mathbf{r}_{\text{new}}$ different from $\mathbf{r}_{\text{old}}$

OSP: Offset Score Propagation

$$\mathbf{q}_{\text{offset}} \leftarrow (1-c)(\tilde{\mathbf{B}}^{\top} - \tilde{\mathbf{A}}^{\top})\mathbf{r}_{\text{old}} = (1-c)(\Delta\mathbf{A})^{\top}\mathbf{r}_{\text{old}}$$

$$\mathbf{x}_{\text{offset}}^{(i)} \leftarrow ((1-c)\tilde{\mathbf{B}}^{\top})^{i}\mathbf{q}_{\text{offset}}$$

$$\mathbf{r}_{\text{offset}} \leftarrow \sum_{i=0}^{\infty}\mathbf{x}_{\text{offset}}^{(i)} = \sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^{\top})^{i}\mathbf{q}_{\text{offset}}$$

$$\mathbf{r}_{\text{new}} \leftarrow \mathbf{r}_{\text{old}} + \mathbf{r}_{\text{offset}}$$

1. Calculate **an offset seed vector $\mathbf{q}_{offset}$**
2. Propagate the offset scores across G+ΔG to get **an off set score vector $\mathbf{r}_{offset}$**
3. Finally, OSP adds up $\mathbf{r}_{\text{old}}$ and $\mathbf{r}_{\text{offset}}$ to get $\boldsymbol{r}_{new}$

OSP-T: OSP with Trade-off

Algorithm 1: OSP and OSP-T Algorithm

Require: previous RWR score vector: $\mathbf{r}_{\text{old}}$, row-normalized adjacency matrix: $\tilde{\mathbf{A}}$, update in $\tilde{\mathbf{A}}$: $\Delta\mathbf{A}$, restart probability: c, error tolerance: ϵ

Ensure: updated RWR score vector: $\mathbf{r}_{\text{new}}$

set seed offset vector $\mathbf{q}_{\text{offset}} = (1-c)(\Delta\mathbf{A})^\top \mathbf{r}_{\text{old}}$

set $\mathbf{r}_{\text{offset}} = \mathbf{0}$ and $\mathbf{x}^{(0)}_{\text{offset}} = \mathbf{q}_{\text{offset}}$

for iteration $i = 1$; $\|\mathbf{x}^{(i)}_{\text{offset}}\|_1 > \epsilon$; i++ **do**

  compute $\mathbf{x}^{(i)}_{\text{offset}} \leftarrow (1-c)(\tilde{\mathbf{A}} + \Delta\mathbf{A})^\top \mathbf{x}^{(i-1)}_{\text{offset}}$

  compute $\mathbf{r}_{\text{offset}} \leftarrow \mathbf{r}_{\text{offset}} + \mathbf{x}^{(i)}_{\text{offset}}$

end for

$\mathbf{r}_{\text{new}} \leftarrow \mathbf{r}_{\text{old}} + \mathbf{r}_{\text{offset}}$

return $\mathbf{r}_{\text{new}}$

- Approximate method for dynamic RWR
- Use the same algorithm with OSP
- Regulates accuracy and speed using higher error tolerance parameter ε

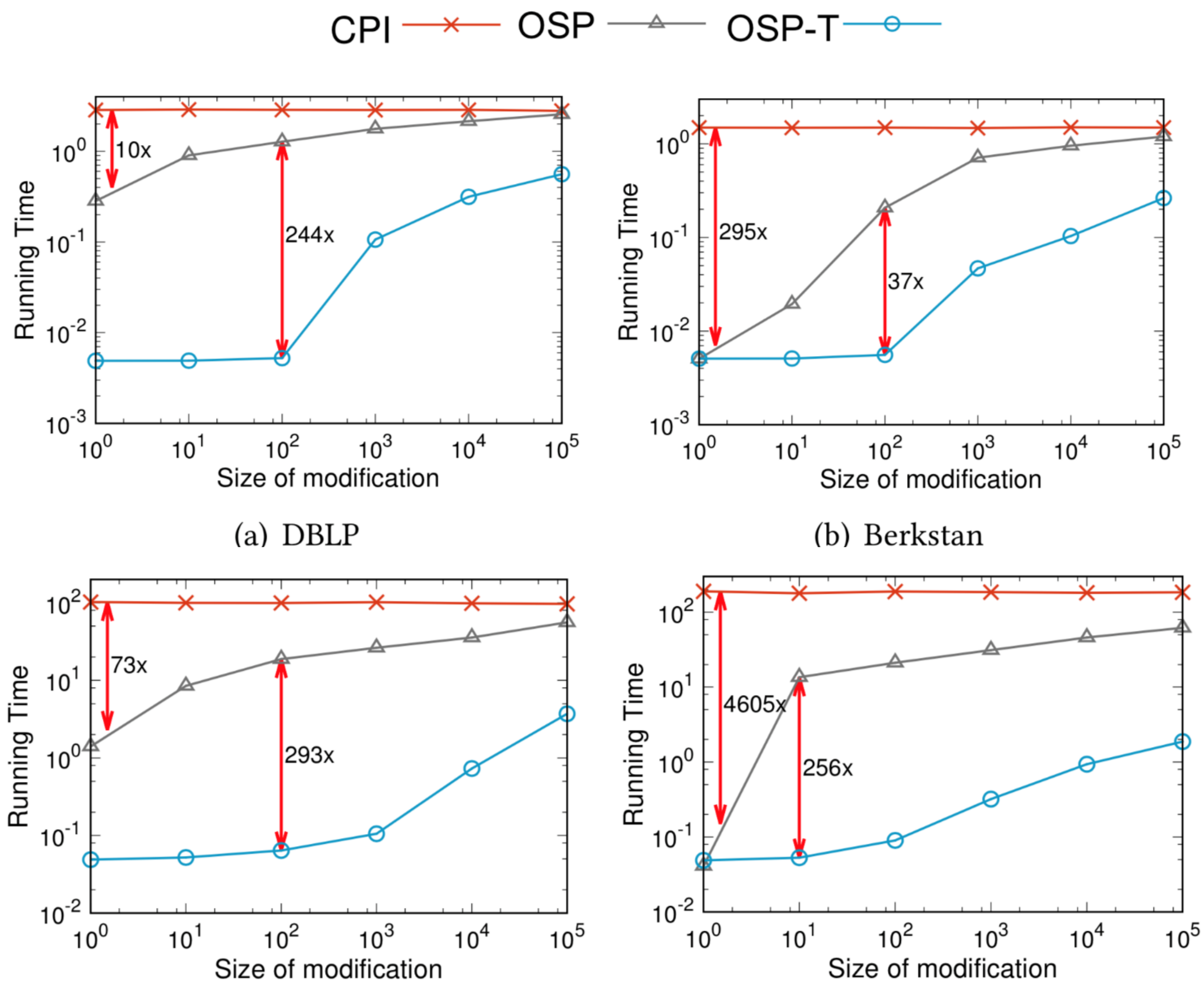
Experimental Questions

- Q1. (Performance of OSP)
  - How much does OSP improve performance for dynamic RWR computation from baseline static method CPI?
- Q2. (Performance of OSP-T)
  - How much does OSP-T enhance computation efficiency, accuracy compared with its competitors?

Q1. Performance of OSP

- How much does OSP improve performance for dynamic RWR computation from baseline static method CPI?
- Running time for tracking RWR exactly on a dynamic graph G varying the size of ΔG
  - Initial graph G with all its edges
  - Modify G by deleting edges.
    - 1 edges to 10^5 edges

Q1. Performance of OSP



(a) DBLP (b) Berkstan

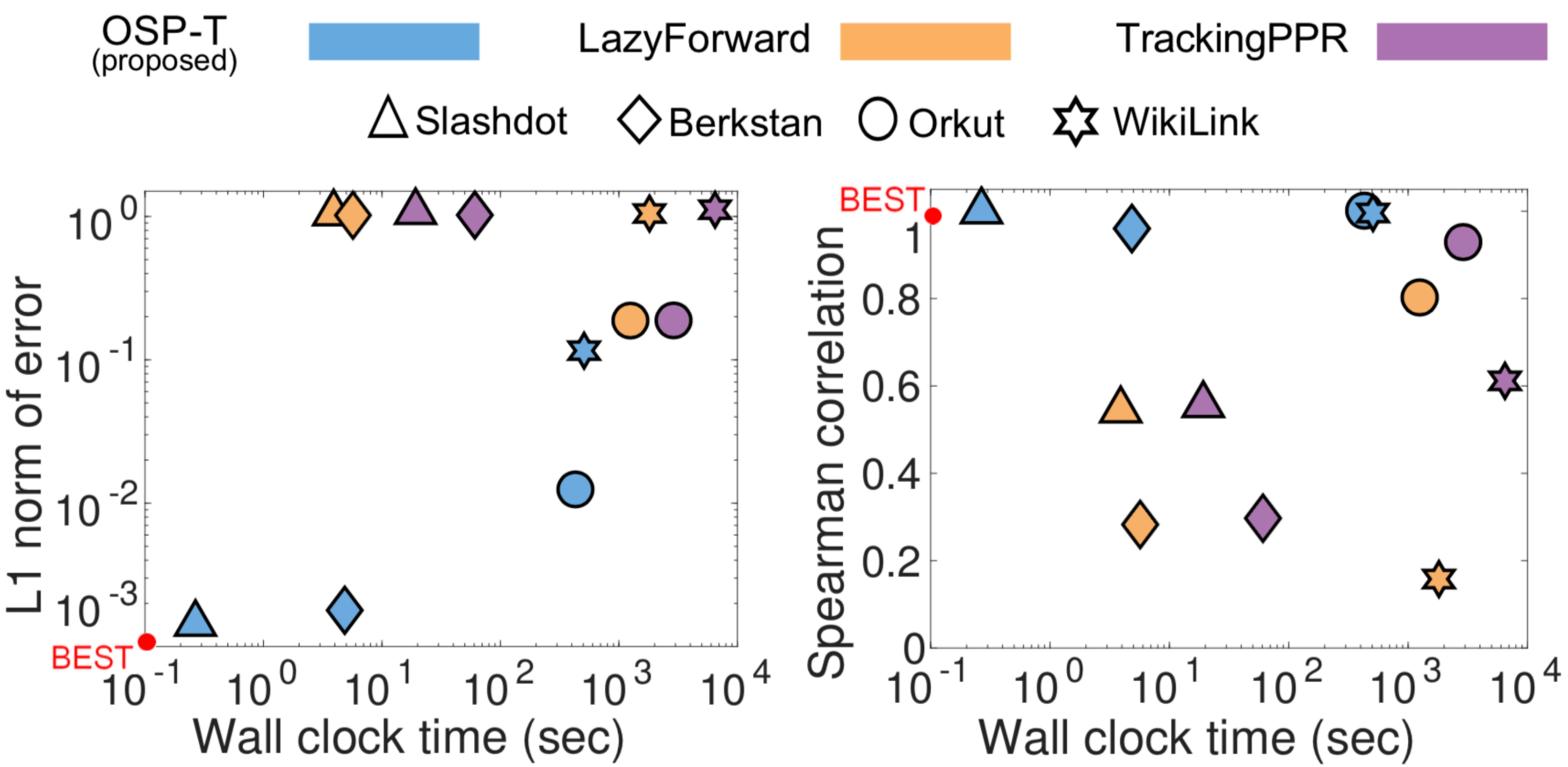
(c) LiveJournal (d) Orkut

Q2. Performance of OSP-T

- How much does OSP-T enhance computation efficiency, accuracy compared with its competitors?
- Experimental setting
  - Generate a uniformly random edge stream and divide the stream into two parts
  - Extract 10 snapshots from the second part
  - Initialize a graph with the first part of the stream
  - Update the graph for each new snapshot arrival
  - At the end of the updates, compare each algorithm.

Q2. Performance of OSP-T

- Trade-off between accuracy and running time



(a) Accuracy on L1 norm of error (b) Accuracy on Rank

Conclusion: OSP

- OSP (Offset Score Propagation)
  1. Calculate offset scores around the modified edges
  2. Propagate the offset scores across the updated graph
  3. Merge them with previous RWR scores to get updated RWR scores
- Main Results
  - Exactness of OSP
  - Error bound and time complexity of OSP-T
  - Faster and more accurate RWR computation than other methods on Dynamic graphs

http://datalab.snu.ac.kr/osp

Outline

- ☑ Random Walk with Restart (RWR)
- ☑ Fast Exact RWR
- ☑ Fast Approximate RWR
- ☐ **Conclusions**

Conclusions

- RWR for ranking in graphs: important problem with many real world applications
  - Web search, friend recommendation, product (e.g. TV program) recommendation, …
- BePI: state-of-the-art method for *exact* RWR
  - Linear algebra + Graph theory + Real World Graph Analysis
- TPA and OSP: state-of-the-art methods for *approximate* RWR

References

- U Kang and Christos Faloutsos, Beyond `Caveman Communities': Hubs and Spokes for Graph Compression and Mining, IEEE International Conference on Data Mining (ICDM) 2011, Vancouver, Canada
- Kijung Shin, Jinhong Jung, Lee Sael, and U Kang, BEAR: Block Elimination Approach for Random Walk with Restart on Large Graphs, ACM International Conference on Management of Data (SIGMOD) 2015, Melbourne, Australia.
- Jinhong Jung, Namyong Park, Lee Sael, and U Kang, BePI: Fast and Memory-Efficient Method for Billion-Scale Random Walk with Restart, ACM International Conference on Management of Data (SIGMOD) 2017, Chicago, IL, USA.
- Minji Yoon, Jinhong Jung, and U Kang, TPA: Fast, Scalable, and Accurate Method for Approximate Random Walk with Restart on Billion Scale Graphs, 34th IEEE International Conference on Data Engineering (ICDE) 2018, Paris, France.
- Minji Yoon, Woojeong Jin, and U Kang, Fast and Accurate Random Walk with Restart on Dynamic Graphs with Guarantees, The Web Conference (WWW) 2018, Lyon, France.

Thank you !

http://datalab.snu.ac.kr