BePI: Fast and Memory-Efficient Method for Billion-Scale Random Walk with Restart




Jinhong Jung — SEOUL NATIONAL UNIVERSITY

Namyong Park — SEOUL NATIONAL UNIVERSITY

Lee Sael — SUNY Korea, The State University of New York

U Kang — SEOUL NATIONAL UNIVERSITY

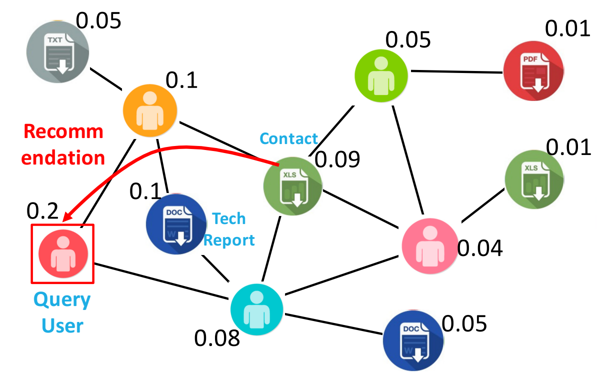
Outline

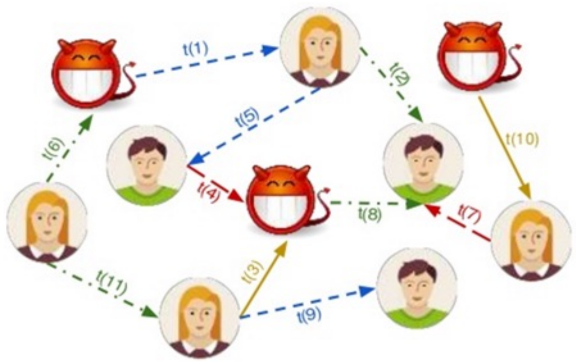
1. **Introduction**
2. **Proposed Method**
3. **Experiment**
4. **Conclusion**

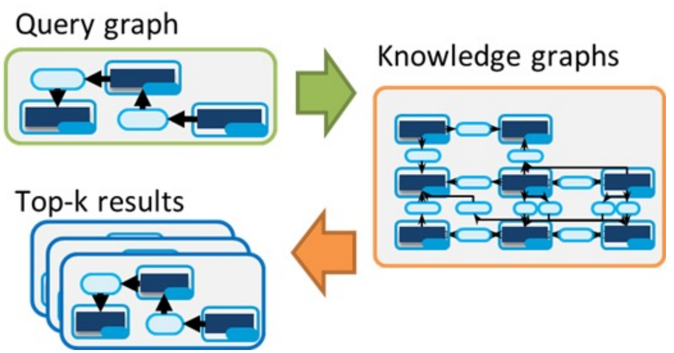
Introduction

Random Walk with Restart

- Measures the relevance between nodes in a graph
- Used in many datamining applications based on graphs







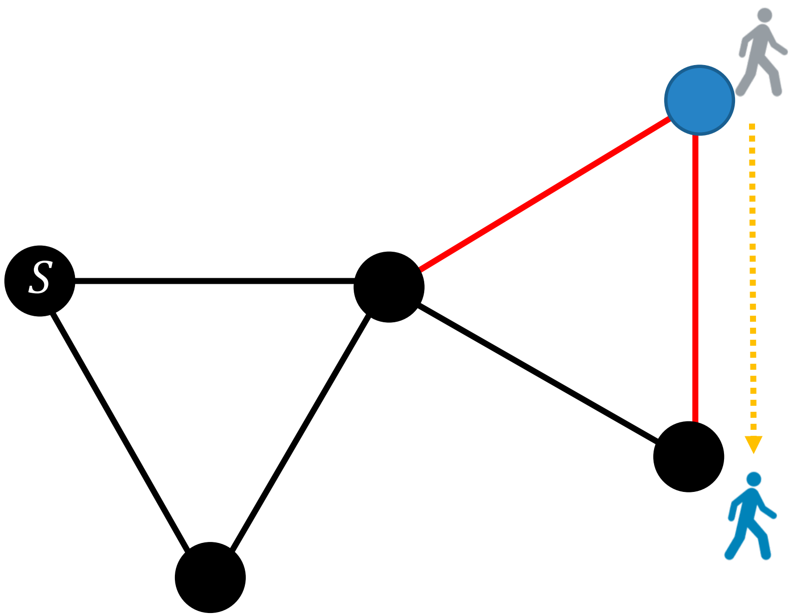
- Recommendation
  - Friends, movies, documents
- Anomaly detection
  - Spammer, trolls, frauds
- Question & Answering System
  - Subgraph matching

Random Walk with Restart is an important tool for graph analysis!

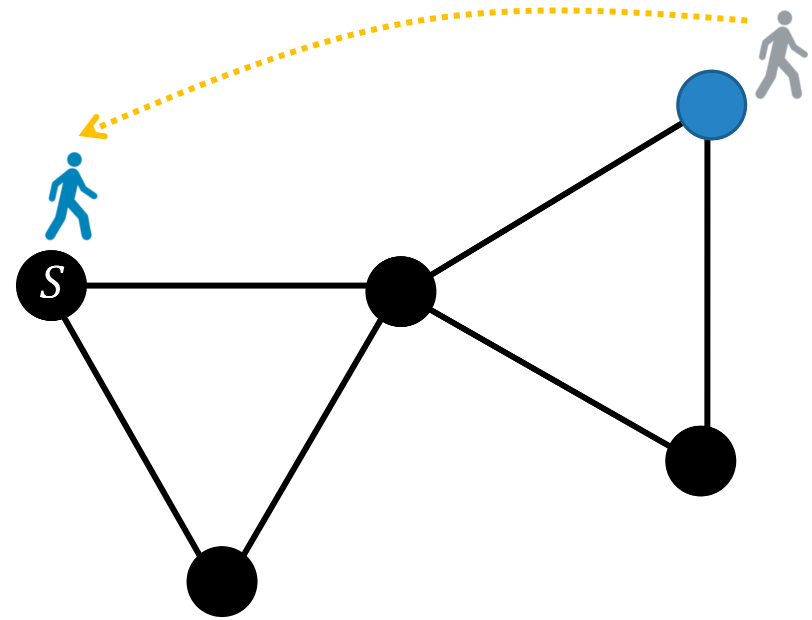
Random Walk with Restart (1)

Measures node relevance scores using random surfer

- The surfer starts at query node s on a graph
- **Random walk**: moves to one of neighbors with prob. $1 - c$
- **Restart**: jumps back to query node s with prob. c



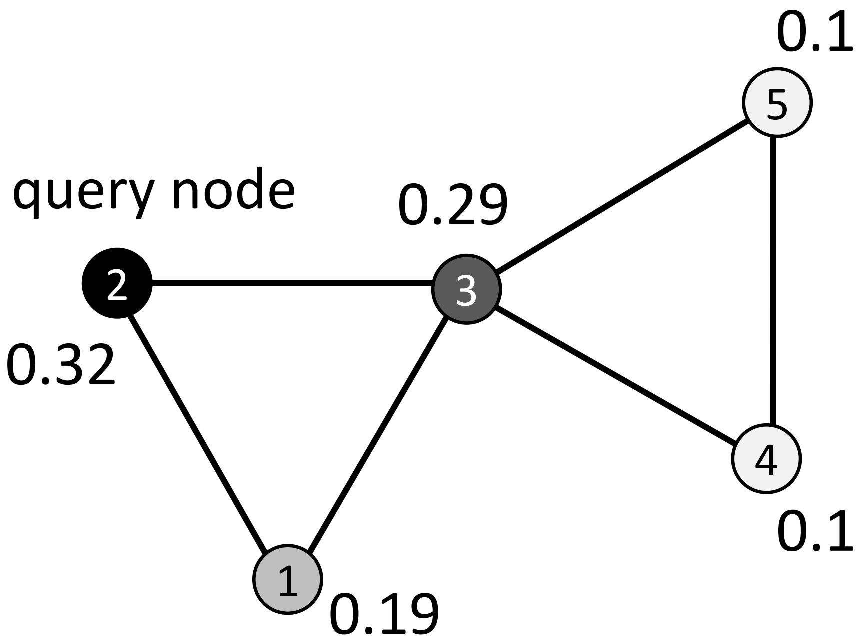
Random walk (with prob. $1 - c$)



Restart (with prob. c)

Random Walk with Restart (2)

Computes the stationary probability that the surfer stays at each node



Node	RWR Score (relevance with node 2)
1	0.19
2	0.32
3	0.29
4	0.10
5	0.10

Restarting probability $c = 0.2$

Problem Definition – RWR (1)

Given: adjacency matrix $\mathbf{A}$, query node s and restart probability c

Find: RWR score vector $\mathbf{r}_\mathrm{s}$ w.r.t. the query node s

$$\mathbf{r}_\mathrm{s} = (1-c)\widetilde{\mathbf{A}}^{\mathbf{T}}\mathbf{r}_\mathrm{s} + c\mathbf{q}_\mathrm{s}$$

Input:

- $\widetilde{\mathbf{A}} \in \mathbb{R}^{n \times n}$: row-normalized adjacency matrix
- $\mathbf{q}_\mathrm{s} \in \mathbb{R}^{n \times 1}$: query vector ($s$-th unit vector)
- $c \in \mathbb{R}$: restart probability

Output:

- $\mathbf{r}_\mathrm{s} \in \mathbb{R}^{n \times 1}$: RWR score vector with regard to query node s

Problem Definition – RWR (2)

Computing RWR is equivalent to solving a linear system

$$\mathbf{r}_{\mathrm{s}} = (1-c)\tilde{\mathbf{A}}^{\mathrm{T}}\mathbf{r}_{\mathrm{s}} + \mathrm{c}\mathbf{q}_{\mathrm{s}}$$

$$\Leftrightarrow \left(\mathbf{I} - (1-c)\tilde{\mathbf{A}}^{\mathrm{T}}\right)\mathbf{r}_{\mathrm{s}} = c\mathbf{q}_{\mathrm{s}}$$

$$\Leftrightarrow \mathbf{H}\mathbf{r}_{\mathrm{s}} = c\mathbf{q}_{\mathrm{s}}$$

- Given $\mathbf{H} \in \mathbb{R}^{n\times n}$, $\mathbf{q}_{\mathrm{s}} \in \mathbb{R}^{n}$, and c, solve the linear system to obtain $\mathbf{r}_{\mathrm{s}} \in \mathbb{R}^{n}$

Existing methods for RWR

Iterative Methods

- Iteratively update RWR scores until convergence
  - e.g., Power iteration

$$\mathbf{r}_{s}^{(i)} \leftarrow (1-c)\tilde{\mathbf{A}}^{T}\mathbf{r}_{s}^{(i-1)} + c\mathbf{q}_{s}$$

- No preprocessing phase
- Query phase (repetitive cost)
  - Given $\mathbf{q}_s$, repeat the update rule until convergence

Preprocessing Methods

- Compute RWR scores directly from precomputed data
  - e.g., Inversion

$$\mathbf{H}\mathbf{r}_{s} = c\mathbf{q}_{s} \Leftrightarrow \mathbf{r}_{s} = c\mathbf{H}^{-1}\mathbf{q}_{s}$$

- Preprocessing phase (one time)
  - Compute $\mathbf{H}^{-1}$
- Query phase (repetitive cost)
  - Given $\mathbf{q}_s$, compute $\mathbf{r}_s = c\mathbf{H}^{-1}\mathbf{q}_s$

Challenges

Q. How can we compute RWR scores quickly on very large graphs?

Iterative methods

- **Pros**: scale to very large graphs
  - Do not need preprocessed data
- **Cons**: slow RWR computation speed
  - The whole iterations need to be repeated for each query node

Preprocessing methods

- **Pros**: fast RWR computation speed
  - Directly compute the scores from precomputed results
- **Cons**: cannot handle very large graphs
  - Heavy computation cost and memory consumption due to matrix inversion

Challenge: How to devise a fast and scalable algorithm for computing RWR scores on very large graphs?

Outline

1. Introduction
2. Proposed Method
3. Experiment
4. Conclusion

Proposed Method

BePI (Best of Preprocessing and Iterative approaches)

- A fast and scalable method by taking the advantages of both preprocessing and iterative approaches

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

Proposed Method

BePI (Best of Preprocessing and Iterative approaches)

- A fast and scalable method by taking the advantages of both preprocessing and iterative approaches

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 the RWR computation speed

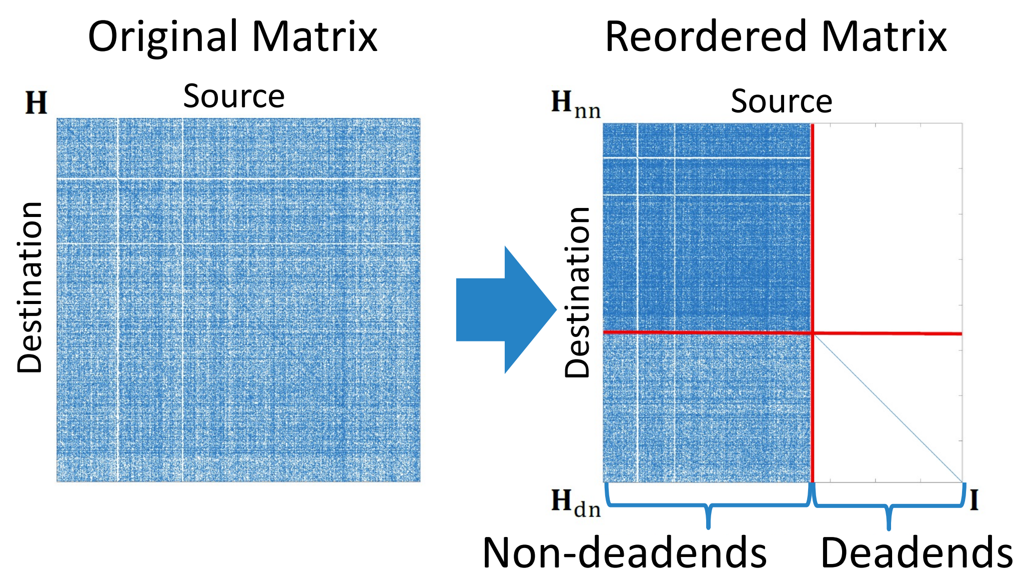
Proposed Method – Idea 1 $\mathbf{Hr} = c\mathbf{q}_s$

Exploit graph characteristics to adopt a preprocessing approach for fast query speed

- Reorder node ids to permute $\mathbf{H}$ based on ***deadend*** and ***hub-and-spoke*** structures
- Apply block elimination as a preprocessing approach

Deadend

- **Deadend** is a node having no out-going edges
- File or Image in web-document networks
- Deadends get high ids
- Non-deadends get low ids



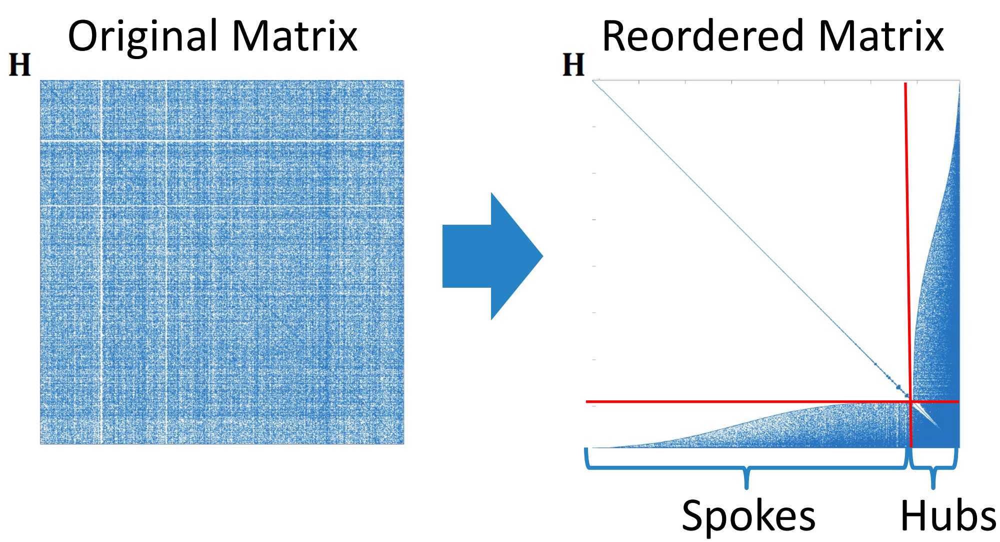
Proposed Method – Idea 1 $\mathbf{Hr} = c\mathbf{q}_s$

Reorder node ids to permute $\mathbf{H}$ based on *deadend* and *hub-and-spoke* structures

- The entries of $\mathbf{H}$ are concentrated by reordering nodes based on **hub-and-spoke** structure [Kang et al., `11]

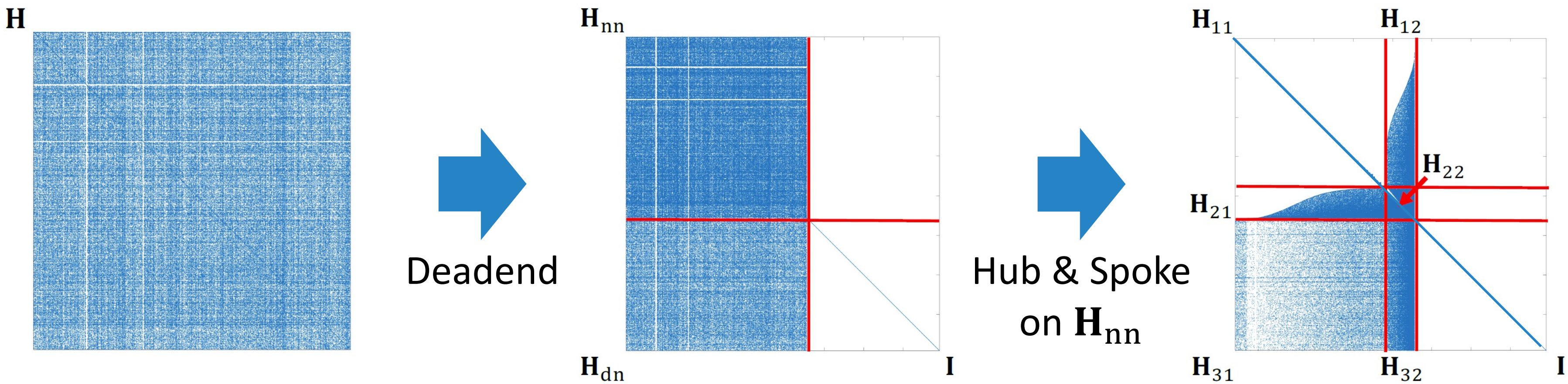
Hub-and-spoke

- **Hubs** are high degree nodes, **spokes** are low degree nodes
- Few hubs, and a majority of spokes in real-world graphs
- Hubs get high ids
- Spokes get low ids



Proposed Method – Idea 1

Combine deadend and hub & spoke reordering



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

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

Proposed Method – Idea 1

Details

Apply block elimination as a preprocessing approach

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

Block elimination
See Lemma 1

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

$\mathbf{S} = \mathbf{H}_{22} - \mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12}$, the Schur complement of $\mathbf{H}_{11}$

Precompute the blue-colored matrices to make RWR computation fast!

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
- **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

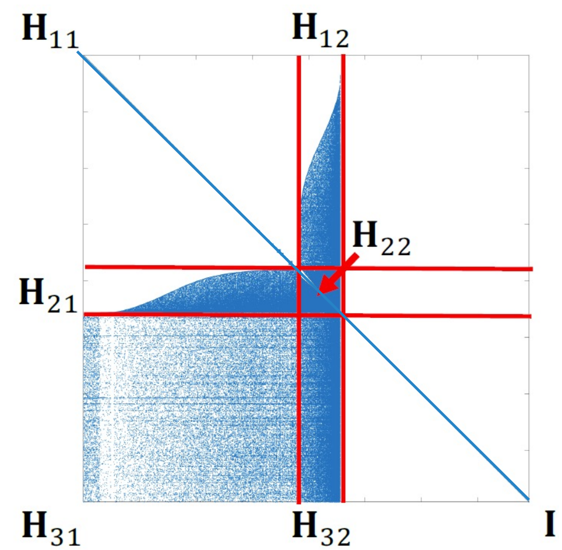
Proposed Method – Idea 2

Incorporate an iterative method into the preprocessing approach to increase the scalability

- Computing $\mathbf{H}_{11}^{-1}$ is trivial since it is block diagonal
- But, inverting $\mathbf{S}$ is impractical in very large graphs
  - $\dim(\mathbf{S}) =$ # 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 $\mathbf{S}$ using ***an iterative linear solver*** [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 solution $\boldsymbol{r}_2$ without inverting $\mathbf{S}$

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

Introducing the linear solver increases the scalability of RWR computation!

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
- **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

Proposed Method – Idea 3

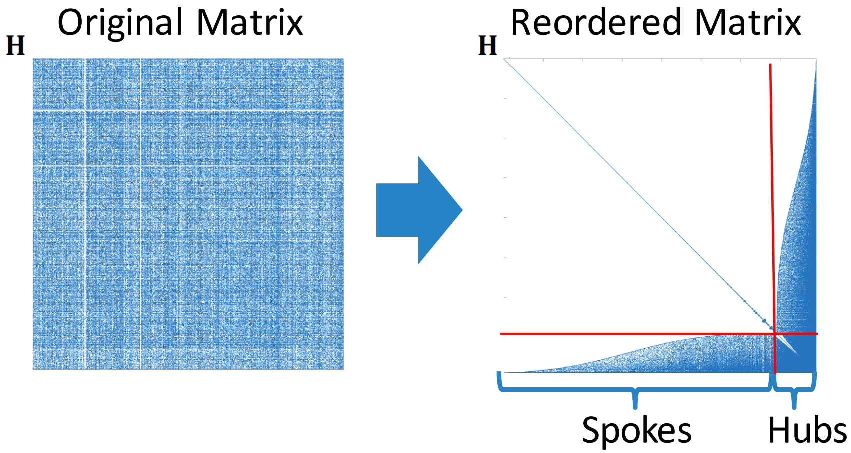
Optimize the performance of the iterative method to accelerate RWR computation speed

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

- The running time of linear solvers is $O(\mathrm{T}|\mathbf{S}|)$
  - $|\mathbf{S}|$: number of non-zeros of $\mathbf{S}$, T: number of iterations
- Optimization 1) How to decrease $|\mathbf{S}|$?
  - $\Rightarrow$ Control hub selection ratio k in hub & spoke method
- Optimization 2) How to decrease T?
  - $\Rightarrow$ Exploit a preconditioner

Propose Method
Idea 3
Optimization 1

Hub-and-spoke reordering method



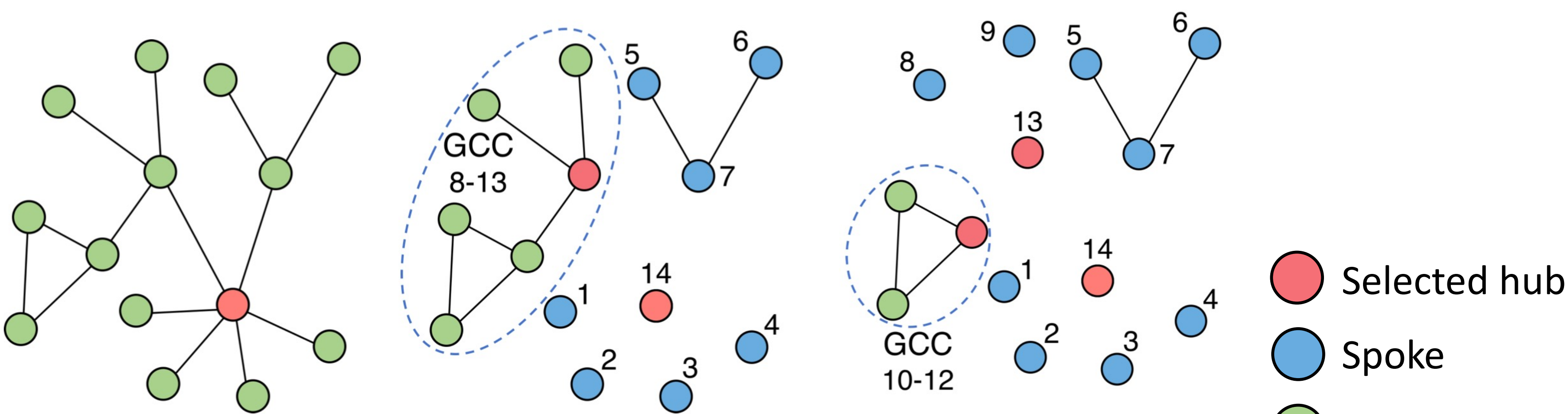
Hub-and-spoke reordering [Kang et al., `11]

For each iteration, select hubs with a hub selection ratio k

Disconnect the hubs and assign node ids for hubs & spokes

Repeat the above in the GCC (Giant Connected Comp.)

$k = 0.07$ & $(\# \; of \; nodes) \; n = 14$, select $\lceil kn \rceil = 1$ hub for each iteration



(a) Step 1 (b) Step 2 (c) Step 3

According to hub selection ratio k, # of hubs changes

Hub-and-spoke reordering method

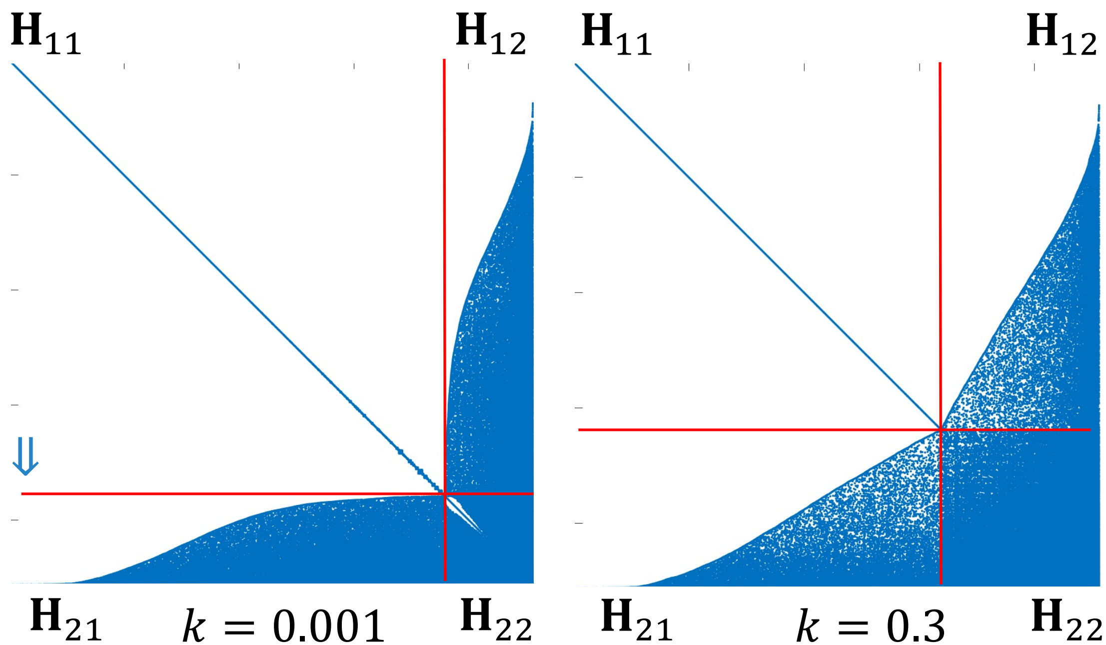
Propose Method Idea 3 Optimization 1

Proposed Method – Idea 3 $\mathbf{S}\mathbf{r}_2 = \tilde{\mathbf{q}}_2$

Optimization 1) Reduce the number of non-zeros of $\mathbf{S}$

- According to hub selection ratio k, # of hubs is different
- $\Rightarrow$ # of non-zeros of sub-matrices in $\mathbf{H}$ changes
- $\Rightarrow$ # of non-zeros of $\mathbf{S}$ changes ($\mathbf{S} = \mathbf{H}_{22} - \mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12}$)

- If k increases, then
- # of hubs increases
- $|\mathbf{H}_{22}|$ increases ↑
- $|\mathbf{H}_{12}|\&|\mathbf{H}_{21}|$ decrease ↓
- $|\mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12}|$ **decreases a lot!** ⇓
- Thus, $|\mathbf{S}|$ **decreases !!**

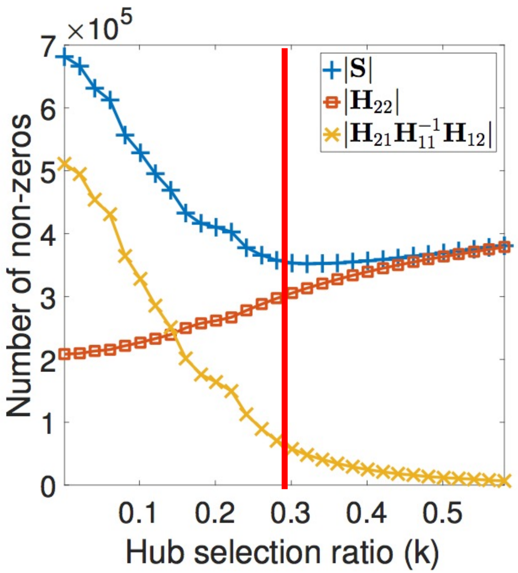


Proposed Method – Idea 3 $\mathbf{S}\mathbf{r}_2 = \tilde{\mathbf{q}}_2$

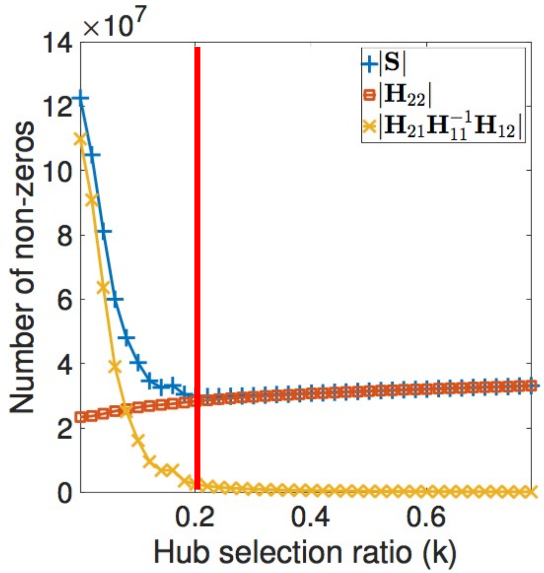
Optimization 1) Reduce the number of non-zeros of $\boldsymbol{S}$

- Pick a hub selection ratio k that minimizes $|\mathbf{S}|$

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



(a) Slashdot



(c) Flickr

- Efficiency of the iterative method on $\mathbf{S}$ is improved!
  - $O(\mathrm{T}|\mathbf{S}|)$ where T is # of iter.
- Space efficiency for $\mathbf{S}$ is also improved!
- No loss of accuracy!
- $k = 0.2 \sim 0.3$ provides a good performance in large-scale graphs

Proposed Method – Idea 3 $\mathbf{S}\mathbf{r}_2 = \tilde{\mathbf{q}}_2$

Optimization 2) Exploit the preconditioner for the linear system on $\mathbf{S}$

- Make the iterative method converge faster $\Rightarrow \mathrm{T} \downarrow$
- Exploit *incomplete LU decomposition* as preconditioners

$$\mathbf{S} \simeq \tilde{\boldsymbol{L}}\tilde{\boldsymbol{U}}$$

  - Fast decomposition and the sparsity pattern of $\mathbf{S}$ is preserved
- Implicit preconditioned system

$$\tilde{\boldsymbol{U}}^{-1}\tilde{\boldsymbol{L}}^{-1}\mathbf{S}\mathbf{r}_2 = \tilde{\boldsymbol{U}}^{-1}\tilde{\boldsymbol{L}}^{-1}\tilde{\mathbf{q}}_2$$

- Preconditioned iterative solvers [Saad `93] solve the implicit preconditioned system without matrix inversion

Outline

1. Introduction
2. Proposed Method
3. Experiment
4. Conclusion

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?

Experimental Settings

Machine: single workstation with 512GB memory

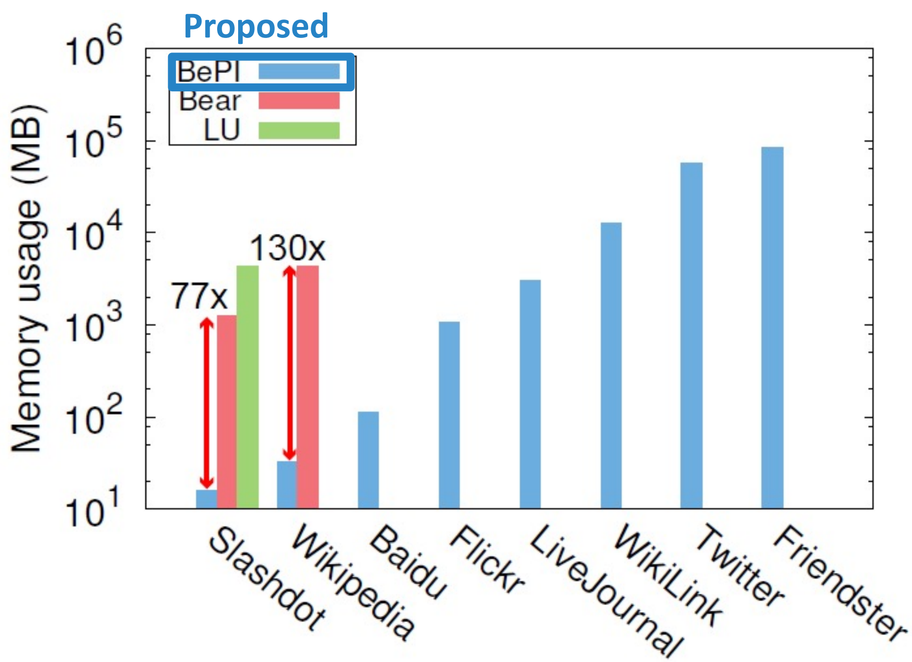
Datasets: large-scale real-world graph data

dataset	n	m
Slashdot	79,120	515,581
Wikipedia	100,312	1,627,472
Baidu	415,641	3,284,317
Flickr	2,302,925	33,140,017
LiveJournal	4,847,571	68,475,391
WikiLink	11,196,007	340,240,450
Twitter	41,652,230	1,468,365,182
Friendster	68,349,466	2,586,147,869

- n: the number of nodes
- m: the number of edges
- Various domain of graphs
  - Social, web, vote, …
- 500K ~ 2B edges in graphs

Q1. Space Efficiency

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



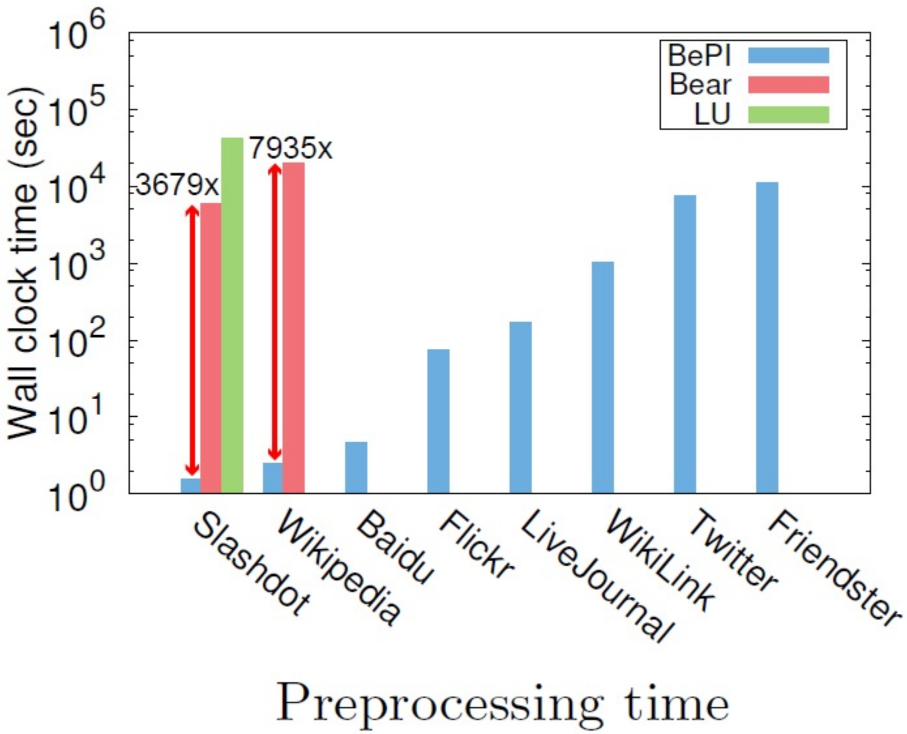
Memory space for preprocessed data

BePI requires up to $130\times$ less memory space than other preprocessing methods!

Only BePI preprocesses all datasets.

Q2. Preprocessing Time

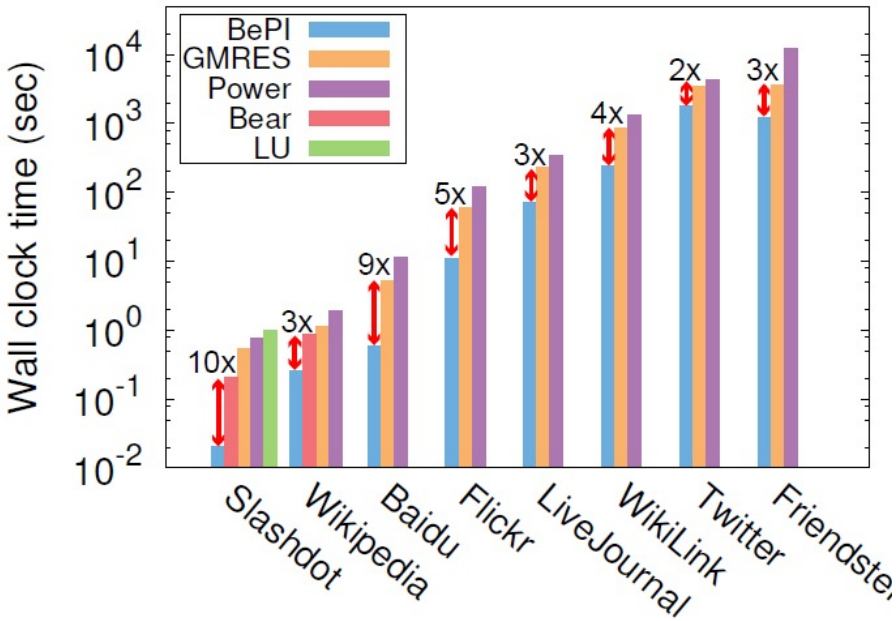
How long does the preprocessing phase of **BePI** take?



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

Q3. Query Time

How quickly does **BePI** respond to an RWR query?



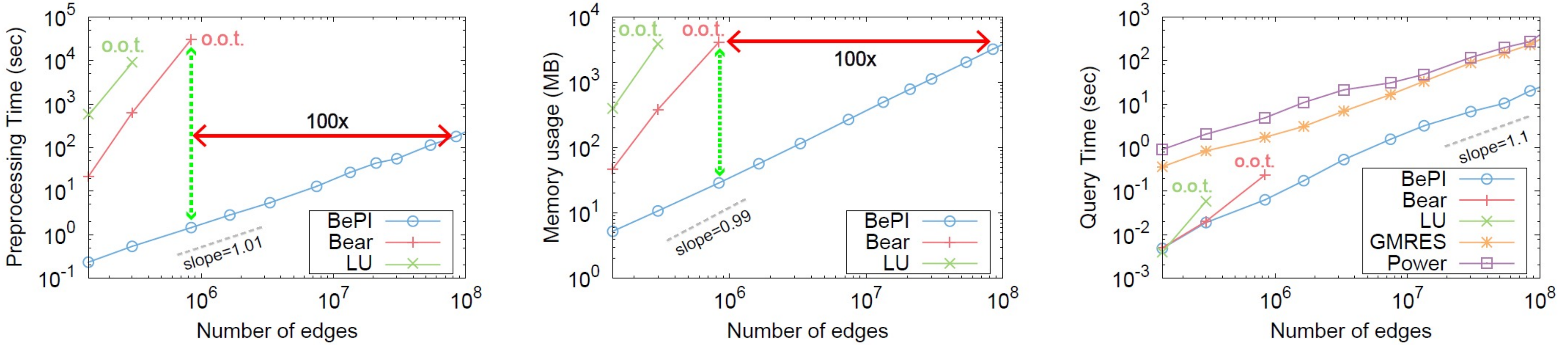
Query time

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
- Provides near linear scalability in terms of time and memory usage



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

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

Outline

1. Introduction
2. Proposed Method
3. Experiment
4. Conclusion

Conclusion

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 in large-scale graphs
- Requires **130**× less memory space & processes **100**× larger graphs than other preprocessing methods
- Computes RWR scores **9**× faster than other existing methods

Thank you!

Codes & datasets

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