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

Jinhong Jung
Seoul National University
jinhongjung@snu.ac.kr

Namyong Park
Seoul National University
namyong.park@snu.ac.kr

Lee Sael
The State University of New York (SUNY) Korea
sael@sunykorea.ac.kr

U Kang
Seoul National University
ukang@snu.ac.kr

ABSTRACT

How can we measure similarity between nodes quickly and accurately on large graphs? Random walk with restart (RWR) provides a good measure, and has been used in various data mining applications including ranking, recommendation, link prediction and community detection. However, existing methods for computing RWR do not scale to large graphs containing billions of edges; iterative methods are slow in query time, and preprocessing methods require too much memory.

In this paper, we propose BePI, a fast, memory-efficient, and scalable method for computing RWR on billion-scale graphs. BePI exploits the best properties from both preprocessing methods and iterative methods. BePI uses a block elimination approach, which is a preprocessing method, to enable fast query time. Also, BePI uses a preconditioned iterative method to decrease memory requirement. The performance of BePI is further improved by decreasing nonzeros of the matrix for the iterative method. Through extensive experiments, we show that BePI processes 100× larger graphs, and requires up to 130× less memory space than other preprocessing methods. In the query phase, BePI computes RWR scores up to 9× faster than existing methods.

Keywords

Random walk with restart; relevance score in graph

1. INTRODUCTION

Identifying node-to-node proximity in a graph is a fundamental tool for various graph mining applications, and has been recognized as an important research problem in the data mining community [2, 10, 14, 46]. Random walk with restart (RWR) provides a good relevance score, taking into account the global network structure [20] and the multi-faceted relationship between nodes [40] in a graph. RWR has been successfully utilized in many graph mining tasks including ranking [41], recommendation [28], link prediction [3], and community detection [1, 18, 45].

Existing methods for scalable computation of RWR scores can be classified into two categories: iterative approaches and preprocessing approaches. Iterative methods, such as power iteration [33], compute an RWR score by repeatedly updating it until convergence. While they require much less memory space compared to preprocessing methods, they are slow in the query phase because matrix-vector multiplications should be performed each time for a different query node. This makes iterative methods not fast enough for billion-scale graphs.

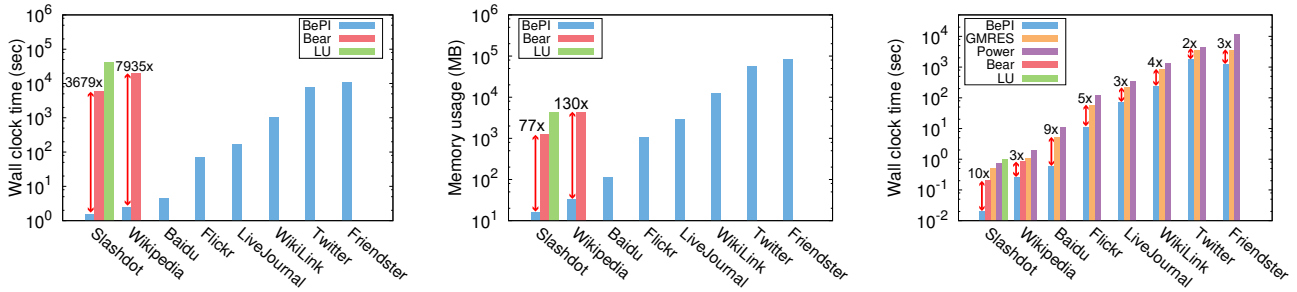
On the other hand, preprocessing methods compute RWR scores using precomputed intermediate matrices. Since preprocessed matrices need to be computed just once, and then can be reused, they are fast in the query phase, especially when they should serve many query nodes. However, existing preprocessing approaches have high memory require-

Table 1: Table of symbols.

Symbol	Definition
G	input graph
n	number of nodes in G
m	number of edges in G
n_1	number of spokes in G
n_2	number of hubs in G
n_3	number of deadends in G
n_{1i}	number of nodes in the ith diagonal block of $\mathbf{H_{11}}$
b	number of diagonal blocks in $\mathbf{H_{11}}$
s	seed node (=query node)
c	restart probability
k	hub selection ratio in the hub-and-spoke reordering method [23]
ϵ	error tolerance
$\mathbf{A}$	$(n \times n)$ adjacency matrix of G
$\mathbf{A}_{\text{nn}}$	adjacency matrix containing edges from non-deadend nodes to non-deadend nodes
$\mathbf{A}_{\text{nd}}$	adjacency matrix containing edges from non-deadend nodes to deadend nodes
$\tilde{\mathbf{A}}$	$(n \times n)$ row-normalized adjacency matrix of G
$\mathbf{H}$	$(n \times n)$ $\mathbf{H} = \mathbf{I} - (1-c)\tilde{\mathbf{A}}^{\mathbf{T}}$
$\mathbf{H}_{ij}$	$(n_i \times n_j)$ (i,j)-th partition of $\mathbf{H}$
$\mathbf{S}$	$(n_2 \times n_2)$ Schur complement of $\mathbf{H_{11}}$
$\mathbf{L}_1, \mathbf{U}_1$	$(n_1 \times n_1)$ LU factors of $\mathbf{H_{11}}$
$\tilde{\mathbf{L}}_2, \tilde{\mathbf{U}}_2$	$(n_2 \times n_2)$ incomplete LU factors of $\mathbf{S}$
$\mathbf{q}, \mathbf{q}_i$	$(n \times 1)$ starting vector, $(n_i \times 1)$ i-th partition of $\mathbf{q}$
$\mathbf{r}, \mathbf{r}_i$	$(n \times 1)$ relevance vector, $(n_i \times 1)$ i-th partition of $\mathbf{r}$
$\lvert\mathbf{A}\rvert$	number of non-zero entries of a matrix $\mathbf{A}$

SIGMOD'17, May 14-19, 2017, Chicago, Illinois, USA

DOI: http://dx.doi.org/10.1145/3035918.3035950



(a) Preprocessing time (b) Memory space for preprocessed data (c) Query time

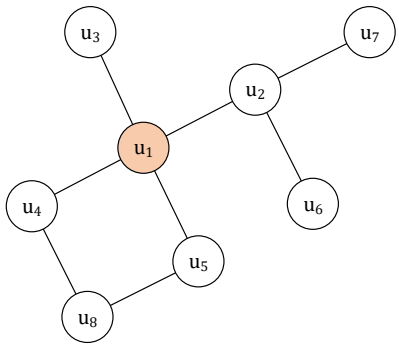
Figure 1: Performance of BePI: (a) and (b) compare the preprocessing time, and the memory space for preprocessed data, respectively, among preprocessing methods; (c) compares the query time among all tested methods. Bars are omitted if the corresponding experiments run out of memory or time (more than 24 hours). (a) In the preprocessing phase, BePI is the fastest and the most scalable among all preprocessing methods. Only BePI successfully preprocesses billion-scale graphs such as Twitter and Friendster datasets. (b) BePI uses the least amount of space for preprocessed data across all the datasets. Only BePI preprocesses all the datasets, whereas Bear and LU decomposition fail except for the two smallest ones. (c) In the query phase, BePI computes RWR scores faster than other competitors over all datasets. Details on these experiments are presented in Section 4.

ments in common, due to the space for the preprocessed matrices, which makes it difficult to scale them up to billion-scale real-world graphs such as Twitter or Friendster datasets (see Table 2).

In this paper, we propose BePI (BEst of Preprocessing and Iteartive approaches for RWR), a fast, memory-efficient, and scalable method for computing RWR on billion-scale graphs. BePI addresses the challenges faced by previous approaches by combining the best of both preprocessing and iterative methods. BePI uses a block elimination approach, which is a preprocessing method, to achieve fast query time. BePI incorporates an iterative method within the block elimination to decrease memory requirements by avoiding expensive matrix inversion. The performance of BePI is further enhanced via matrix sparsification and preconditioning. Through extensive experiments with various real-world graphs, we demonstrate the superiority of BePI over existing methods as shown in Figure 1. The main contributions of this paper are the followings:

- **Algorithm.** We propose BePI, a fast, memory-efficient, and scalable algorithm for computing RWR on billion-scale graphs. BePI efficiently computes RWR scores based on precomputed matrices by exploiting an iterative method, reducing the number of non-zeros of a matrix, and applying a preconditioner.
- **Analysis.** We give theoretical guarantees of the accuracy of BePI. We also analyze the time and the space complexities of BePI, and show that the complexities are smaller than those of the state-of-the art method.
- **Experiment.** BePI processes 100× larger graphs and requires 130× less memory space than existing preprocessing methods. Moreover, BePI provides near linear scalability in terms of preprocessing and query cost. BePI computes RWR scores up to 9× faster than existing iterative methods.

The code of our method and datasets used in the paper are available at http://datalab.snu.ac.kr/bepi. The rest of the paper is organized as follows. In Section 2, we give preliminaries on the definition and algorithms of RWR. We describe our proposed method BePI in Section 3. After presenting our experimental results in Section 4, we provide a review on related works in Section 5. We conclude in Section 6.



node	score	rank
u_1	0.287	1
u_2	0.171	2
u_3	0.068	6
u_4	0.124	3
u_5	0.124	4
u_6	0.054	7
u_7	0.054	8
u_8	0.118	5

Figure 2: Example of RWR. In the example, the query node is u_1 and RWR scores w.r.t. u_1 are presented in the table. The RWR scores are utilized for personalized ranking or link recommendation for u_1.

2. PRELIMINARIES

In this section, we present the preliminaries on random walk with restart (RWR) and two different approaches, iterative methods and preprocessing methods. Symbols used in the paper are summarized in Table 1.

2.1 Random Walk with Restart

Given a graph G, a query node s, and a restart probability c, random walk with restart (RWR) [41] measures proximity scores $\mathbf{r}$ between the query node s and each node on the graph. RWR leverages the proximities by allowing a random surfer to move around the graph. Suppose that a random surfer starts at node s, and takes one of the following actions at each node:

- **Random Walk.** The surfer randomly moves to one of the neighbors from the current node with probability $1 - c$.
- **Restart.** The surfer goes back to the query node s with probability c.

The proximity or the RWR score between a node u and the query node s is the steady-state probability that the surfer is at node u after performing RWR starting from node s. If the proximity is high, we consider that nodes u and s are highly related, e.g., they are close friends in a social network. Thus, RWR provides relevance scores between the query node s

and each node, and it is utilized as a personalized ranking for the query node s [41].

For example, suppose u_1 is the query node as shown in Figure 2. The RWR scores w.r.t. u_1 are presented in the table of the figure, and the scores are used for the personalized ranking for u_1. Also, we are able to recommend to friends for u_1 based on the scores. The RWR score of u_8 is higher than that of u_6 because u_8 is highly correlated to u_1 by the connections with u_4 and u_5. Thus, u_8 will be recommended to u_1 rather than u_6 would based on the RWR scores.

RWR scores for all nodes w.r.t. the query node s are represented as an RWR score vector $\mathbf{r}$ which is defined by the following recursive equation [33, 41] :

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

where $\tilde{\mathbf{A}}$ is the row-normalized adjacency matrix of the graph G, and $\mathbf{q}$ is the starting vector whose entry that corresponds to the node s is set to 1, and others to 0. From Equation (1), we obtain the following linear equation:

$$(\mathbf{I} - (1-c)\tilde{\mathbf{A}}^{\mathbf{T}})\mathbf{r} = c\mathbf{q} \Leftrightarrow \mathbf{H}\mathbf{r} = c\mathbf{q} \tag{2}$$

where $\mathbf{H} = \mathbf{I} - (1-c)\tilde{\mathbf{A}}^{\mathbf{T}}$. Note that $\mathbf{q}$ is an RWR query, and $\mathbf{r}$ is the result corresponding to the query. $\mathbf{q}$ is determined by the query node s, and $\mathbf{r}$ is distinct for each RWR query. RWR is a special case of Personalized PageRank (PPR) which sets multiple seed nodes in the starting vector $\mathbf{q}$ while RWR sets only one seed node [33].

2.2 Iterative Methods for RWR

Iterative methods update the RWR score vector $\mathbf{r}$ iteratively. The most well-known method is the power iteration method [33] which repeatedly updates $\mathbf{r}$ as follows:

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

where $\mathbf{r}^{(i)}$ denotes the vector $\mathbf{r}$ at the i-th iteration. The repetition continues until $\mathbf{r}$ has converged (i.e., $\|\mathbf{r}^{(i)} - \mathbf{r}^{(i-1)}\|_2 \leq \epsilon$). The vector $\mathbf{r}$ is guaranteed to converge to a unique solution if $0 < c < 1$ [27]. Krylov subspace methods [36] are also used to compute the solution of a linear system shown in Equation (2). These methods iterate a procedure to search the solution in the Krylov subspace. Since the matrix $\mathbf{H}$ is non-singular and non-symmetric [27], any Krylov subspace method, such as GMRES [37], which handles a non-symmetric matrix, can be applied to Equation (2). While these iterative methods do not require preprocessing, they have expensive query cost especially when there are lots of queries, since the whole iterations need to be repeated for each query.

2.3 Preprocessing Methods for RWR

Many real-world applications require RWR scores of any pair of nodes, e.g., scores between two arbitrary users in social networks. Hence, quickly computing RWR queries is important and useful for real-world applications. Preprocessing methods directly calculate $\mathbf{r}$ based on precomputed results to accelerate the query speed. One naive approach is to compute $\mathbf{H}^{-1}$ as follows:

$$\mathbf{r} = c\mathbf{H}^{-1}\mathbf{q}.$$

Once $\mathbf{H}^{-1}$ is obtained in the preprocessing phase, $\mathbf{r}$ can be computed efficiently in the query phase. However, obtaining $\mathbf{H}^{-1}$ is impractical for large graphs because inverting the matrix is very time-consuming and $\mathbf{H}^{-1}$ is too dense to fit into memory. Several preprocessing methods were proposed to alleviate the problem about $\mathbf{H}^{-1}$. Fujiwara et al. [14] proposed to use matrix factorizations such as QR or LU factorization to replace $\mathbf{H}^{-1}$ (e.g., $\mathbf{H}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}$ if $\mathbf{H}$ is LU factorized). They reordered $\mathbf{H}$ based on nodes' degrees and community structures to make the inverses of factors sparse. Shin et al. [38] developed a block elimination approach called Bear which exploits a node reordering technique [23] to concentrate non-zeros of $\mathbf{H}$, and uses block elimination [9] to compute the solution. While these preprocessing methods compute RWR queries quickly based on precomputed results, they have scalability issues for processing very large graphs because they require heavy computational cost and large memory space caused by matrix inversion inside the preprocessing phase. That is, matrix inversion requires $O(n^3)$ time and $O(n^2)$ space where n is the dimension of a matrix to be processed. Under those complexities, if n is greater than a million, it is infeasible to complete a preprocessing phase based on matrix inversion and store preprocessed data.

3. PROPOSED METHOD

In this section, we describe our proposed method BePI for fast, memory-efficient, and scalable RWR computation.

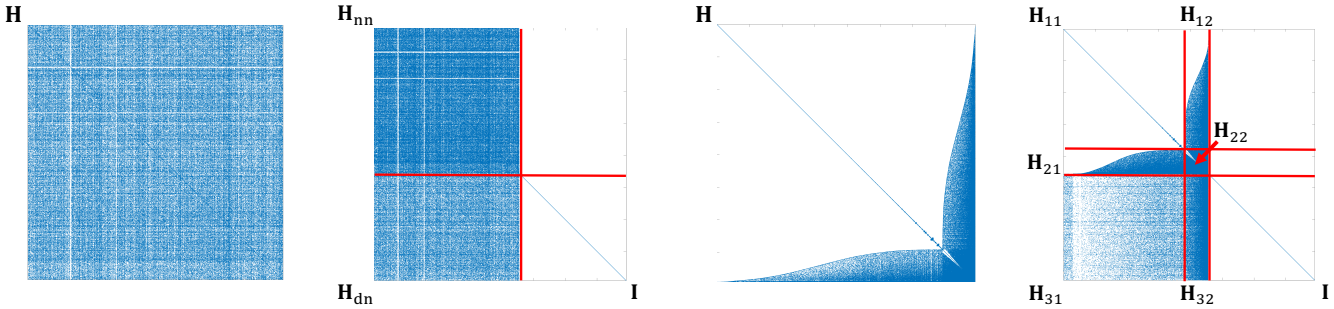
3.1 Overview

Preprocessing methods process relatively large graphs, and compute RWR scores quickly. However, they cannot handle very large graphs due to their high memory requirement. On the other hand, iterative methods scale to very large graphs, but show slow query speed. In this paper, our purpose is to devise a fast and scalable algorithm by taking the advantages of both preprocessing methods and iterative methods.

We present a basic version of our method BePI-B, and two optimized versions: BePI-S and BePI. BePI-B reorders nodes based on the characteristics of real-world graphs and adopts block elimination as a preprocessing method to reduce query time. Moreover, BePI-B exploits an iterative method within the block elimination approach to process very large graphs. BePI-S further improves the performance of the iterative method by sparsifying a matrix in terms of running time and memory requirement. On top of that, BePI accelerates the query speed by applying a preconditioner to the iterative method. The main ideas of our proposed method are summarized as follows:

- **BePI-B: exploiting graph characteristics** to reorder nodes and apply block elimination (Section 3.2), and **incorporating an iterative method into block elimination** to increase the scalability of RWR computation (Section 3.3).
- **BePI-S: sparsifying the Schur complement** to improve the performance of the iterative method (Section 3.4).
- **BePI: preconditioning a linear system** to make the iterative method converge faster (Section 3.5).

BePI comprises two phases: the preprocessing phase and the query phase. In the preprocessing phase (Algorithm 3), BePI precomputes several matrices which are required by the query phase. In the query phase (Algorithm 4), BePI computes RWR scores for each query by exploiting the precomputed matrices. Note that the preprocessing phase is



(a) Original matrix $\mathbf{H}$ (b) Deadend reordering (c) Hub-and-spoke reordering (d) Deadend and hub-and-spoke reordering

Figure 3: The results of node reordering on the Slashdot dataset. (a) is the original matrix $\mathbf{H}$ before node reordering. (b) and (c) are $\mathbf{H}$ reordered by deadend reordering and hub-and-spoke reordering, respectively. (d) is $\mathbf{H}$ reordered by the hub-and-spoke reordering method on top of the result of the deadend reordering method. BEPI computes RWR scores on the reordered matrix $\mathbf{H}$ in (d). $\mathbf{H}_{11}$ in (d) is a block diagonal matrix.

run once, and the query phase is run for each seed node. To exploit sparsity of graphs, we save all matrices in a sparse matrix format such as compressed column storage [12] which stores only non-zero entries and their locations.

3.2 BePI-B: Exploiting Graph Characteristics for Node Reordering and Block Elimination

BEPI-B first reorders $\mathbf{H} = \mathbf{I} - (1-c)\tilde{\mathbf{A}}^{\mathbf{T}}$ based upon real-world graph characteristics, and applies block elimination for efficient RWR computation. Previous works [26, 38, 16] have shown that node reordering methods reduce computational cost of operations based on adjacency matrices of real-world graphs. For further improvement, we propose to mix node reordering strategies based on two graph characteristics: 1) deadends, and 2) hub-and-spoke structure. After reordering nodes, we apply block elimination as a preprocessing method to reduce query cost.

3.2.1 *Node Reordering Based on Deadends and Hub-and-Spoke Structure*

Deadends. Deadends are nodes having no out-going edges. Many deadends are produced from various sources such as a page containing only a file or an image in real-world graphs (see Table 2). Deadends have been used to improve the performance of graph operations [26]. In this paper, we reorder nodes based on deadends for efficient RWR computation. Suppose that an adjacency matrix $\mathbf{A}$ is reordered so that non-deadends and deadends are separated as follows:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{\mathrm{nn}} & \mathbf{A}_{\mathrm{nd}} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

where $\mathbf{A}_{\mathrm{nn}}$ is a submatrix containing edges from non-deadend nodes to non-deadend nodes, and $\mathbf{A}_{\mathrm{nd}}$ is a submatrix containing edges from non-deadend nodes to deadend nodes. Then, Equation (2) is represented as follows:

$$\mathbf{H}\mathbf{r} = c\mathbf{q} \Leftrightarrow \begin{bmatrix} \mathbf{H}_{\mathrm{nn}} & \mathbf{0} \\ \mathbf{H}_{\mathrm{dn}} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{r}_{\mathrm{n}} \\ \mathbf{r}_{\mathrm{d}} \end{bmatrix} = c \begin{bmatrix} \mathbf{q}_{\mathrm{n}} \\ \mathbf{q}_{\mathrm{d}} \end{bmatrix}$$

where $\mathbf{H}_{\mathrm{nn}} = \mathbf{I} - (1-c)\tilde{\mathbf{A}}^{\mathbf{T}}_{\mathrm{nn}}$ and $\mathbf{H}_{\mathrm{dn}} = -(1-c)\tilde{\mathbf{A}}^{\mathbf{T}}_{\mathrm{nd}}$. Figure 3(b) presents the example of $\mathbf{H}$ reordered by the deadend reordering approach. The partitioned solutions $\mathbf{r}_{\mathrm{n}}$ and $\mathbf{r}_{\mathrm{d}}$ are obtained from the following equations:

$$\mathbf{H}_{\mathrm{nn}}\mathbf{r}_{\mathrm{n}} = c\mathbf{q}_{\mathrm{n}} \tag{3}$$

$$\mathbf{r}_{\mathrm{d}} = c\mathbf{q}_{\mathrm{d}} - \mathbf{H}_{\mathrm{dn}}\mathbf{r}_{\mathrm{n}} \tag{4}$$

Note that the dimension and the number of non-zeros of $\mathbf{H}_{\mathrm{nn}}$ are smaller than those of $\mathbf{H}$. The partitioned solution $\mathbf{r}_{\mathrm{d}}$ is easily computed if we have $\mathbf{r}_{\mathrm{n}}$. Hence, the deadend reordering approach enables to obtain RWR scores by solving the linear system in Equation (3) which is smaller than the original one in Equation (2). One naive method for computing Equation (3) is to invert $\mathbf{H}_{\mathrm{nn}}$, i.e., $\mathbf{r}_{\mathrm{n}} = \mathbf{H}_{\mathrm{nn}}^{-1}\mathbf{q}_{\mathrm{n}}$. However, obtaining $\mathbf{H}_{nn}^{-1}$ is infeasible in very large graphs because its dimension is still too large to invert. To efficiently solve the linear system in Equation (3), we introduce another reordering technique based on the hub-and-spoke structure on top of the deadend reordering approach.

Hub-and-spoke structure. Most real-world graphs have the hub-and-spoke structure meaning they follow power-law degree distribution with few *hubs* (very high degree nodes) and majority of *spokes* (low degree nodes) [13]. The structure is exploited to concentrate entries of an adjacency matrix by reordering nodes as shown in Figure 3(c). The reordered matrix based on the hub-and-spoke structure has improved the performance of operations on graphs [38]. We use the hub-and-spoke structure to efficiently solve Equation (3). Any reordering method based on the hub-and-spoke structure can be utilized for the purpose; in this paper, we use SlashBurn [23] because it shows the best performance in concentrating entries of an adjacency matrix (more details in Appendix A).

We reorder nodes of the submatrix $\mathbf{A}_{\mathrm{nn}}$ using the hub-and-spoke reordering method so that the reordered matrix contains a large but easy-to-invert submatrix such as a block diagonal one as shown in Figure 3(c). After reordered by the deadend approach and the hub-and-spoke reordering method, $\mathbf{H}$ is partitioned as follows:

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{\mathrm{nn}} & \mathbf{0} \\ \mathbf{H}_{\mathrm{dn}} & \mathbf{I} \end{bmatrix} \Leftrightarrow \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} & \mathbf{0} \\ \mathbf{H}_{21} & \mathbf{H}_{22} & \mathbf{0} \\ \mathbf{H}_{31} & \mathbf{H}_{32} & \mathbf{I} \end{bmatrix} \tag{5}$$

where $\mathbf{H}_{\mathrm{nn}}$ is partitioned to $\begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} \\ \mathbf{H}_{21} & \mathbf{H}_{22} \end{bmatrix}$, and $\mathbf{H}_{\mathrm{dn}}$ is partitioned to $\begin{bmatrix} \mathbf{H}_{31} & \mathbf{H}_{32} \end{bmatrix}$. Figure 3(d) illustrates the example of the reordered matrix $\mathbf{H}$ in Equation (5). Let n_1 be the number of spokes, n_2 be the number of hubs (see Appendix A), and n_3 be the number of deadends. n_1 and n_2 are determined by the hub-and-spoke reordering method, and n_3 is computed by the deadend reordering method. $\mathbf{H}_{11}$ is an $n_1 \times n_1$ matrix, and $\mathbf{H}_{22}$ is an $n_2 \times n_2$ matrix. $\mathbf{H}_{31}$ is an $n_3 \times n_1$ matrix, and $\mathbf{H}_{32}$ is an $n_3 \times n_2$ matrix. Note that

$\mathbf{H}_{11}$ is block diagonal as shown in Figure 3(d) since $\mathbf{H}_{\text{nn}}$ has the same sparsity pattern as that of the reordered matrix $\mathbf{A}_{\text{nn}}^{\mathbf{T}}$ except for the diagonal entries, and the upper left part of the reordered matrix $\mathbf{A}_{\text{nn}}^{\mathbf{T}}$ is a block diagonal matrix.

3.2.2 Block Elimination

By plugging Equation (5) into $\mathbf{Hr} = c\mathbf{q}$, the linear system is represented as follows:

$$\mathbf{Hr} = c\mathbf{q} \Leftrightarrow \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} & \mathbf{0} \\ \mathbf{H}_{21} & \mathbf{H}_{22} & \mathbf{0} \\ \mathbf{H}_{31} & \mathbf{H}_{32} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \end{bmatrix} = c \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \mathbf{q}_3 \end{bmatrix}. \quad (6)$$

The partitioned linear system in Equation (6) is solved by applying block elimination [9]. That is, the RWR solution vector $\mathbf{r}$ is obtained from the following lemma:

LEMMA 1 (BLOCK ELIMINATION [9, 38]). *The linear system in Equation* (6) *is solved by block elimination, and the solution* $\mathbf{r}$ *is represented as follows:*

$$\mathbf{r} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{11}^{-1}(c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2) \\ \mathbf{S}^{-1}(c\mathbf{q}_2 - \mathbf{H}_{21}(\mathbf{H}_{11}^{-1}(c\mathbf{q}_1))) \\ c\mathbf{q}_3 - \mathbf{H}_{31}\mathbf{r}_1 - \mathbf{H}_{32}\mathbf{r}_2 \end{bmatrix} \quad (7)$$

where $\mathbf{S} = \mathbf{H}_{22} - \mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12}$ *is the Schur complement of* $\mathbf{H}_{11}$. *Note that the dimension of* $\mathbf{S}$ *is* $n_2 \times n_2$ *where* n_2 *is the number of hubs.*

PROOF. *See Appendix D.* □

If all matrices in Equation (7) are precomputed, the RWR score vector $\mathbf{r}$ is efficiently calculated, i.e., only matrix vector multiplications are required for the query computation.

3.3 BePI-B: Incorporating an Iterative Method into Block Elimination

BEPI-B incorporates an iterative method within the block elimination to compute RWR on very large graphs. Based on the block elimination approach, the RWR vector $\mathbf{r} = [\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3]^{\mathbf{T}}$ is obtained by solving the following linear systems:

$$\mathbf{H}_{11}\mathbf{r}_1 = c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2 \quad (8)$$

$$\mathbf{S}\mathbf{r}_2 = c\mathbf{q}_2 - \mathbf{H}_{21}(\mathbf{H}_{11}^{-1}(c\mathbf{q}_1)) \quad (9)$$

$$\mathbf{r}_3 = c\mathbf{q}_3 - \mathbf{H}_{31}\mathbf{r}_1 - \mathbf{H}_{32}\mathbf{r}_2 \quad (10)$$

where $\mathbf{S}$ is the Schur complement of $\mathbf{H}_{11}$. Note that those equations are derived from Equation (7). $\mathbf{r}_3$ is easily obtained from $\mathbf{r}_1$ and $\mathbf{r}_2$ based on Equation (10). $\mathbf{r}_1$ is also easily computed if we have $\mathbf{r}_2$, because $\mathbf{H}_{11}$ is block diagonal and consists of small blocks; hence, $\mathbf{H}_{11}$ is easy-to-invert (i.e., $\mathbf{r}_1 = \mathbf{H}_{11}^{-1}(c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2)$). However, on very large graphs, inverting the Schur complement $\mathbf{S}$ is infeasible because the dimension of $\mathbf{S}$ is large (see Table 2). Hence, computing $\mathbf{r}_2$ in Equation (9) with $\mathbf{S}^{-1}$ is impractical on billion-scale graphs. Our solution for this problem is to exploit an iterative method to solve the linear system w.r.t. $\mathbf{r}_2$. This approach enables to avoid matrix inversion; consequently, the preprocessing time and the storage cost for $\mathbf{S}^{-1}$ are eliminated. In the preprocessing phase, BEPI-B precomputes several matrices required in Equations (8), (9), and (10). In the query phase, BEPI-B computes those equations for a given seed node based on the precomputed matrices.

BePI-B: Preprocessing phase (Algorithm 1). BEPI-B first reorders the adjacency matrix $\mathbf{A}$ using the deadend reordering technique (line 1). Then, BEPI-B permutes the adjacency matrix $\mathbf{A}_{\text{nn}}$ using the hub-and-spoke reordering method (details in Appendix A) so that the reordered matrix contains a large block diagonal matrix as seen in Figure 3(c) (line 2). Notice that when we permute $\mathbf{A}_{\text{nn}}$, the rows of $\mathbf{A}_{\text{nd}}$ also need to be permuted according to the permutation produced by the hub-and-spoke reordering method. In BEPI-B, we choose a hub selection ratio k which makes the dimension of the Schur complement n_2 small enough in order to concentrate entries of $\mathbf{A}_{\text{nn}}$ as much as possible. Then, BEPI-B computes and partitions $\mathbf{H}$ (lines 3 and 4). When we compute $\mathbf{H}_{11}^{-1}$, we invert the LU factors of $\mathbf{H}_{11}$ since this approach is more efficient in terms of time and space than directly inverting $\mathbf{H}_{11}$ as suggested in [14, 38] (line 5). BEPI-B finally computes the Schur complement of $\mathbf{H}_{11}$ (line 6).

Algorithm 1: Preprocessing phase in BEPI-B and BEPI-S

Input: graph: G, restart probability: c
Output: precomputed matrices: $\mathbf{L}_1^{-1}$, $\mathbf{U}_1^{-1}$, $\mathbf{S}$, $\mathbf{H}_{12}$, $\mathbf{H}_{21}$, $\mathbf{H}_{31}$ and $\mathbf{H}_{32}$.

1: reorder $\mathbf{A}$ using the deadend reordering approach
2: reorder $\mathbf{A}_{nn}$ using the hub-and-spoke reordering method with the following hub selection ratio k:
(BEPI-B only) select k which makes n_2 small
(BEPI-S only) select k which minimizes $|\mathbf{S}|$
3: compute $\tilde{\mathbf{A}}$, and $\mathbf{H} = \mathbf{I} - (1-c)\tilde{\mathbf{A}}^T$
4: partition $\mathbf{H}$ into $\mathbf{H}_{11}, \mathbf{H}_{12}, \mathbf{H}_{21}, \mathbf{H}_{22}, \mathbf{H}_{31}$, and $\mathbf{H}_{32}$
5: decompose $\mathbf{H}_{11}$ into $\mathbf{L}_1$ and $\mathbf{U}_1$ using LU decomposition and compute $\mathbf{L}_1^{-1}$ and $\mathbf{U}_1^{-1}$
6: compute the Schur complement of $\mathbf{H}_{11}$, $\mathbf{S} = \mathbf{H}_{22} - \mathbf{H}_{21}(\mathbf{U}_1^{-1}(\mathbf{L}_1^{-1}(\mathbf{H}_{12})))$
7: **return** $\mathbf{L}_1^{-1}$, $\mathbf{U}_1^{-1}$, $\mathbf{S}$, $\mathbf{H}_{12}$, $\mathbf{H}_{21}$, $\mathbf{H}_{31}$, and $\mathbf{H}_{32}$

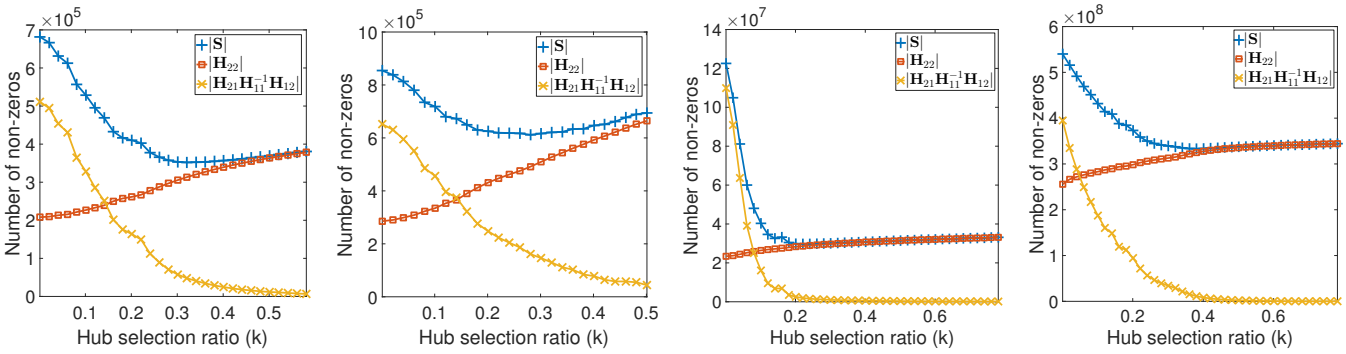
Algorithm 2: Query phase in BEPI-B and BEPI-S

Input: seed node: s, restart probability: c, error tolerance: ϵ, precomputed matrices: $\mathbf{L}_1^{-1}$, $\mathbf{U}_1^{-1}$, $\mathbf{S}$, $\mathbf{H}_{12}$, $\mathbf{H}_{21}$, $\mathbf{H}_{31}$, and $\mathbf{H}_{32}$
Output: relevance vector: $\mathbf{r}$

1: create $\mathbf{q}$ whose sth entry is 1 and the others are 0
2: partition $\mathbf{q}$ into $\mathbf{q}_1$, $\mathbf{q}_2$, and $\mathbf{q}_3$
3: compute $\tilde{\mathbf{q}}_2 = c\mathbf{q}_2 - \mathbf{H}_{21}(\mathbf{U}_1^{-1}(\mathbf{L}_1^{-1}c\mathbf{q}_1)$
4: solve $\mathbf{S}\mathbf{r}_2 = \tilde{\mathbf{q}}_2$ using an iterative method and the error tolerance ϵ
5: compute $\mathbf{r}_1 = \mathbf{U}_1^{-1}(\mathbf{L}_1^{-1}(c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2))$
6: compute $\mathbf{r}_3 = c\mathbf{q}_3 - \mathbf{H}_{31}\mathbf{r}_1 - \mathbf{H}_{32}\mathbf{r}_2$
7: create $\mathbf{r}$ by concatenating $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$
8: **return** $\mathbf{r}$

BePI-B: Query phase (Algorithm 2). In the query phase, BEPI-B computes the RWR score vector $\mathbf{r}$ for a given seed node s based on the precomputed matrices. The vector $\mathbf{q}$ denotes the length-n starting vector whose entry at the index of the seed node s is 1 and otherwise 0. It is partitioned into the length-n_1 vector $\mathbf{q}_1$, the length-n_2 vector $\mathbf{q}_2$, and the length-n_3 vector $\mathbf{q}_3$ (lines 1 and 2). BEPI-B first solves the linear system w.r.t. $\mathbf{r}_2$ in Equation (9) using an iterative method (lines 3 and 4). Then, BEPI-B computes $\mathbf{r}_1$ and $\mathbf{r}_3$ (lines 5 and 6).

Since $\mathbf{S}$ is non-symmetric and invertible [50], any iterative methods for a non-symmetric matrix can be used; in this paper, we use GMRES since it is the state-of-the-art method in terms of efficiency and accuracy. GMRES repeats an iteration procedure until the relative residual is less than an error tolerance ϵ (i.e., $\|\mathbf{S}\mathbf{r}_2^{(i)} - \tilde{\mathbf{q}}_2\|_2 / \|\tilde{\mathbf{q}}_2\|_2 \leq \epsilon$ where $\mathbf{r}_2^{(i)}$ indicates $\mathbf{r}_2$ at the i-th iteration of GMRES).



(a) Slashdot (b) Wikipedia (c) Flickr (d) WikiLink

Figure 4: The number of non-zeros of the Schur complement $|\mathbf{S}|$ with different hub selection ratio k on the Slashdot, the Wikipedia, the Flickr, and the WikiLink datasets. The figures show the trade-off problem for selecting k. If we select large k, then $|\mathbf{S}|$ decreases compared to small k. However, if we choose too large k (e.g., when k is greater than 0.3 in the sub-figures), then $|\mathbf{S}|$ increases. We set k between 0.2 and 0.3 since those constants decrease $|\mathbf{S}|$ enough (see Table 2).

3.4 BePI-S: Sparsifying the Schur Complement

We present BePI-S which improves on BePI-B by decreasing the number of non-zero entries of the Schur complement $\mathbf{S}$ used by the iterative procedure in BePI-B. Since the time complexity of iterative methods depends on the number of non-zeros of the matrix, this approach saves time for solving the linear system on $\mathbf{S}$. Also, decreasing non-zero entries of $\mathbf{S}$ reduces the storage cost for $\mathbf{S}$. By the definition of $\mathbf{S}$ (i.e., $\mathbf{S} = \mathbf{H}_{22} - \mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12}$), the entries of $\mathbf{S}$ are determined by $\mathbf{H}_{22}$ and $\mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12}$. Thus, the number of non-zeros of $\mathbf{S}$ is roughly bounded as follows:

$$|\mathbf{S}| \leq |\mathbf{H}_{22}| + |\mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12}|$$

where $|\mathbf{A}|$ is the number of non-zeros of the matrix $\mathbf{A}$.

To decrease the number of non-zeros of $\mathbf{S}$, BePI-S sets a hub selection ratio k which minimizes the number of non-zeros of $\mathbf{S}$. If we increase k, the hub-and-spoke reordering method selects more hubs at each step; therefore, n_2 increases, and n_1 decreases (i.e., $n - n_3 = n_1 + n_2$). In other words, $|\mathbf{H}_{22}|$ increases while $|\mathbf{H}_{11}|$, $|\mathbf{H}_{12}|$, and $|\mathbf{H}_{21}|$ decrease; thus, $|\mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12}|$ is also reduced. The point is that, with a suitable choice of k, $|\mathbf{S}|$ decreases since $|\mathbf{H}_{22}|$ slightly increases while $|\mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12}|$ is significantly reduced. Note that this is a trade-off problem between the number of entries of $\mathbf{H}_{22}$ and that of $\mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12}$. If we set k too large, then although $|\mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12}|$ decreases a lot, $|\mathbf{H}_{22}|$ also increases a lot; therefore, $|\mathbf{S}|$ becomes large. Figure 4 illustrates the trade-off problem on real-world graphs.

BePI-S: Preprocessing phase (Algorithm 1). BePI-S precomputes the matrices demanded in the query phase on top of BePI-B. First of all, BePI-S reorders $\mathbf{A}$ and $\mathbf{A}_{nn}$ using the deadend and the hub-and-spoke reordering methods similarly to BePI-B (line 1 and 2). However, when BePI-S reorders $\mathbf{A}_{nn}$ using the hub-and-spoke reordering method, we set a hub selection ratio k which minimizes the number of non-zeros of the Schur complement $\mathbf{S}$ (line 2). We empirically select k as 0.2 or 0.3 which makes the Schur complement sparse enough, as presented in Figure 4 and Table 2. As we will discuss in Section 4.5, BePI-S accelerates preprocessing speed by up to 10× and saves memory space by up to 5× compared to BePI-B.

BePI-S: Query phase (Algorithm 2). BePI-S computes RWR scores for a given seed node based on the precomputed matrices. Note that the query phase of BePI-S is the same as that of BePI-B. However, the query speed of BePI-S is faster than that of BePI-B because BePI-S decreases the number of non-zeros of the Schur complement used in the iterative method (line 4). As we will see in Section 4.5, BePI-S leads to up to 5× performance improvement in terms of query speed compared to BePI-B.

3.5 BePI: Preconditioning a Linear System for the Iterative Method

Our final method BePI improves BePI-S by exploiting a preconditioner [7] to enhance the speed of the iterative method in the query phase. The main purpose of preconditioning is to modify a linear system so that iterative methods converge faster. More specifically, preconditioning decreases the condition number of the matrix to be solved and makes the eigenvalues of the modified system to form a tighter cluster away from the origin. The small condition number and the tight eigenvalue distribution are the main criteria for fast convergence [37, 42]. A standard approach is to use a non-singular matrix $\mathbf{M}$ as a preconditioner. With $\mathbf{M}$, a linear system $\mathbf{Ax} = \mathbf{b}$ is preconditioned to $\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}\mathbf{b}$. Notice that the solution of the original system is the same as that of the preconditioned system.

BePI exploits a preconditioner to make convergence faster when solving the linear system of $\mathbf{S}$ in Equation (9) using an iterative method. Among various preconditioning techniques such as incomplete LU decomposition (ILU) [36] or Sparse Approximate Inverse (SPAI) [8], we choose ILU as a preconditioner because ILU factors are easily computed and effective for preconditioning. The incomplete LU decomposition of a matrix $\mathbf{A}$ is a sparse approximation of the LU factors of the matrix, i.e., $\mathbf{A} \simeq \tilde{\mathbf{L}}\tilde{\mathbf{U}}$. The ILU factors, $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$, have the same sparsity pattern as the lower and upper triangular parts of $\mathbf{A}$, respectively.

The linear system, $\mathbf{Sr}_2 = \tilde{\mathbf{q}}_2$, in Equation (9) is preconditioned with the ILU factors of $\mathbf{S}$ as follows:

$$\tilde{\mathbf{U}}_2^{-1}\tilde{\mathbf{L}}_2^{-1}\mathbf{Sr}_2 = \tilde{\mathbf{U}}_2^{-1}\tilde{\mathbf{L}}_2^{-1}\tilde{\mathbf{q}}_2 \tag{11}$$

where $\mathbf{S} \simeq \tilde{\mathbf{L}}_2\tilde{\mathbf{U}}_2$ and $\tilde{\mathbf{q}}_2 = c\mathbf{q}_2 - \mathbf{H}_{21}(\mathbf{H}_{11}^{-1}(c\mathbf{q}_1))$. Then, an iterative method finds the solution $\mathbf{r}_2$ of the preconditioned system in Equation (11). However, it is difficult to explicitly construct the preconditioned system due to the inversion of the ILU factors. Instead of directly obtaining $\tilde{\mathbf{U}}_2^{-1}$ and $\tilde{\mathbf{L}}_2^{-1}$, many preconditioned iterative methods, such as preconditioned GMRES [35], involve a procedure which iteratively

Algorithm 3: Preprocessing phase in BePI

Input: graph: G, restart probability: c
Output: precomputed matrices: $\mathbf{L}_1^{-1}$, $\mathbf{U}_1^{-1}$, $\mathbf{S}$, $\tilde{\mathbf{L}}_2$, $\tilde{\mathbf{U}}_2$, $\mathbf{H}_{12}$, $\mathbf{H}_{21}$, $\mathbf{H}_{31}$ and $\mathbf{H}_{32}$.

1: reorder $\mathbf{A}$ using the deadend reordering approach
2: reorder $\mathbf{A}_{nn}$ using the hub-and-spoke reordering method with a hub selection ratio k which minimizes $|\mathbf{S}|$
3: compute $\tilde{\mathbf{A}}$, and $\mathbf{H} = \mathbf{I} - (1-c)\tilde{\mathbf{A}}^T$
4: partition $\mathbf{H}$ into $\mathbf{H}_{11}, \mathbf{H}_{12}, \mathbf{H}_{21}, \mathbf{H}_{22}, \mathbf{H}_{31}$, and $\mathbf{H}_{32}$
5: decompose $\mathbf{H}_{11}$ into $\mathbf{L}_1$ and $\mathbf{U}_1$ using LU decomposition and compute $\mathbf{L}_1^{-1}$ and $\mathbf{U}_1^{-1}$
6: compute the Schur complement of $\mathbf{H}_{11}$, $\mathbf{S} = \mathbf{H}_{22} - \mathbf{H}_{21}(\mathbf{U}_1^{-1}(\mathbf{L}_1^{-1}(\mathbf{H}_{12})))$
7: compute incomplete LU factors of $\mathbf{S} \simeq \tilde{\mathbf{L}}_2\tilde{\mathbf{U}}_2$
8: **return** $\mathbf{L}_1^{-1}$, $\mathbf{U}_1^{-1}$, $\mathbf{S}$, $\tilde{\mathbf{L}}_2$, $\tilde{\mathbf{U}}_2$, $\mathbf{H}_{12}$, $\mathbf{H}_{21}$, $\mathbf{H}_{31}$, and $\mathbf{H}_{32}$

Algorithm 4: Query phase in BePI

Input: seed node: s, restart probability: c, error tolerance: ϵ, precomputed matrices: $\mathbf{L}_1^{-1}$, $\mathbf{U}_1^{-1}$, $\mathbf{S}$, $\tilde{\mathbf{L}}_2$, $\tilde{\mathbf{U}}_2$, $\mathbf{H}_{12}$, $\mathbf{H}_{21}$, $\mathbf{H}_{31}$, and $\mathbf{H}_{32}$
Output: relevance vector: $\mathbf{r}$

1: create $\mathbf{q}$ whose sth entry is 1 and the others are 0
2: partition $\mathbf{q}$ into $\mathbf{q}_1$, $\mathbf{q}_2$, and $\mathbf{q}_3$
3: compute $\tilde{\mathbf{q}}_2 = c\mathbf{q}_2 - \mathbf{H}_{21}(\mathbf{U}_1^{-1}(\mathbf{L}_1^{-1}c\mathbf{q}_1))$
4: solve the preconditioned system $\tilde{\mathbf{U}}_2^{-1}\tilde{\mathbf{L}}_2^{-1}\mathbf{S}\mathbf{r}_2 = \tilde{\mathbf{U}}_2^{-1}\tilde{\mathbf{L}}_2^{-1}\tilde{\mathbf{q}}_2$ using a preconditioned iterative method with $\tilde{\mathbf{L}}_2$ and $\tilde{\mathbf{U}}_2$, and the error tolerance ϵ
5: compute $\mathbf{r}_1 = \mathbf{U}_1^{-1}(\mathbf{L}_1^{-1}(c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2))$
6: compute $\mathbf{r}_3 = c\mathbf{q}_3 - \mathbf{H}_{31}\mathbf{r}_1 - \mathbf{H}_{32}\mathbf{r}_2$
7: create $\mathbf{r}$ by concatenating $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$
8: **return** $\mathbf{r}$

preconditions the original system by taking advantage of triangular matrix, $\tilde{\mathbf{L}}_2$ and $\tilde{\mathbf{U}}_2$, without explicitly constructing the preconditioned system and inverting the preconditioner (see more details in Appendix A). We exploit a preconditioned iterative method to solve the preconditioned system in Equation (11) with the preconditioner.

BePI: Preprocessing phase (Algorithm 3). BePI precomputes the matrices required for computing RWR scores in the query phase. When BePI reorders nodes using the hub-and-spoke reordering method, BePI also chooses the hub selection ratio k which minimizes the number of nonzeros of the Schur complement $\mathbf{S}$ as in BePI-S (line 2). After reordering nodes, BePI computes $\mathbf{H}$ and the Schur complement $\mathbf{S}$ (lines 3∼6). Then, BePI calculates the ILU factors of $\mathbf{S}$, $\tilde{\mathbf{L}}_2$ and $\tilde{\mathbf{U}}_2$ (line 7) to obtain a preconditioner for the iterative method in the query phase. Note that the storage cost of $\tilde{\mathbf{L}}_2$ and $\tilde{\mathbf{U}}_2$ is the same as that of $\mathbf{S}$, since $\tilde{\mathbf{L}}_2$ and $\tilde{\mathbf{U}}_2$ follow the same sparsity pattern of $\mathbf{S}$.

BePI: Query phase (Algorithm 4). In the query phase, BePI computes Equations (8), (9), and (10) to obtain the RWR score vector $\mathbf{r}$ w.r.t. a seed node s based on the matrices precomputed by Algorithm 3. BePI first sets the starting vector $\mathbf{q}$ for given seed node s (lines 1 and 2). Then, BePI solves the preconditioned system in Equation (11) using an iterative method such as preconditioned GMRES (see details in Appendix A) with the preconditioner $\tilde{\mathbf{L}}_2$ and $\tilde{\mathbf{U}}_2$ (lines 3 and 4). After obtaining $\mathbf{r}_2$, BePI computes $\mathbf{r}_1$ and $\mathbf{r}_3$ (lines 5 and 6). As we will see in Section 4.5, the preconditioner accelerates query speed by up to 4× compared to BePI-S.

3.6 Theoretical Results

We analyze the time and space complexities of BePI. Moreover, we analyze the accuracy bound of BePI, since BePI exploits an iterative method. Note that all matrices are saved in a sparse matrix format such as compressed column storage [12] which contains only non-zero entries and their locations, and all sparse matrix operations such as sparse matrix vector multiplication only consider non-zero entries to exploit such sparsity.

3.6.1 Time Complexity

We provide proofs for the time complexity of BePI.

THEOREM 1. *The preprocessing phase of* BePI *takes* $O(\lceil n_2/(k\times l)\rceil(m + l\log l) + \sum_{i=1}^{b} n_{1i}^3 + n_2\sum_{i=1}^{b} n_{1i}^2 + |\mathbf{S}| + \min(n_2^2 n_1, n_2 m))$ *where* $l = n_1 + n_2$ *and* k *is the hub selection ratio of the hub-and-spoke reordering method.*

PROOF. *Computing* $\mathbf{L}_1^{-1}$, $\mathbf{U}_1^{-1}$, *and* $\mathbf{S}$ *after doing the hub-and-spoke reordering method takes* $O(\lceil n_2/(k\times l)\rceil(m + l\log l) + \sum_{i=1}^{b} n_{1i}^3 + n_2\sum_{i=1}^{b} n_{1i}^2 + \min(n_2^2 n_1, n_2 m))$ [38] *where* $l = n_1 + n_2$ *and* $\lceil n_2/(k\times n)\rceil$ *indicates the number of iterations of* SlashBurn. *Since incomplete LU decomposition for a sparse matrix* $\mathbf{A}$ *takes* $O(|\mathbf{A}|)$ [36], *it takes* $O(|\mathbf{S}|)$ *to compute* $\tilde{\mathbf{L}}_2$ *and* $\tilde{\mathbf{U}}_2$. □

According to Theorem 1, the preprocessing cost of BePI mainly depends on the number of iterations of the reordering method and the computations related to the Schur complement. Note that since the number of iterations of the hub-and-spoke reordering method [23] and $|\mathbf{S}|$ are reduced as k increases, the preprocessing cost decreases as in Figures 6(a) and 8. Also, Theorem 1 indicates that the preprocessing cost of BePI is much smaller than that of Bear, the state-of-the-art block elimination approach, since BePI demands $O(|\mathbf{S}|)$ and Bear requires $O(n_2^3)$ (i.e., $|\mathbf{S}| \ll n_2^3$) while other factors are the same in both methods.

THEOREM 2. *The query phase of* BePI *takes* $O(\sum_{i=1}^{b} n_{1i}^2 + \min(n_1 n_2, m) + min(n_1 n_3, m) + min(n_2 n_3, m) + T|\mathbf{S}|)$ *where* T *is the number of iterations.*

PROOF. *Since it takes* $O(\sum_{i=1}^{b} n_{1i}^2 + \min(n_1 n_2, m))$ *to compute* $\tilde{\mathbf{q}}_2 = c\mathbf{q}_2 - \mathbf{H}_{21}(\mathbf{H}_{11}^{-1}(c\mathbf{q}_1))$ [38], *and solving a sparse linear system* $\mathbf{A}\mathbf{x} = \mathbf{b}$ *with an iterative method takes* $O(T|\mathbf{A}|)$ *where* T *is the number of iterations* [37], *it takes* $O(\sum_{i=1}^{b} n_{1i}^2 + \min(n_1 n_2, m) + T|\mathbf{S}|)$ *to solve the linear system of* $\mathbf{S}$. *Note that the time complexity for computing* $\mathbf{r}_1$ *is the same as that of* $\tilde{\mathbf{q}}_2$. *For* $\mathbf{r}_3$, *it takes* $O(min(n_1 n_3, m) + min(n_2 n_3, m))$. □

Theorem 2 implies that the query cost of BePI mainly depends on the number of iterations T and $|\mathbf{S}|$. Since $|\mathbf{S}|$ and T are reduced by the sparsification of the Schur complement and the preconditioner, respectively, the query cost of BePI decreases compared to those of BePI-B and BePI-S.

3.6.2 Space Complexity

We provide a proof for the space complexity of BePI.

THEOREM 3. BePI *requires* $O(\sum_{i=1}^{b} n_{1i}^2 + \min(n_1 n_2, m) + min(n_1 n_3, m) + min(n_2 n_3, m) + |\mathbf{S}|)$ *memory space for preprocessed matrices:* $\mathbf{L}_1^{-1}$, $\mathbf{U}_1^{-1}$, $\mathbf{S}$, $\tilde{\mathbf{L}}_2$, $\tilde{\mathbf{U}}_2$, $\mathbf{H}_{12}$, $\mathbf{H}_{21}$, $\mathbf{H}_{31}$, *and* $\mathbf{H}_{32}$.

PROOF. *It requires* $O(min(n_1 n_2, m))$ *memory space for* $\mathbf{H}_{12}$ *and* $\mathbf{H}_{12}$, *and* $O(\sum_{i=1}^{b} n_{1i}^2)$ *memory space for* $\mathbf{L}_1^{-1}$ *and* $\mathbf{U}_1^{-1}$ [38]. *Also, it requires* $O(min(n_1 n_3, m) + min(n_2 n_3, m))$ *memory space for* $\mathbf{H}_{31}$ *and* $\mathbf{H}_{32}$. *Since the space cost for incomplete LU factors is the same as that of the given sparse matrix, it requires* $O(|\mathbf{S}|)$ *for* $\mathbf{S}$, $\tilde{\mathbf{L}}_2$ *and* $\tilde{\mathbf{U}}_2$. □

Theorem 3 indicates that the space cost of BePI mainly depends on $O(|\mathbf{S}|)$ because the number of non-zeros of $\mathbf{S}$ is larger than those of other matrices except for the incomplete LU factors of $\mathbf{S}$. Note that BePI demands much smaller memory space than the state-of-the-art method Bear because the space cost of Bear mainly depends on $O(n_2^2)$; i.e., $|\mathbf{S}| \ll n_2^2$. Also, through sparsifying the Schur complement $\mathbf{S}$, the space costs of BePI and BePI-S decrease compared to that of BePI-B which is the basic version without sparsifying $\mathbf{S}$.

3.6.3 Accuracy Bound

We analyze the accuracy bound of the RWR score vector $\mathbf{r}$ computed by BePI. Since $\mathbf{r}$ consists of $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$, and $\mathbf{r}_1$ and $\mathbf{r}_3$ are computed after $\mathbf{r}_2$, we first analyze the bound of $\mathbf{r}_2$ in Lemma 2, that of $\mathbf{r}_1$ in Lemma 3, and that of $\mathbf{r}_3$ in Lemma 4. Then, we conclude the bound of $\mathbf{r}$ in Theorem 4 using these lemmas.

LEMMA 2 (ACCURACY BOUND OF $\mathbf{r}_2$). *Let $\mathbf{r}_2^*$ be the true solution of the linear system $\mathbf{S}\mathbf{r}_2 = \tilde{\mathbf{q}}_2$ where $\tilde{\mathbf{q}}_2 = c\mathbf{q}_2 - \mathbf{H}_{21}(\mathbf{H}_{11}^{-1}(c\mathbf{q}_1))$, and $\mathbf{r}_2^{(k)}$ be the solution computed by* BePI *after the relative residual becomes less than a given tolerance ϵ at the k-th iteration. Then, $||\mathbf{r}_2^* - \mathbf{r}_2^{(k)}||_2 \leq \frac{||\tilde{\mathbf{q}}_2||_2}{\sigma_{min}(\mathbf{S})}\epsilon$ where $\sigma_{min}(\mathbf{S})$ is the smallest singular value of $\mathbf{S}$.*

PROOF. See Appendix E. □

LEMMA 3 (ACCURACY BOUND OF $\mathbf{r}_1$). *Let $\mathbf{r}_1^*$ be the true solution of the linear system $\mathbf{H}_{11}\mathbf{r}_1 = \tilde{\mathbf{q}}_1$ where $\tilde{\mathbf{q}}_1 = c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2$, and $\mathbf{r}_1^{(k)}$ be the solution of $\mathbf{H}_{11}\mathbf{r}_1^{(k)} = \tilde{\mathbf{q}}_1^{(k)}$ where $\tilde{\mathbf{q}}_1^{(k)} = c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2^{(k)}$. Then, $||\mathbf{r}_1^* - \mathbf{r}_1^{(k)}||_2 \leq \frac{||\mathbf{H}_{12}||_2}{\sigma_{min}(\mathbf{H}_{11})}||\mathbf{r}_2^* - \mathbf{r}_2^{(k)}||_2 \leq \frac{||\mathbf{H}_{12}||_2||\tilde{\mathbf{q}}_2||_2}{\sigma_{min}(\mathbf{H}_{11})\sigma_{min}(\mathbf{S})}\epsilon$ where $\sigma_{min}(\mathbf{A})$ is the smallest singular value of a matrix $\mathbf{A}$.*

PROOF. See Appendix F. □

LEMMA 4 (ACCURACY BOUND OF $\mathbf{r}_3$). *Let $\mathbf{r}_3^*$ be the true solution of the equation $\mathbf{r}_3 = c\mathbf{q}_3 - \mathbf{H}_{31}\mathbf{r}_1^* - \mathbf{H}_{32}\mathbf{r}_2^*$, and $\mathbf{r}_3^{(k)}$ be the solution of $\mathbf{r}_3^{(k)} = c\mathbf{q}_3 - \mathbf{H}_{31}\mathbf{r}_1^{(k)} - \mathbf{H}_{32}\mathbf{r}_2^{(k)}$. Then, $\|\mathbf{r}_3^* - \mathbf{r}_3^{(k)}\|_2 \leq \|\mathbf{H}_{31}\|_2\|\mathbf{r}_1^* - \mathbf{r}_1^{(k)}\|_2 + \|\mathbf{H}_{32}\|_2\|\mathbf{r}_2^* - \mathbf{r}_2^{(k)}\|_2$.*

PROOF. See Appendix G. □

THEOREM 4 (ACCURACY BOUND OF BePI). *Let $\mathbf{r}^*$ be the true solution of the linear system $\mathbf{H}\mathbf{r} = c\mathbf{q}$, and $\mathbf{r}^{(k)}$ be the solution $\mathbf{r}^{(k)} = \left[\mathbf{r}_1^{(k)}, \mathbf{r}_2^{(k)}, \mathbf{r}_3^{(k)}\right]^{\mathbf{T}}$ where $\mathbf{r}_2^{(k)}$ is the solution of $\mathbf{S}\mathbf{r}_2 = \tilde{\mathbf{q}}_2$ computed by* BePI *after the residual becomes less than the error tolerance ϵ at the k-th iteration, $\mathbf{r}_1^{(k)}$ is the solution of $\mathbf{H}_{11}\mathbf{r}_1^{(k)} = c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2^{(k)}$, and $\mathbf{r}_3^{(k)}$ is the solution of $\mathbf{r}_3 = c\mathbf{q}_3 - \mathbf{H}_{31}\mathbf{r}_1^{(k)} - \mathbf{H}_{32}\mathbf{r}_2^{(k)}$. Let $\alpha = \frac{||\mathbf{H}_{12}||_2}{\sigma_{min}(\mathbf{H}_{11})}$. Then, $||\mathbf{r}^* - \mathbf{r}^{(k)}||_2$ is bounded as follows:*

$$||\mathbf{r}^* - \mathbf{r}^{(k)}||_2 \leq \left(\sqrt{(\alpha||\mathbf{H}_{31}||_2 + ||\mathbf{H}_{32}||_2)^2 + \alpha^2 + 1}\right)\frac{||\tilde{\mathbf{q}}_2||_2}{\sigma_{min}(\mathbf{S})}\epsilon.$$

PROOF. *By the definition of L2-norm, $||\mathbf{r}^* - \mathbf{r}^{(k)}||_2^2$ is represented as follows:*

$$||\mathbf{r}^* - \mathbf{r}^{(k)}||_2^2 = \left\|\begin{bmatrix}\mathbf{r}_1^* - \mathbf{r}_1^{(k)}\\ \mathbf{r}_2^* - \mathbf{r}_2^{(k)}\\ \mathbf{r}_3^* - \mathbf{r}_3^{(k)}\end{bmatrix}\right\|_2^2$$
$$= \|\mathbf{r}_1^* - \mathbf{r}_1^{(k)}\|_2^2 + \|\mathbf{r}_2^* - \mathbf{r}_2^{(k)}\|_2^2 + \|\mathbf{r}_3^* - \mathbf{r}_3^{(k)}\|_2^2$$

Then, by Lemma 4, it is bounded as follows:

$$||\mathbf{r}^* - \mathbf{r}^{(k)}||_2^2 \leq \|\mathbf{r}_1^* - \mathbf{r}_1^{(k)}\|_2^2 + \|\mathbf{r}_2^* - \mathbf{r}_2^{(k)}\|_2^2 + \|\mathbf{H}_{31}\|_2^2\|\mathbf{r}_1^* - \mathbf{r}_1^{(k)}\|_2^2$$
$$+ \|\mathbf{H}_{32}\|_2^2\|\mathbf{r}_2^* - \mathbf{r}_2^{(k)}\|_2^2 + 2\|\mathbf{H}_{31}\|_2\|\mathbf{H}_{32}\|_2\|\mathbf{r}_1^* - \mathbf{r}_1^{(k)}\|_2\|\mathbf{r}_2^* - \mathbf{r}_2^{(k)}\|_2$$

From Lemma 3, $||\mathbf{r}_1^* - \mathbf{r}_1^{(k)}||_2 \leq \frac{||\mathbf{H}_{12}||_2}{\sigma_{min}(\mathbf{H}_{11})}||\mathbf{r}_2^* - \mathbf{r}_2^{(k)}||_2 = \alpha||\mathbf{r}_2^* - \mathbf{r}_2^{(k)}||_2$ where $\alpha = \frac{||\mathbf{H}_{12}||_2}{\sigma_{min}(\mathbf{H}_{11})}$. Hence, the bound is represented as follows:

$$||\mathbf{r}^* - \mathbf{r}^{(k)}||_2^2 \leq \|\mathbf{r}_2^* - \mathbf{r}_2^{(k)}\|_2^2\Big(\alpha^2 + 1 + \alpha^2\|\mathbf{H}_{31}\|_2^2 + \|\mathbf{H}_{32}\|_2^2 + 2\alpha\|\mathbf{H}_{31}\|_2\|\mathbf{H}_{32}\|_2\Big)$$
$$= ||\mathbf{r}_2^* - \mathbf{r}_2^{(k)}||_2^2\left(\alpha^2 + 1 + (\alpha\|\mathbf{H}_{31}\|_2 + \|\mathbf{H}_{32}\|_2)^2\right).$$

By Lemma 2, $\|\mathbf{r}_2^* - \mathbf{r}_2^{(k)}\|_2 \leq \frac{||\tilde{\mathbf{q}}_2||_2}{\sigma_{min}(\mathbf{S})}\epsilon$; thus, the above inequality is written as follows:

$$||\mathbf{r}^* - \mathbf{r}^{(k)}||_2^2 \leq \left(\alpha^2 + 1 + (\alpha\|\mathbf{H}_{31}\|_2 + \|\mathbf{H}_{32}\|_2)^2\right)\left(\frac{\|\tilde{\mathbf{q}}_2\|_2}{\sigma_{min}(\mathbf{S})}\epsilon\right)^2.$$

Finally, the bound of $||\mathbf{r}^* - \mathbf{r}^{(k)}||_2$ is represented in the following inequality:

$$||\mathbf{r}^* - \mathbf{r}^{(k)}||_2 \leq \left(\sqrt{(\alpha||\mathbf{H}_{31}||_2 + ||\mathbf{H}_{32}||_2)^2 + \alpha^2 + 1}\right)\frac{||\tilde{\mathbf{q}}_2||_2}{\sigma_{min}(\mathbf{S})}\epsilon$$ □

According to Theorem 4, the accuracy of BePI is bounded by the norms and the smallest singular values of the input matrices and the error tolerance ϵ. Also, Theorem 4 indicates that BePI guarantees $||\mathbf{r}^* - \mathbf{r}^{(k)}||_2 \leq \epsilon_T$ where ϵ_T is the target accuracy if we set the error tolerance to ϵ satisfying the following inequality:

$$0 < \epsilon \leq \left(\sqrt{(\alpha||\mathbf{H}_{31}||_2 + ||\mathbf{H}_{32}||_2)^2 + \alpha^2 + 1}\right)^{-1}\frac{\sigma_{min}(\mathbf{S})}{||\tilde{\mathbf{q}}_2||_2}\epsilon_T.$$

4. EXPERIMENTS

In this section, we evaluate the performance of our method BePI, and compare it to other existing methods for computing RWR scores. We aim to answer the following questions from the experiments:

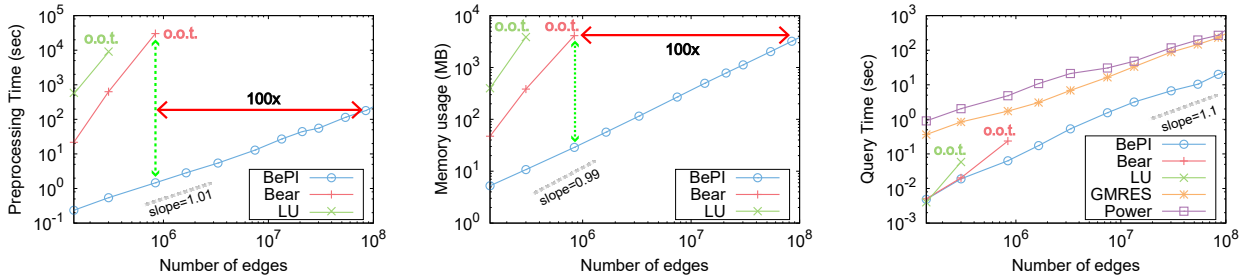
- **Q1. Preprocessing cost (Section 4.2).** How much memory space do BePI and other methods require for their preprocessed results? How long does this preprocessing phase take?
- **Q2. Query cost (Section 4.3).** How quickly does BePI respond to an RWR query compared to other methods?
- **Q3. Scalability (Section 4.4).** How well does BePI scale up compared to other methods?
- **Q4. Effectiveness of the optimizations (Section 4.5).** How effective are the sparsification of the Schur complement (Section 3.4) and the preconditioning (Section 3.5) in terms of preprocessing and query cost?
- **Q5. Effects of the hub selection ratio k (Section 4.6)** How does the hub selection ratio k in Algorithm 3 affect the performance of BePI in terms of running time and memory requirement?

4.1 Experimental Settings

Machine. All experiments are conducted on a workstation with a single CPU Intel(R) Xeon(R) CPU E7540 @ 2.00GHz and 500GB memory.

Table 2: Summary of real-world datasets. A brief description of each dataset is in Appendix H. n is the number of nodes, m is the number of edges, and k is the hub selection ratio in SlashBurn used for BePI-S and BePI. n_1 is the number of spokes, n_2 is the number of hubs, and n_3 is the number of deadends. For BePI-B, we set k to 0.001 in SlashBurn. Note that n_2 of BePI-S are the same as that of BePI.

dataset	n	m	k	n_1 in BePI-B	n_1 in BePI and BePI-S	n_2 in BePI-B	n_2 in BePI and BePI-S	n_3
Slashdot	79,120	515,581	0.30	37,872	31,920	7,728	13,680	33,520
Wikipedia	100,312	1,627,472	0.25	79,737	72,187	16,512	24,062	4,063
Baidu	415,641	3,284,317	0.20	347,596	315,586	46,886	78,896	21,159
Flickr	2,302,925	33,140,017	0.20	1,717,120	1,554,006	225,388	388,502	360,417
LiveJournal	4,847,571	68,475,391	0.30	3,138,041	2,655,345	1,156,291	1,638,987	553,239
WikiLink	11,196,007	340,240,450	0.20	8,670,438	8,062,003	2,505,984	3,114,419	19,585
Twitter	41,652,230	1,468,365,182	0.20	33,927,419	24,061,969	6,175,862	16,041,312	1,548,949
Friendster	68,349,466	2,586,147,869	0.20	43,666,118	33,666,118	12,444,080	22,444,080	12,239,268



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

Figure 5: Scalability of BePI compared to other methods on the WikiLink dataset. (a), (b), and (c) show the scalability of the three methods in terms of the number of edges. o.o.t. stands for out of time (more than 24 hours). BePI shows up to 100× better scalability than existing preprocessing methods, and scales well with regard to the size of graphs. Also, BePI provides near linear scalability in terms of preprocessing and query cost.

Methods. We compare our methods with power iteration, LU decomposition, a Krylov subspace method (GMRES), and Bear, all of which are described in Section 2. We evaluate our approach using three different versions:

- BePI-B is the basic version without the sparsification of the Schur complement and the preconditioner.
- BePI-S exploits only the sparsification of the Schur complement without the preconditioner.
- BePI uses both the sparsification of the Schur complement and the preconditioner.

Approximate methods are excluded from the experiments since all the aforementioned methods including our methods compute exact RWR scores. All these methods are implemented in C++ and Eigen [19] which is an open source C++ numerical linear algebra package.

Data. The graph data used in our experiments are summarized in Table 2. A brief description of each dataset is in Appendix H.

Parameters. We set the restart probability c to 0.05 as in the previous works [41, 38]. For Bear and BePI-B, we set k of the hub-and-spoke reordering method to 0.001 as in the previous work [38]. For BePI-S and BePI, we set k of the hub-and-spoke reordering method differently for each dataset as described in Table 2 to make the Schur complement sparse. For larger graphs, 0.2 is usually used for k. The error tolerance ϵ for power iteration, GMRES, and our method is set to 10^{-9}. We set the time limit for preprocessing to 24 hours.

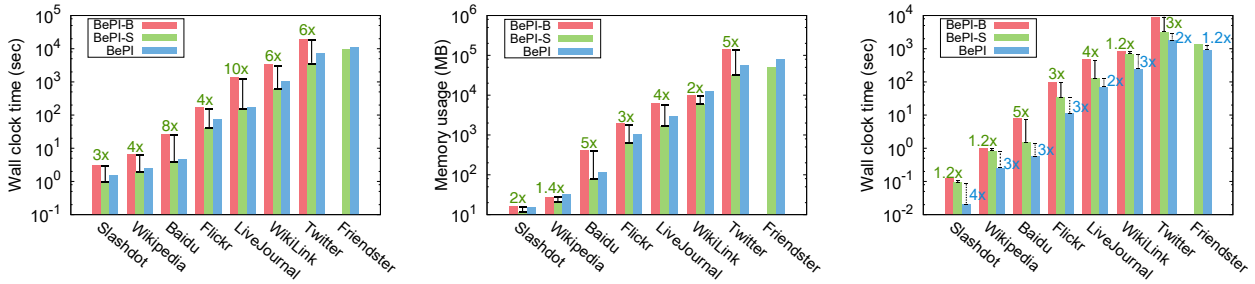
4.2 Preprocessing Cost

We examine the cost of the preprocessing phase of BePI in terms of preprocessing time and memory space for preprocessed data. We compare our method with Bear and LU decomposition, the best preprocessing methods. Preprocessing time is measured in wall-clock time, and it includes the time taken for SlashBurn in BePI and Bear. Figures 1(a) and 1(b) show the preprocessing time and the memory space usage of preprocessed data. Note that only BePI successfully performs the preprocessing phase for all the datasets, while other methods fail because their memory requirements are high, or they run out of time. As seen in Figure 1(a), BePI requires the least amount of time, which is less than about 2 hours for all the datasets. For the Slashdot dataset, which is the smallest dataset, BePI is $3,679\times$ faster than Bear. For other datasets, Bear and LU decomposition fail to show the results (they took more than 24 hours). To compare memory efficiency, we measure how much memory each method requires for the preprocessed matrices. As seen in Figure 1(b), BePI requires the least amount of space for preprocessed matrices. BePI requires up to $130\times$ less memory space than other competitors in all the datasets, which indicates the superiority of our method in terms of scalability compared to other preprocessing methods.

4.3 Query Cost

We compare BePI with other methods in terms of query cost. We compare our method with power iteration, GMRES, Bear, and LU decomposition. We measure the average query time for 30 random seed nodes.

As presented in Figure 1(c), only BePI and iterative methods successfully compute RWR scores on all the datasets, and BePI outperforms competitors for large graphs. For the Baidu dataset, BePI is up to $9\times$ faster than GMRES, which



(a) Effect on preprocessing time (b) Effect on space for preprocessed data (c) Effect on query time

Figure 6: Effect of the sparsification of the Schur complement and the preconditioning. In these figures, bars are omitted in case the corresponding experiments run out of memory. In terms of the effect of the sparsification of the Schur complement, (a) and (b) show that the preprocessing cost is reduced: BePI-S is up to 10× faster than BePI-B, and BePI-S requires up to 5× less memory space than BePI-B. Moreover, (c) presents that the query time is also decreased: BePI-S is up to 5× faster than BePI-B in the query phase. In terms of the effect of the preconditioning, the preprocessing cost of BePI is slightly larger than that of BePI-S as seen in (a) and (b) due to the additional operation for incomplete LU factors. However, BePI is up to 4× faster than BePI-S in the query phase thanks to the effect of the preconditioning as shown in (c).

is the second best one. For the largest Friendster dataset, BePI is 3× faster than GMRES. Compared to power iteration, BePI is 19× and 10× faster for the Baidu and the Friendster datasets, respectively.

4.4 Scalability

We compare the scalability of BePI against existing methods, in terms of the number of edges. For the WikiLink dataset, we extract the principal submatrices, which are the upper left part of the adjacency matrix, of different lengths so that the number of edges of each matrix is different. For each submatrix, we preprocess the matrix using BePI, Bear, and LU decomposition. Then, we compute RWR scores using BePI, Bear, LU decomposition, power iteration, and GMRES. We measure preprocessing time, memory usage and average query time for 30 randomly selected seed nodes.

Figure 5 presents that BePI shows a good scalability with respect to the number of edges, while other preprocessing methods fail to scale up. As shown in Figures 5(a) and 5(b), BePI processes 100× larger graph, while using less memory space than other preprocessing methods. Also, the slope of the fitted line for BePI is 1.01 in Figure 5(a), 0.99 in Figure 5(b), and 1.1 in Figure 5(c). These results indicate that BePI provides near linear scalability in terms of preprocessing and query cost.

4.5 Effects of Sparse Schur Complement and Preconditioning

4.5.1 *Effects on Preprocessing Phase*

We examine the effects of the sparsification of the Schur complement (Section 3.4) and the preconditioning (Section 3.5) in the preprocessing phase of BePI. We measure the preprocessing time and the space for preprocessed data required by BePI, BePI-S, and BePI-B for each dataset.

To investigate the effect of the sparsification of the Schur complement, we first compare BePI-B with BePI-S in terms of the preprocessing time and the memory space. For preprocessing time, Figure 6(a) shows that BePI-S is up to 10× faster than BePI-B. For memory space, Figure 6(b) presents that BePI-S requires up to 5× less memory space than BePI-B. Table 3 summarizes the reduction of non-zero entries of the Schur complement after applying the sparsification of the Schur complement. For all datasets, the number of non-zero entries of $\mathbf{S}$ decreases by the sparsification. Especially, BePI-S reduces the number of non-zeros of $\mathbf{S}$ by 9.8× than BePI-B for the Baidu dataset. BePI-B runs out of time when computing $\mathbf{S}$ for the largest Friendster dataset.

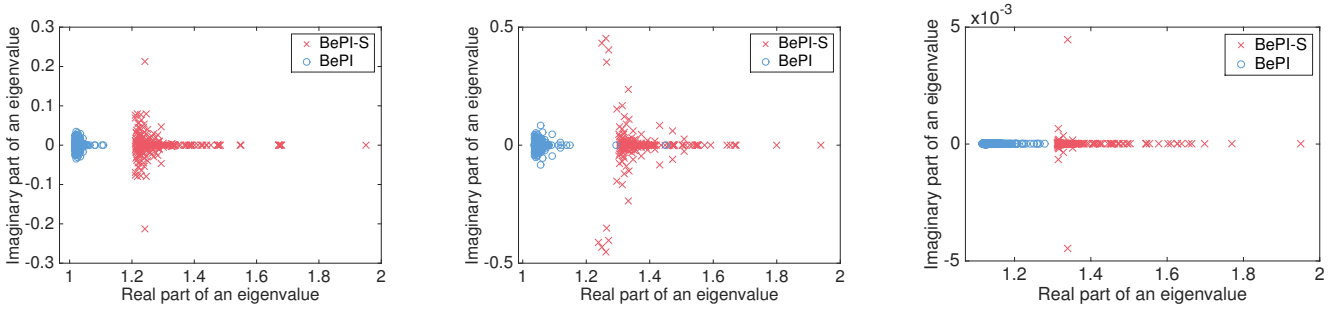
Table 3: The number of non-zeros of $\mathbf{S}$ computed by our methods. Note that the number of non-zeros of $\mathbf{S}$ decreases by the sparsification of the Schur complement. BePI-B runs out of time (more than 24 hours) when computing $\mathbf{S}$ for the Friendster dataset, while BePI-S and BePI successfully compute it.

dataset	A: (|S| in BePI-B)	B: (|S| in BePI or BePI-S)	ratio (A/B)
Slashdot	664,686	353,559	1.9×
Wikipedia	844,983	626,887	1.3×
Baidu	23,136,773	2,359,563	9.8×
Flickr	113,842,305	29,990,289	3.8×
LiveJournal	417,551,300	83,070,865	5.0×
WikiLink	555,468,477	377,197,963	1.5×
Twitter	8,494,161,448	1,640,399,051	5.2×
Friendster	o.o.t.	2,018,006,285	–

Table 4: The average number of iterations to compute $\mathbf{r}_2$ by BePI-S and BePI. After preconditioning, the number of iterations for solving the linear system of $\mathbf{S}$ decreases.

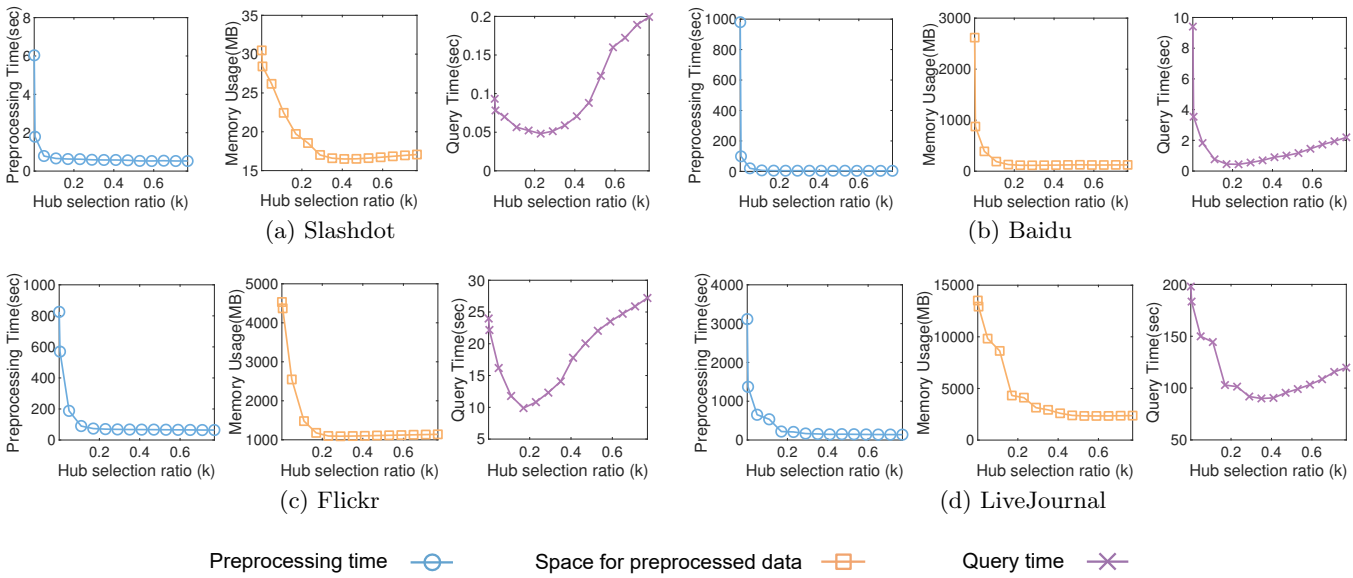
dataset	A: (# iterations in BePI-S)	B: (# iterations in BePI)	ratio (A/B)
Slashdot	43.2	6.6	6.5×
Wikipedia	52.4	13.1	4.0×
Baidu	42.6	14.9	2.9×
Flickr	44.2	11.3	3.9×
LiveJournal	49.1	16.2	3.0×
WikiLink	70.2	16.5	4.3×
Twitter	60.3	18.7	3.2×
Friendster	24.2	10.5	2.3×

Compared to BePI-S, BePI uses slightly more memory space as seen in Figure 6(b). In addition, the preprocessing phase of BePI takes slightly longer than that of BePI-S. The reason is that BePI computes the incomplete LU factors of $\mathbf{S}$, $\tilde{\mathbf{L}}_2$ and $\tilde{\mathbf{U}}_2$, in the preprocessing phase, while BePI-S does not. However, the gap between them is small in terms of the preprocessing time and the memory space; furthermore, BePI achieves faster query time thanks to the incomplete LU factors, which we describe in the following subsection.



(a) Slashdot (b) Wikipedia (c) Baidu

Figure 7: Distribution of the top-200 eigenvalues of the preconditioned Schur complement (blue o's) and the original Schur complement (red x's). X-axis and y-axis represent the real part and the imaginary part of an eigenvalue, respectively. Results from three different datasets, Slashdot, Wikipedia, and Baidu, show that the dispersion of eigenvalue distribution becomes much smaller when the Schur complement is preconditioned.



(a) Slashdot (b) Baidu (c) Flickr (d) LiveJournal

Figure 8: Effects of the hub selection ratio k in Algorithm 3. According to results, preprocessing time and memory usage of BePI decrease as k increases. Especially, when k is small (e.g., $k = 0.001$), preprocessing time and memory consumption are high. The query speed of BePI is the fastest when k is around $0.2 \sim 0.3$ as shown in the figures.

4.5.2 *Effects on Query Phase*

We investigate the effects of the sparsification of the Schur complement and the preconditioner on the query phase of our method. To evaluate the effects, we generate 30 random seeds, and measure the average query time using BePI, BePI-S, and BePI-B. Figure 6(c) compares these methods in terms of query time.

We first compare BePI-B and BePI-S to see the effect of the sparsification of the Schur complement. According to the result shown in Figure 6(c), BePI-S is up to 5× faster than BePI-B. This speedup is due to the reduction in the number of non-zeros of $\mathbf{S}$ by the sparsification of the Schur Complement as described in Table 2.

For analyzing the effect of preconditioning, we compare BePI-S and BePI. BePI is up to 4× faster than BePI-S as shown in Figure 6(c). Applying the preconditioner reduces the number of iterations for computing $\mathbf{r}_2$, as summarized in Table 4. This faster convergence is closely related to the tighter clustering of eigenvalues of the preconditioned Schur complement [37]. Figure 7 shows that the eigenvalues in BePI form a tight cluster, while those in BePI-S do not. In sum, BePI is up to 13× faster than BePI-B in the query phase, which indicates that the query cost is effectively reduced with the sparsification of the Schur complement and the preconditioner.

4.6 Effects of the Hub Selection Ratio

We investigate the effects of the hub selection ratio k (Algorithm 3) on the performance of our method BePI. We measure preprocessing time, memory space of preprocessed data, and query time of BePI varying k on the Slashdot, the Baidu, the Flickr, and the LiveJournal datasets. As shown in Figure 8, the performance of BePI in terms of preprocessing time and memory usage becomes improved as k increases. In particular, BePI requires high preprocessing time and memory space when k is very small (e.g., $k = 0.001$). In terms of query time, BePI shows the best performance when k is from 0.2 through 0.3 as presented in Figure 8. There are two reasons for these effects. First, if we set a large k in Algorithm 3, then the running time of the hub-and-spoke reordering method decreases because the number of iterations of the reordering method is reduced. Also, as described in Section 3.4 and Table 3, the number of non-zeros of the Schur complement decreases as k increases

from 0; thus, the memory usage is reduced. However, setting too large k is not good for query time because the number of non-zeros and the dimension of the Schur complement become large. As shown in Figure 8, when k is around 0.2, it provides a good trade-off between preprocessing time, memory usage, and query time.

5. RELATED WORKS

We review previous works on RWR from three perspectives: (1) relevance measures and applications, (2) approximate and top-k methods, and (3) preprocessing methods.

Relevance measures and applications of RWR. There are various node-to-node relevance measure methods based on link analysis and random walk in graphs: PageRank [32, 44], Personalized RageRank [5, 4, 15], SimRank [2, 25, 51], and Random Walk with Restart [14, 38, 49]. As a popular relevance measure, RWR has been used widely for many applications. Kim et al. [24] proposed an image segmentation algorithm that finds a generative model for each label by using RWR in measuring the proximity between a pixel and a seed. Andersen et al. [1] introduced a local graph partitioning algorithm that employed RWR to find a good cut with small conductance around a starting node. Gleich et al. [18] and Whang et al. [45] improved this algorithm in terms of seeding strategy. Wang et al. [43] proposed an image annotation method that generates candidate annotations, re-ranks them using RWR, and reserves the top ones. Pan et al. [34] proposed a general, RWR-based method to discover correlations across multimedia data. Sun et al. [39] proposed algorithms for neighborhood formulation and anomaly detection using RWR in bipartite graphs. By combining RWR and supervised learning, Backstrom et al. [3] presented a link prediction method that learns a random walk biasing function from the node and edge attributes. Zhu et al. [52] employed RWR in their transductive model for content-based image retrieval. Jung et al. [21] designed a personalized ranking model in signed networks based on the concept of RWR.

Approximate and top-k methods for RWR. As discussed in Section 2.2, iterative approaches often fail to scale up for real-world applications due to high computational cost. Several approximate methods have been developed to overcome this problem. Observing that the relevance scores are highly skewed, and real-world graphs often exhibit a block-wise structure, Sun et al. [39] proposed an approximate algorithm that performs RWR only on the partition containing the seed node, while setting the relevance score of other nodes outside the partition to 0. Building on similar observations, Tong et al. [41] proposed approximate algorithms, B_LIN and its derivatives, in which they applied a low-rank approximation to the cross-partition links using eigenvalue decomposition. Gleich et al. [17] proposed methods that apply RWR only to a part of the graph, which is determined adaptively in the query phase. Andersen et al. [1] presented an algorithm for local graph partitioning problem that computes PageRank vectors approximately. Fast-PPR, a Monte Carlo-based method proposed by Lofgren et al. [30], estimates the single pair PPR (Personalized PageRank) between a start node and a target node by employing a bi-directional scheme. Bahmani et al. [4] developed a fast MapReduce algorithm based on Monte Carlo simulation for approximating PPR scores. To compute PPR approximately, Xie et al. [47] used a model reduction approach where solutions are projected to a low dimensional space. Also, several works have been proposed to focus on the k most relevant nodes w.r.t. a seed node instead of calculating the RWR scores of every node. K-dash, a top-k method proposed by Fujiwara et al. [14], computes the RWR scores of top-k nodes by exploiting precomputed sparse matrices and pruning strategies. Wu et al. [46] proposed Fast Local Search (FLOS) which finds top-k relevant nodes in terms of various measures including RWR. However, approximate and top-k computation for RWR scores are insufficient for many data mining applications [24, 1, 34, 3, 52, 41, 48] which require accurate RWR scores for any pair of nodes, whereas BePI calculates the RWR scores of all nodes exactly.

Preprocessing methods for RWR. The query speed of RWR can be accelerated significantly by precomputing $\mathbf{H}^{-1}$, as discussed in Section 2.3. However, matrix inversion does not scale up for large graphs, as it involves a dense matrix that is too large to fit in memory. To tackle this problem, alternative preprocessing methods have been developed. Tong et al. [41] proposed NB_LIN, which decomposes the adjacency matrix using a low-rank approximation in the preprocessing phase, and approximates $\mathbf{H}^{-1}$ from the decomposed matrices in the query phase. Fujiwara et al. applied LU decomposition [14] and QR decomposition [16] to the adjacency matrix to obtain sparser matrices to use in place of $\mathbf{H}^{-1}$. Prior to applying LU decomposition [14], they reordered $\mathbf{H}$ based on the degree of nodes and the community structure to make $\mathbf{L}^{-1}$ and $\mathbf{U}^{-1}$ sparse. Bear [38, 22] preprocesses the adjacency matrix by exploiting node reordering and block elimination techniques. While all of these methods made performance improvements over previous approaches, they suffer from the scalability problem when it comes to billion-scale graphs.

In addition to the techniques described above, computing relevance scores on dynamic graphs is also an interesting topic [5, 22, 6]. A conventional strategy for preprocessing methods on dynamic graphs is batch update, e.g., it stores update information such as edge insertions for one day, and re-preprocesses the changed graph at midnight. Note that our method is desirable for this case since our method is efficient in terms of preprocessing time. Another approach is to incrementally update preprocessed data as suggested in [22].

6. CONCLUSION

In this paper, we propose BePI, a fast, memory-efficient, and scalable algorithm for random walk with restart computation on billion-scale graphs. BePI takes the advantages of both preprocessing methods and iterative methods by incorporating an iterative method within a block elimination approach. Furthermore, BePI improves the performance by decreasing the number of non-zeros of a matrix and applying a preconditioner. Consequently, BePI achieves a better scalability as well as faster query time than existing methods. We give theoretical analysis on the accuracy and complexities of BePI. Also, we experimentally show that BePI processes up to 100× larger graph, and requires up to 130× less memory space than other preprocessing methods. In the query phase, BePI computes RWR scores 9× faster than other existing methods in large graphs which other preprocessing methods fail to process, due to running out of memory or time. Future research directions include extending BePI to a distributed environment.

Acknowledgment

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Science, ICT and Future Planning (grant No. 2015R1C1A2A01055739 and 2015K1A3A1A14021055). U Kang is the corresponding author.

7. REFERENCES

[1] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *FOCS*, pages 475–486, 2006.

[2] I. Antonellis, H. G. Molina, and C. C. Chang. Simrank++: Query rewriting through link analysis of the click graph. *PVLDB*, 1(1):408–421, 2008.

[3] L. Backstrom and J. Leskovec. Supervised random walks: Predicting and recommending links in social networks. In *WSDM*, pages 635–644, 2011.

[4] B. Bahmani, K. Chakrabarti, and D. Xin. Fast personalized pagerank on mapreduce. In *SIGMOD*, pages 973–984. ACM, 2011.

[5] B. Bahmani, A. Chowdhury, and A. Goel. Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment*, 4(3):173–184, 2010.

[6] B. Bahmani, R. Kumar, M. Mahdian, and E. Upfal. Pagerank on an evolving graph. pages 24–32, 2012.

[7] M. Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of computational Physics*, 182(2):418–477, 2002.

[8] M. Benzi, C. D. Meyer, and M. Tuma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM Journal on Scientific Computing*, 17(5):1135–1149, 1996.

[9] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2009.

[10] S. Chakrabarti, A. Pathak, and M. Gupta. Index design and query processing for graph conductance search. *PVLDB*, 20(3):445–470, 2011.

[11] J. W. Demmel. *Applied numerical linear algebra*. Siam, 1997.

[12] I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software (TOMS)*, 15(1):1–14, 1989.

[13] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, volume 29, pages 251–262. ACM, 1999.

[14] Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa. Fast and exact top-k search for random walk with restart. *PVLDB*, 5(5):442–453, 2012.

[15] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, T. Mishima, and M. Onizuka. Efficient ad-hoc search for personalized pagerank. In *SIGMOD*, pages 445–456. ACM, 2013.

[16] Y. Fujiwara, M. Nakatsuji, T. Yamamuro, H. Shiokawa, and M. Onizuka. Efficient personalized pagerank with accuracy assurance. In *KDD*, pages 15–23, 2012.

[17] D. Gleich and M. Polito. Approximating personalized pagerank with minimal use of web graph data. *Internet Mathematics*, 3(3):257–294, 2006.

[18] D. F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *KDD*, pages 597–605, 2012.

[19] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[20] J. He, M. Li, H.-J. Zhang, H. Tong, and C. Zhang. Manifold-ranking based image retrieval. In *ACM Multimedia*, pages 9–16, 2004.

[21] J. Jung, W. Jin, L. Sael, and U. Kang. Personalized ranking in signed networks using signed random walk with restart. In *ICDM*, 2016.

[22] J. Jung, K. Shin, L. Sael, and U. Kang. Random walk with restart on large graphs using block elimination. *ACM Transactions on Database Systems*.

[23] U. Kang and C. Faloutsos. Beyond 'caveman communities': Hubs and spokes for graph compression and mining. In *ICDM*, pages 300–309, 2011.

[24] T. H. Kim, K. M. Lee, and S. U. Lee. Generative image segmentation using random walks with restart. In *Computer Vision - ECCV 2008, 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part III*, pages 264–275, 2008.

[25] M. Kusumoto, T. Maehara, and K.-i. Kawarabayashi. Scalable similarity search for simrank. In *SIGMOD*, pages 325–336. ACM, 2014.

[26] A. N. Langville and C. D. Meyer. A reordering for the pagerank problem. *SIAM Journal on Scientific Computing*, 27(6):2112–2120, 2006.

[27] A. N. Langville and C. D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2011.

[28] S. Lee, S.-i. Song, M. Kahng, D. Lee, and S.-g. Lee. Random walk based entity ranking on graph for multidimensional recommendation. In *RecSys*, pages 93–100. ACM, 2011.

[29] Y. Lim, U. Kang, and C. Faloutsos. Slashburn: Graph compression and mining beyond caveman communities. *IEEE Trans. Knowl. Data Eng.*, 26(12):3077–3089, 2014.

[30] P. A. Lofgren, S. Banerjee, A. Goel, and C. Seshadhri. Fast-ppr: Scaling personalized pagerank estimation for large graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1436–1445. ACM, 2014.

[31] C. D. Meyer. *Matrix analysis and applied linear algebra*. Siam, 2000.

[32] I. Mitliagkas, M. Borokhovich, A. G. Dimakis, and C. Caramanis. Frogwild!: fast pagerank approximations on graph engines. *Proceedings of the VLDB Endowment*, 8(8):874–885, 2015.

[33] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. *Technical report, Stanford University*, 1999.

[34] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, pages 653–658, 2004.

[35] Y. Saad. A flexible inner-outer preconditioned gmres algorithm. *SIAM Journal on Scientific Computing*, 14(2):461–469, 1993.

[36] Y. Saad. *Iterative methods for sparse linear systems*. Siam, 2003.

[37] Y. Saad and M. H. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[38] K. Shin, J. Jung, L. Sael, and U. Kang. Bear: Block elimination approach for random walk with restart on large graphs. In *SIGMOD*, 2015.

[39] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*, pages 418–425, 2005.

[40] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *KDD*, pages 404–413, 2006.

[41] H. Tong, C. Faloutsos, and J.-Y. Pan. Random walk with restart: fast solutions and applications. *KAIS*, 14(3):327–346, 2008.

[42] L. N. Trefethen and D. Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

[43] C. Wang, F. Jing, L. Zhang, and H. Zhang. Image annotation refinement using random walk with restarts. In *Proceedings of the 14th ACM International Conference on Multimedia, Santa Barbara, CA, USA, October 23-27, 2006*, pages 647–650, 2006.

[44] Y. Wang and D. J. DeWitt. Computing pagerank in a distributed internet search system. In *Proceedings of the*

Thirtieth international conference on Very large data bases-Volume 30, pages 420–431. VLDB Endowment, 2004.

[45] J. J. Whang, D. F. Gleich, and I. S. Dhillon. Overlapping community detection using seed set expansion. In *CIKM*, pages 2099–2108, 2013.

[46] Y. Wu, R. Jin, and X. Zhang. Fast and unified local search for random walk based k-nearest-neighbor query in large graphs. In *SIGMOD*, pages 1139–1150, 2014.

[47] W. Xie, D. Bindel, A. Demers, and J. Gehrke. Edge-weighted personalized pagerank: Breaking a decade-old performance barrier. In *KDD*, pages 1325–1334. ACM, 2015.

[48] Z. Yin, M. Gupta, T. Weninger, and J. Han. A unified framework for link recommendation using random walks. In *ASONAM*, pages 152–159. IEEE, 2010.

[49] A. W. Yu, N. Mamoulis, and H. Su. Reverse top-k search using random walk with restart. *Proceedings of the VLDB Endowment*, 7(5):401–412, 2014.

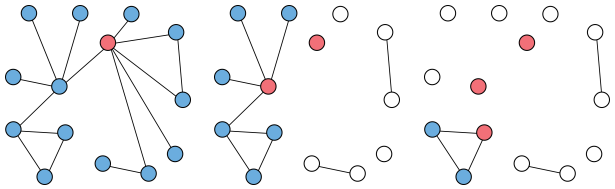
[50] F. Zhang. *The Schur complement and its applications*, volume 4. Springer Science & Business Media, 2006.

[51] W. Zheng, L. Zou, Y. Feng, L. Chen, and D. Zhao. Efficient simrank-based similarity join over large graphs. *Proceedings of the VLDB Endowment*, 6(7):493–504, 2013.

[52] S. Zhu, L. Zou, and B. Fang. Content based image retrieval via a transductive model. *J. Intell. Inf. Syst.*, 42(1), 2014.

APPENDIX

A. DETAILS OF SLASHBURN



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

Figure 9: Hub selection in SlashBurn when $\lceil kn \rceil = 1$ where $\lceil kn \rceil$ indicates the number of selected hubs. Red nodes are hubs; white nodes are spokes that belong to the disconnected components; blue colored are nodes that belong to the giant connected component after Step 1.

SlashBurn [23, 29] is a graph algorithm to concentrate the non-zeros of the adjacency matrix by node reordering. Let n be the number of nodes in a graph, and k be the hub selection ratio whose range is between 0 and 1 where $\lceil kn \rceil$ indicates the number of nodes selected by SlashBurn as hubs per iteration. SlashBurn disconnects $\lceil kn \rceil$ *hub nodes* (high degree nodes) from the graph, and divides the graph into the giant connected component (GCC) and the disconnected components. The nodes in the disconnected components are called *spokes*. Then, SlashBurn reorders nodes such that the hub nodes get the highest ids, the spokes get the lowest ids, and the nodes in the GCC get the ids in the middle. SlashBurn repeats this procedure on the GCC recursively until the size of GCC becomes smaller than $\lceil kn \rceil$. After SlashBurn is done, the reordered adjacency matrix contains a large and sparse block diagonal matrix in the upper left area, as shown in Figure 3(c). Figure 9 illustrates the operation of SlashBurn when $\lceil kn \rceil = 1$.

B. DETAILS OF PRECONDITIONED GMRES

Preconditioned GMRES [35, 42] computes the solution $\mathbf{r}_2$ of the preconditioned linear system in Equation (11), $\mathbf{M}^{-1}\mathbf{S}\mathbf{r}_2 = \mathbf{M}^{-1}\tilde{\mathbf{q}}_2$, where $\mathbf{S}$ is incomplete LU decomposed into $\tilde{\mathbf{L}}_2$ and $\tilde{\mathbf{U}}_2$, and $\mathbf{M}^{-1} = \tilde{\mathbf{U}}_2^{-1}\tilde{\mathbf{L}}_2^{-1}$ is a preconditioner. Algorithm 5 describes the procedure of preconditioned GMRES. It iteratively finds $\mathbf{r}_2^{(i)}$ which minimizes the residual $\|\mathbf{M}^{-1}(\mathbf{S}\mathbf{r}_2^{(i)} - \tilde{\mathbf{q}}_2)\|_2$ such that $\mathbf{r}_2^{(i)}$ is in the i-th order preconditioned Krylov subspace $\mathcal{K}_i$ represented as follows:

$$\mathcal{K}_i = \{\mathbf{M}^{-1}\tilde{\mathbf{q}}_2, (\mathbf{M}^{-1}\mathbf{S})\mathbf{M}^{-1}\tilde{\mathbf{q}}_2, ..., (\mathbf{M}^{-1}\mathbf{S})^{i-1}\mathbf{M}^{-1}\tilde{\mathbf{q}}_2\}.$$

Since the vectors consisting of the Krylov subspace $\mathcal{K}_i$ are almost linearly dependent, directly finding the solution using the vectors as basis could be unstable. Instead, preconditioned GMRES finds $\mathbf{r}_2^{(i)}$ in a subspace generated by the orthonormal vectors $\{\mathbf{p}_1, ..., \mathbf{p}_i\}$ in $\mathcal{K}_i$, which are iteratively computed by Arnoldi Iteration (lines 4∼10). The solution $\mathbf{r}_2^{(i)}$ is $\mathbf{P}_i\mathbf{y}$ where $\mathbf{P}_i = [\mathbf{p}_1, ..., \mathbf{p}_i]$ is an orthonormal matrix and $\mathbf{y} \in \mathbb{R}^n$. Then, the partial similarity transformation of $\mathbf{M}^{-1}\mathbf{S}$ by Arnoldi Iteration at the i-th iteration is represented as follows:

$$(\mathbf{M}^{-1}\mathbf{S})\mathbf{P}_i = \mathbf{P}_{i+1}\mathbf{H}_i$$

where $\mathbf{H}_i$ is a Hessenberg matrix in $\mathbb{R}^{(i+1)\times i}$ and $h_{j,k}$ is the (j,k)-th entry of $\mathbf{H}_i$. Since the columns of $\mathbf{P}_i$ are orthonormal (i.e., $\mathbf{P^T}_i\mathbf{P}_i = \mathbf{I}$) and $\mathbf{r}_2^{(i)} = \mathbf{P}_i\mathbf{y}$, the original residual is modified as follows:

$$\|\mathbf{M}^{-1}(\mathbf{S}\mathbf{r}_2^{(i)} - \tilde{\mathbf{q}}_2)\|_2 \Rightarrow \|\mathbf{P}_{i+1}\mathbf{H}_i\mathbf{y} - \mathbf{M}^{-1}\tilde{\mathbf{q}}_2\|_2$$
$$\Rightarrow \|\mathbf{H}_i\mathbf{y} - \mathbf{P}_{i+1}^T\mathbf{M}^{-1}\tilde{\mathbf{q}}_2\|_2 \Rightarrow \|\mathbf{H}_i\mathbf{y} - \|\mathbf{M}^{-1}\tilde{\mathbf{q}}_2\|_2\mathbf{e}_1\|_2$$

Note that we exploit $\mathbf{p}_1 = (\mathbf{M}^{-1}\tilde{\mathbf{q}}_2)/\|\mathbf{M}^{-1}\tilde{\mathbf{q}}_2\|_2$ (lines 1∼2) for the last transformation. Consequently, preconditioned GMRES finds $\mathbf{y}^*$ which minimizes the modified residual $\|\mathbf{H}_i\mathbf{y} - \|\mathbf{M}^{-1}\tilde{\mathbf{q}}_2\|_2\mathbf{e}_1\|_2$ using a linear least square method such as QR decomposition (line 11); and then, it computes the solution $\mathbf{r}_2^{(i)}$ based on $\mathbf{y}^*$ (line 12). Preconditioned GMRES repeats the procedure for finding $\mathbf{r}_2^{(i)}$ until the residual is less than ϵ (line 13).

Note that we do not need to obtain $\mathbf{M}^{-1} = \tilde{\mathbf{U}}_2^{-1}\tilde{\mathbf{L}}_2^{-1}$ if

Algorithm 5: Preconditioned GMRES [35, 42]

Input: matrix: $\mathbf{S}$, vector: $\tilde{\mathbf{q}}_2$, preconditioner: $\tilde{\mathbf{L}}_2$ and $\tilde{\mathbf{U}}_2$, error tolerance: ϵ
Output: solution: $\mathbf{r}_2$ of the preconditioned system in Equation (11)

1: $\mathbf{t} = \tilde{\mathbf{U}}_2^{-1}(\tilde{\mathbf{L}}_2^{-1}\tilde{\mathbf{q}}_2) \Leftrightarrow \tilde{\mathbf{U}}_2 \backslash_B (\tilde{\mathbf{L}}_2 \backslash_F \tilde{\mathbf{q}}_2)$
2: $\mathbf{p}_1 = \mathbf{t}/\|\mathbf{t}\|_2$
3: **for** i=1:n_2 **do**
find an orthonormal vector $\mathbf{p}_{i+1}$ against $\{\mathbf{p}_1, ..., \mathbf{p}_i\}$ in $\mathcal{K}_{i+1}$ using the following iteration (lines 4∼10), called Arnoldi Iteration [42]
4: $\mathbf{v} = \tilde{\mathbf{U}}_2^{-1}(\tilde{\mathbf{L}}_2^{-1}(\mathbf{S}\mathbf{p}_i)) \Leftrightarrow \tilde{\mathbf{U}}_2 \backslash_B (\tilde{\mathbf{L}}_2 \backslash_F (\mathbf{S}\mathbf{p}_i))$
5: **for** j=1:i **do**
6: $h_{j,i} = \mathbf{p}_j^T\mathbf{v}$
7: $\mathbf{v} = \mathbf{v} - h_{j,i}\mathbf{p}_j$
8: **end for**
9: $h_{i+1,i} = \|\mathbf{v}\|_2$
10: $\mathbf{p}_{i+1} = \mathbf{v}/h_{i+1,i}$
11: solve $\mathbf{y}^* \leftarrow \text{argmin}_{\mathbf{y}} \|\mathbf{H}_i\mathbf{y} - \|\mathbf{t}\|_2\mathbf{e}_1\|_2$ using a linear least square method (e.g., QR decomposition)
12: $\mathbf{r}_2^{(i)} = \mathbf{P}_i\mathbf{y}^*$
13: **if** $\|\mathbf{H}_i\mathbf{y}^* - \|\mathbf{t}\|_2\mathbf{e}_1\|_2 < \epsilon$ **then**
14: $\mathbf{r}_2 \leftarrow \mathbf{r}_2^{(i)}$; break
15: **end if**
16: **end for**
17: **return** $\mathbf{r}_2$

$\mathbf{M}$ consists of triangular matrices. For example, when we should perform an operation in the form of $\mathbf{z} = \tilde{\mathbf{U}}_2^{-1}(\tilde{\mathbf{L}}_2^{-1}\mathbf{w})$ (lines 1 and 4), forward and backward substitutions efficiently compute $\mathbf{z}$ without matrix inversion [11], i.e., $\mathbf{z} = \tilde{\mathbf{U}}_2 \backslash_B (\tilde{\mathbf{L}}_2 \backslash_F \mathbf{w})$ where $\backslash_F$ and $\backslash_B$ are defined as follows:

- Forward substitution $\backslash_F$: $\mathbf{x} = \mathbf{L}^{-1}\mathbf{b} \Leftrightarrow \mathbf{x} = \mathbf{L}\backslash_F \mathbf{b}$ where $\mathbf{L}$ is a lower triangular matrix.
- Backward substitution $\backslash_B$: $\mathbf{x} = \mathbf{U}^{-1}\mathbf{b} \Leftrightarrow \mathbf{x} = \mathbf{U}\backslash_B \mathbf{b}$ where $\mathbf{U}$ is an upper triangular matrix.

The time complexity of the substitution algorithms is the same as that of matrix-vector multiplication [11]. Hence, preconditioned GMRES efficiently finds $\mathbf{r}_2$ of the preconditioned system without inverting $\tilde{\mathbf{L}}_2$ and $\tilde{\mathbf{U}}_2$.

C. PROOF OF INVERSE INEQUALITY

LEMMA 5. *For a linear system $\mathbf{Ax} = \mathbf{b}$, if a matrix $\mathbf{A}$ is invertible, then $||\mathbf{A}^{-1}||_2^{-1}||\mathbf{x}||_2 \leq ||\mathbf{Ax}||_2$.*

PROOF. Since $\mathbf{A}$ is invertible, $\|\mathbf{x}\|_2$ is bounded as follows:

$$\|\mathbf{x}\|_2 = \|\mathbf{A}^{-1}\mathbf{Ax}\|_2 \leq \|\mathbf{A}^{-1}\|_2 \|\mathbf{A}x\|_2.$$

Hence, $||\mathbf{A}^{-1}||_2^{-1}||\mathbf{x}||_2 \leq ||\mathbf{Ax}||_2$. □

D. PROOF OF LEMMA 1

PROOF. The partitioned linear system in Equation (6) is represented using Equations (3) and (4) as follows:

$$\begin{bmatrix}\mathbf{H}_{11} & \mathbf{H}_{12}\\ \mathbf{H}_{21} & \mathbf{H}_{22}\end{bmatrix}\begin{bmatrix}\mathbf{r}_1\\ \mathbf{r}_2\end{bmatrix} = c\begin{bmatrix}\mathbf{q}_1\\ \mathbf{q}_2\end{bmatrix} \tag{12}$$

$$\mathbf{r}_3 = c\mathbf{q}_3 - \mathbf{H}_{31}\mathbf{r}_1 - \mathbf{H}_{32}\mathbf{r}_2$$

Equation (12) is split into two equations:

$$\mathbf{H}_{11}\mathbf{r}_1 + \mathbf{H}_{12}\mathbf{r}_2 = c\mathbf{q}_1 \tag{13}$$

$$\mathbf{H}_{21}\mathbf{r}_1 + \mathbf{H}_{22}\mathbf{r}_2 = c\mathbf{q}_2 \tag{14}$$

Then, $\mathbf{r}_1$ is obtained from Equation (13) as follows:

$$\begin{aligned}&\mathbf{H}_{11}\mathbf{r}_1 + \mathbf{H}_{12}\mathbf{r}_2 = c\mathbf{q}_1\\ &\Rightarrow \mathbf{H}_{11}\mathbf{r}_1 = c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2\\ &\Rightarrow \mathbf{r}_1 = \mathbf{H}_{11}^{-1}(c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2).\end{aligned}$$

If we plug the above equation of $\mathbf{r}_1$ into Equation (14), then it is represented as follows:

$$\begin{aligned}&\mathbf{H}_{21}\mathbf{r}_1 + \mathbf{H}_{22}\mathbf{r}_2 = c\mathbf{q}_2\\ &\Rightarrow \mathbf{H}_{21}(\mathbf{H}_{11}^{-1}(c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2)) + \mathbf{H}_{22}\mathbf{r}_2 = c\mathbf{q}_2\\ &\Rightarrow \mathbf{H}_{21}(\mathbf{H}_{11}^{-1}(c\mathbf{q}_1)) + (\mathbf{H}_{22} - \mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12})\mathbf{r}_2 = c\mathbf{q}_2\\ &\Rightarrow \mathbf{S}\mathbf{r}_2 = c\mathbf{q}_2 - \mathbf{H}_{21}(\mathbf{H}_{11}^{-1}(c\mathbf{q}_1))\\ &\Rightarrow \mathbf{r}_2 = \mathbf{S}^{-1}(c\mathbf{q}_2 - \mathbf{H}_{21}(\mathbf{H}_{11}^{-1}(c\mathbf{q}_1)))\end{aligned}$$

where $\mathbf{S} = \mathbf{H}_{22} - \mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12}$. Note that $\mathbf{H}$ and $\mathbf{H}_{11}$ is invertible if $0 < c < 1$ since they are strictly diagonally dominant; $\mathbf{S}$ is invertible because $\mathbf{H}$ is invertible [50]. □

E. PROOF OF LEMMA 2

PROOF. Since we use GMRES to solve the linear system of $\mathbf{S}$, we first analyze the accuracy bound of GMRES. Since GMRES stops the iteration when the relative residual $\frac{||\mathbf{S}\mathbf{r}_2^{(k)} - \tilde{\mathbf{q}}_2||_2}{||\tilde{\mathbf{q}}_2||_2} \leq \epsilon$, the inequality is written as follows:

$$\begin{aligned}&||\mathbf{S}\mathbf{r}_2^{(k)} - \tilde{\mathbf{q}}_2||_2 \leq \epsilon||\tilde{\mathbf{q}}_2||_2\\ &\Rightarrow ||\mathbf{S}\mathbf{r}_2^{(k)} - \mathbf{S}\mathbf{r}_2^*||_2 \leq \epsilon||\tilde{\mathbf{q}}_2||_2\\ &\Rightarrow ||\mathbf{S}(\mathbf{r}_2^* - \mathbf{r}_2^{(k)})||_2 \leq \epsilon||\tilde{\mathbf{q}}_2||_2\end{aligned}$$

Note that $\mathbf{H}$ and $\mathbf{H}_{11}$ are invertible because those matrices are diagonally dominant; this fact implies that $\mathbf{S}$ is invertible [50], and we are able to apply Lemma 5 to the last equation as follows:

$$\begin{aligned}&||\mathbf{S}^{-1}||_2^{-1}||\mathbf{r}_2^* - \mathbf{r}_2^{(k)}||_2 \leq ||\mathbf{S}(\mathbf{r}_2^* - \mathbf{r}_2^{(k)})||_2 \leq \epsilon||\tilde{\mathbf{q}}_2||_2\\ &\Rightarrow ||\mathbf{S}^{-1}||_2^{-1}||\mathbf{r}_2^* - \mathbf{r}_2^{(k)}||_2 \leq \epsilon||\tilde{\mathbf{q}}_2||_2\\ &\Rightarrow ||\mathbf{r}_2^* - \mathbf{r}_2^{(k)}||_2 \leq \epsilon||\mathbf{S}^{-1}||_2||\tilde{\mathbf{q}}_2||_2\end{aligned}$$

Since $||\mathbf{S}^{-1}||_2 = \sigma_{min}(\mathbf{S})^{-1}$ [31], $||\mathbf{r}_2^{(k)} - \mathbf{r}_2^*||_2$ is bounded as follows:

$$||\mathbf{r}_2^* - \mathbf{r}_2^{(k)}||_2 \leq \frac{||\tilde{\mathbf{q}}_2||_2}{\sigma_{min}(\mathbf{S})}\epsilon$$

where $\sigma_{min}(\mathbf{S})$ is the smallest singular value of $\mathbf{S}$. □

F. PROOF OF LEMMA 3

PROOF. Since $\mathbf{H}_{11}\mathbf{r}_1^* = c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2^*$ and $\mathbf{H}_{11}\mathbf{r}_1^{(k)} = c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2^{(k)}$, $||\mathbf{H}_{11}\mathbf{r}_1^* - \mathbf{H}_{11}\mathbf{r}_1^{(k)}||_2$ is represented as follows:

$$\begin{aligned}||\mathbf{H}_{11}\mathbf{r}_1^* - \mathbf{H}_{11}\mathbf{r}_1^{(k)}||_2 &= ||c\mathbf{q}_1 - \mathbf{H}_{12}\mathbf{r}_2^* - c\mathbf{q}_1 + \mathbf{H}_{12}\mathbf{r}_2^{(k)}||_2\\ &= ||\mathbf{H}_{12}\mathbf{r}_2^* - \mathbf{H}_{12}\mathbf{r}_2^{(k)}||_2\\ &= ||\mathbf{H}_{12}(\mathbf{r}_2^* - \mathbf{r}_2^{(k)})||_2\end{aligned}$$

Since L2-norm is a sub-multiplicative norm [31], $||\mathbf{H}_{12}(\mathbf{r}_2^* - \mathbf{r}_2^{(k)})||_2 \leq ||\mathbf{H}_{12}||_2||\mathbf{r}_2^* - \mathbf{r}_2^{(k)}||_2$. Hence $||\mathbf{H}_{11}\mathbf{r}_1^* - \mathbf{H}_{11}\mathbf{r}_1^{(k)}||_2$ is bounded as follows:

$$||\mathbf{H}_{11}\mathbf{r}_1^* - \mathbf{H}_{11}\mathbf{r}_1^{(k)}||_2 \leq ||\mathbf{H}_{12}||_2||\mathbf{r}_2^* - \mathbf{r}_2^{(k)}||_2$$

By Lemma 5, $(||\mathbf{H}_{11}^{-1}||_2)^{-1}||\mathbf{r}_1^* - \mathbf{r}_1^{(k)}||_2 \leq ||\mathbf{H}_{11}(\mathbf{r}_1^* - \mathbf{r}_1^{(k)})||_2$. Hence, $||\mathbf{r}_1^* - \mathbf{r}_1^{(k)}||_2$ is bounded as follows:

$$||\mathbf{r}_1^* - \mathbf{r}_1^{(k)}||_2 \leq ||\mathbf{H}_{11}^{-1}||_2||\mathbf{H}_{12}||_2||\mathbf{r}_2^* - \mathbf{r}_2^{(k)}||_2.$$

By Lemma 2, $||\mathbf{r}_2^* - \mathbf{r}_2^{(k)}||_2 \leq \frac{||\tilde{\mathbf{q}}_2||_2}{\sigma_{min}(\mathbf{S})}\epsilon$, and $||\mathbf{H}_{11}^{-1}||_2 = \sigma_{min}(\mathbf{H}_{11})^{-1}$ [31]. Therefore, $||\mathbf{r}_1^* - \mathbf{r}_1^{(k)}||_2$ is bounded as follows:

$$||\mathbf{r}_1^* - \mathbf{r}_1^{(k)}||_2 \leq \frac{||\mathbf{H}_{12}||_2||\tilde{\mathbf{q}}_2||_2}{\sigma_{min}(\mathbf{H}_{11})\sigma_{min}(\mathbf{S})}\epsilon.$$ □

G. PROOF OF LEMMA 4

PROOF. From the triangular inequality and the submultiplicative property of L2-norm, it is represented as follows:

$$\begin{aligned}\|\mathbf{r}_3^* - \mathbf{r}_3^{(k)}\|_2 &= \|-\mathbf{H}_{31}\mathbf{r}_1^* - \mathbf{H}_{32}\mathbf{r}_2^* + \mathbf{H}_{31}\mathbf{r}_1^{(k)} + \mathbf{H}_{32}\mathbf{r}_2^{(k)}\|_2\\ &= \|\mathbf{H}_{31}(\mathbf{r}_1^* - \mathbf{r}_1^{(k)}) + \mathbf{H}_{32}(\mathbf{r}_2^* - \mathbf{r}_2^{(k)})\|_2\\ &\leq \|\mathbf{H}_{31}(\mathbf{r}_1^* - \mathbf{r}_1^{(k)})\|_2 + \|\mathbf{H}_{32}(\mathbf{r}_2^* - \mathbf{r}_2^{(k)})\|_2\\ &\leq \|\mathbf{H}_{31}\|_2\|\mathbf{r}_1^* - \mathbf{r}_1^{(k)}\|_2 + \|\mathbf{H}_{32}\|_2\|\mathbf{r}_2^* - \mathbf{r}_2^{(k)}\|_2\end{aligned}$$ □

H. EXPERIMENTAL DATASETS

We give a brief description of the real-world datasets used for experiments in Section 4.

- **Slashdot**[1]**.** This is the social network of users in the technology news site Slashdot.
- **Wikipedia**[2]**.** This is the small network between articles of the English Wikipedia.
- **Baidu**[3]**.** This is the hyperlink network between articles of the Chinese online encyclopedia Baidu.
- **Flickr**[4]**.** This is the friendship network of Flickr users.
- **LiveJournal**[5]**.** Nodes are users of LiveJournal, and directed edges represent friendships.
- **WikiLink**[6]**.** This network consists of the wiki-links of the English Wikipedia.
- **Twitter**[7]**.** This is the follower network from Twitter, containing 1.4 billion directed follow edges between 41 million Twitter users.
- **Friendster**[8]**.** This is the friendship network of the online social site Friendster.

I. EXPERIMENTS ON ACCURACY

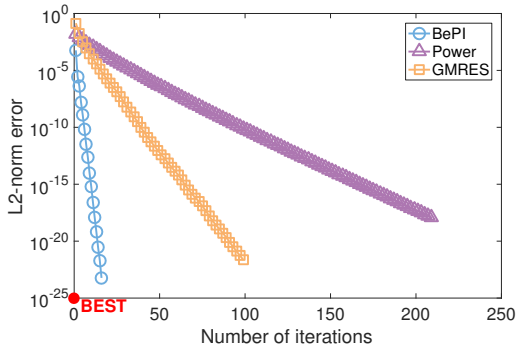


Figure 10: Accuracy of BePI according to the number of iterations. BePI achieves the highest accuracy and the fastest convergence compared to other iterative methods.

We investigate the accuracy of each iterative method compared to exact RWR solutions $\mathbf{r}^* = c\mathbf{H}^{-1}\mathbf{q}$. We perform this experiment on a small social network, the Physicians dataset [9], with 241 nodes and 1,098 edges in order to compute $\mathbf{H}^{-1}$. We select 100 seed nodes randomly, and measure average L2-norm errors between exact RWR solutions $\mathbf{r}^*$ and results $\mathbf{r}^{(i)}$ from each method with $\epsilon = 10^{-9}$ after i-th iterations (i.e., the errors are measured by computing $\|\mathbf{r}^* - \mathbf{r}^{(i)}\|_2$). As seen in Figure 10, our method BePI shows the best performance in terms of accuracy compared to other iterative methods. Furthermore, BePI converges rapidly with higher accuracy, while power iteration and GMRES converge slowly. Note that BePI is an exact method which can make the error smaller than any given error tolerance. As shown in Figure 10, the error of our method monotonically decreases and finally becomes smaller than the given error tolerance, which is also the property of the iterative method that we exploit [37].

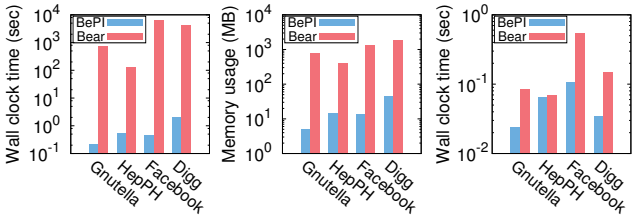
[1] http://dai-labor.de/IRML/datasets
[2] http://konect.uni-koblenz.de/networks/link-dynamic-simplewiki
[3] http://zhishi.me
[4] http://socialnetworks.mpi-sws.org/data-wosn2008.html
[5] http://snap.stanford.edu/data/soc-LiveJournal1.html
[6] http://dumps.wikimedia.org/
[7] http://an.kaist.ac.kr/traces/WWW2010.html
[8] https://archive.org/details/friendster-dataset-201107
[9] http://moreno.ss.uci.edu/data.html#ckm

Table 5: Dataset statistics used in Appendix J.

Dataset	Node	Edge	Description
Gnutella[1]	62,586	147,892	Peer-to-peer network
HepPH[1]	34,546	421,578	Coauthorship network
Facebook[1]	46,952	876,993	Social network
Digg[1]	279,630	1,731,653	Social network

[1] http://konect.uni-koblenz.de/

J. DETAILED COMPARISON WITH THE-STATE-OF-THE-ART METHOD



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

Figure 11: Our method BePI significantly outperforms Bear, the state-of-the-art preprocessing method [38], in terms of preprocessing time and memory usage as shown in (a) and (b), and shows faster query speed as in (c).

We compare our proposed method with Bear, the-state-of-the-art preprocessing method [38]. Since Bear suffers from the scalability issue in very large graphs as described in Section 4, we perform this experiment on relatively small graphs that Bear performs the preprocessing phase successfully. The datasets used in this experiments are summarized in Table 5. As shown in Figure 11, BePI significantly outperforms Bear in terms of preprocessing time, memory usage, and query time.

K. EXPERIMENT ON TOTAL TIME

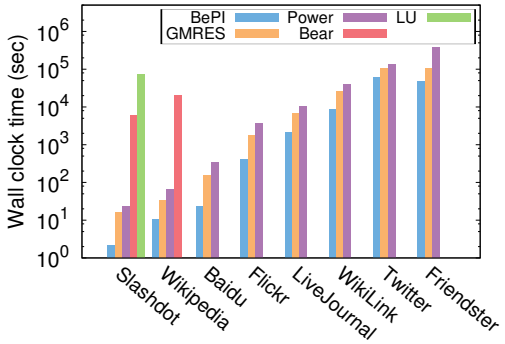


Figure 12: Total running time comparison. For preprocessing methods, preprocessing time and query time are included into total running time. For iterative methods, only query time is included. As presented in the figure, the total time of BePI is smaller than those of other methods.

We measure the total running time for each method. For preprocessing methods BePI, Bear, and LU, we consider the preprocessing time and the query time of 30 queries as the total running time. For iterative methods GMRES and power iteration, we only consider the query time for 30 queries since they do not involve preprocessing steps. As shown in Figure 12, our method BePI provides the best performance among other competitors in terms of total time.