BEAR: Block Elimination Approach for Random Walk with Restart on Large Graphs

Kijung Shin
Seoul National University
koreaskj@snu.ac.kr

Jinhong Jung
KAIST
montecast@kaist.ac.kr

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

U Kang
KAIST
ukang@cs.kaist.ac.kr

ABSTRACT

Given a large graph, how can we calculate the relevance between nodes fast and accurately? Random walk with restart (RWR) provides a good measure for this purpose and has been applied to diverse data mining applications including ranking, community detection, link prediction, and anomaly detection. Since calculating RWR from scratch takes long, various preprocessing methods, most of which are related to inverting adjacency matrices, have been proposed to speed up the calculation. However, these methods do not scale to large graphs because they usually produce large and dense matrices which do not fit into memory.

In this paper, we propose BEAR, a fast, scalable, and accurate method for computing RWR on large graphs. BEAR comprises the preprocessing step and the query step. In the preprocessing step, BEAR reorders the adjacency matrix of a given graph so that it contains a large and easy-to-invert submatrix, and precomputes several matrices including the Schur complement of the submatrix. In the query step, BEAR computes the RWR scores for a given query node quickly using a block elimination approach with the matrices computed in the preprocessing step. Through extensive experiments, we show that BEAR significantly outperforms other state-of-the-art methods in terms of preprocessing and query speed, space efficiency, and accuracy.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Design, Experimentation, Algorithms

Keywords

Proximity; relevance score; random walk with restart

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.
Copyright © 2015 ACM 978-1-4503-2758-9/15/05 ...$15.00.
http://dx.doi.org/10.1145/2723372.2723716.

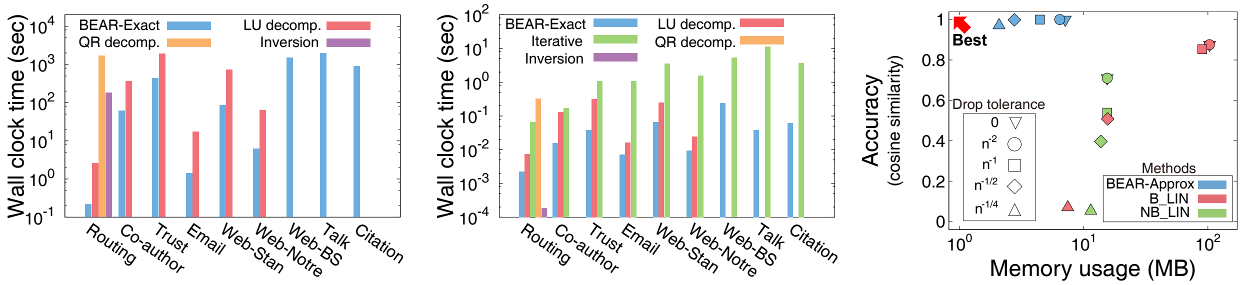
1. INTRODUCTION

Measuring the relevance (proximity) between nodes in a graph becomes the base for various data mining tasks [3, 6, 19, 21, 29, 39, 41, 40, 43, 46] and has received much interest from the database research community [5, 13, 16, 44]. Among many methods [1, 22, 31, 34] to compute the relevance, random walk with restart (RWR) [34] has been popular due to its ability to account for the global network structure [21] and the multi-faceted relationship between nodes [40]. RWR has been used in many data mining applications including ranking [42], community detection [3, 19, 43, 46], link prediction [6], and anomaly detection [39].

However, existing methods for computing RWR are unsatisfactory in terms of speed, accuracy, functionality, or scalability. The iterative method, which naturally follows from the definition of RWR is not fast: it requires repeated matrix-vector multiplications whose computational cost is not acceptable in real world applications where RWR scores for different query nodes need to be computed. Several approximate methods [3, 18, 39, 42] have also been proposed; however, their accuracies or speed-ups are unsatisfactory. Although top-k methods [16, 44] improve efficiency by focusing on finding the k most relevant nodes and ignoring irrelevant ones, finding top-k is insufficient for many data mining applications [3, 6, 18, 21, 39, 41, 43, 46] which require the relevance scores of all nodes or the least relevant nodes. Existing preprocessing methods [16, 17], which achieve better performance by preprocessing the given graph, are not scalable due to their high memory requirements.

In this paper, we propose BEAR, a fast, scalable, and accurate method for computing RWR on large graphs. BEAR comprises two steps: the preprocessing step and the query step. In the preprocessing step, BEAR reorders the adjacency matrix of a given graph so that it contains a large and easy-to-invert submatrix, and precomputes several matrices including the Schur complement [10] of the submatrix. In the query step, BEAR computes the RWR scores for a given query node quickly using a block elimination approach with the matrices computed in the preprocessing step. BEAR has two versions: an exact method BEAR-Exact and an approximate method BEAR-Approx. The former provides accuracy assurance; the latter gives faster query speed and requires less space by allowing small accuracy loss.

Through extensive experiments with various real-world datasets, we demonstrate the superiority of BEAR over



(a) Preprocessing time of exact methods

(b) Query time of exact methods

(c) Space for preprocessed data vs. Accuracy in approximate methods

Figure 1: Performance of BEAR. (a) and (b) compare exact methods, while (c) compares approximate methods. In (a) and (b), bars are omitted if and only if the corresponding experiments run out of memory. (a) In the preprocessing phase, BEAR-Exact is the fastest and the most scalable among all preprocessing methods. (b) BEAR-Exact is the fastest in the query phase except for the smallest Routing dataset. Due to its up to 300× faster query speed, BEAR-Exact outperforms the iterative method, despite the preprocessing cost, when RWR scores for different query nodes are required. (c) BEAR-Approx achieves both space efficiency and accuracy. Details of these experiments are explained in Section 4.

other state-of-the-art methods as seen in Figure 1. We also discuss how our method can be applied to other random-walk-based measures such as personalized PageRank [33], effective importance [9], and RWR with normalized graph Laplacian [42]. The main characteristics of our method are the followings:

- **Fast.** BEAR-Exact is faster up to 8× in the query phase and up to 12× in the preprocessing phase than other exact methods (Figures 1(a) and 1(b)); BEAR-Approx achieves a better time/accuracy trade-off in the query phase than other approximate methods (Figure 8(a)).
- **Space-efficient.** Compared with their respective competitors, BEAR-Exact requires up to 22× less memory space (Figure 5), and BEAR-Approx provides a better space/accuracy trade-off (Figure 1(c)).
- **Accurate.** BEAR-Exact guarantees exactness (Theorem 1); BEAR-Approx enjoys a better trade-off between accuracy, time, and space than other approximate methods (Figure 8).
- **Versatile.** BEAR can be applied to diverse RWR variants, including personalized PageRank, effective importance, and RWR with normalized graph Laplacian (Section 3.4).

The binary code of our method and several datasets are available at `http://kdmlab.org/bear`. The rest of the paper is organized as follows. Section 2 presents preliminaries on RWR. Our proposed BEAR is described in Section 3 followed by experimental results in Section 4. After providing a review on related work in Section 5, we make conclusion in Section 6.

2. PRELIMINARIES

In this section, we describe the preliminaries on random walk with restart and its algorithms. Table 1 lists the symbols used in this paper. We denote matrices by boldface capitals, e.g., $\mathbf{A}$, and vectors by boldface lowercases, e.g., $\mathbf{q}$.

2.1 Random Walk with Restart

Random walk with restart (RWR) [42] measures each node's relevance w.r.t. a given seed node s in a given graph. It assumes a random surfer who occasionally gets bored with following the edges in the graph and restarts at node s. The surfer starts at node s, and at each current node, either restarts at node s (with probability c) or moves to a neighboring node along an edge (with probability $1-c$). The probability that each edge is chosen is proportional to its weight in the adjacency matrix. $\tilde{\mathbf{A}}$ denotes the row-normalized adjacency matrix, whose (u, v)th entry is the probability that a surfer at node u chooses the edge to node v among all edges from node u. The stationary probability of being at each node corresponds to its RWR score w.r.t. node s, and is denoted by $\mathbf{r}$, whose uth entry corresponds to node u's RWR score. The vector $\mathbf{r}$ satisfies the following equation:

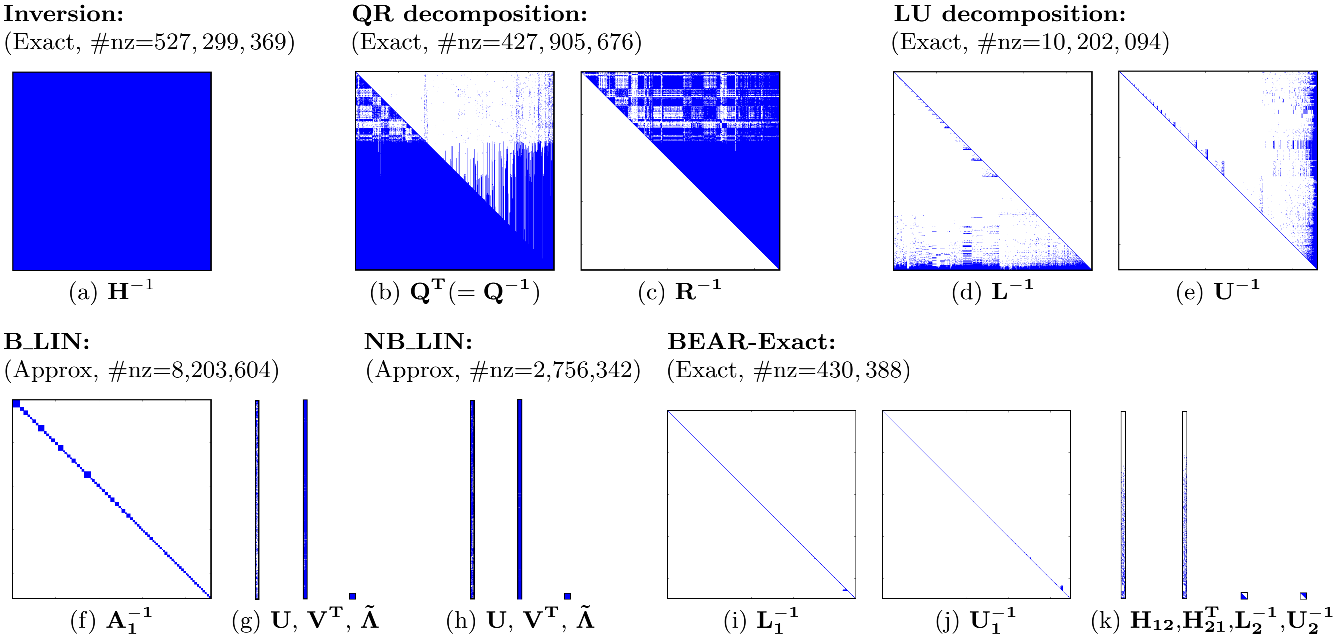
$$\mathbf{r} = (1-c)\tilde{\mathbf{A}}^{\mathbf{T}}\mathbf{r} + c\mathbf{q} \tag{1}$$

where $\mathbf{q}$ is the starting vector with the index of the seed node s is set to 1 and others to 0. It can be obtained by solving the following linear equation:

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

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_{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
ξ	drop tolerance
$\tilde{\mathbf{A}}$	$(n \times n)$ 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-decomposed matrices of $\mathbf{H_{11}}$
$\mathbf{L_2}, \mathbf{U_2}$	$(n_2 \times n_2)$ LU-decomposed matrices of $\mathbf{S}$
$\mathbf{q}$	$(n \times 1)$ starting vector
$\mathbf{q_i}$	$(n_i \times 1)$ ith partition of $\mathbf{q}$
$\mathbf{r}$	$(n \times 1)$ relevance vector
$\mathbf{r_i}$	$(n_i \times 1)$ ith partition of $\mathbf{r}$
T	number of SlashBurn iterations
k	number of hubs removed at a time in SlashBurn
t	rank in B_LIN and NB_LIN
ϵ	threshold to stop iteration in iterative methods
ϵ_b	threshold to expand nodes in RPPR and BRPPR



(a) $\mathbf{H}^{-1}$ (b) $\mathbf{Q}^{\mathbf{T}}(= \mathbf{Q}^{-1})$ (c) $\mathbf{R}^{-1}$ (d) $\mathbf{L}^{-1}$ (e) $\mathbf{U}^{-1}$

(f) $\mathbf{A_1^{-1}}$ (g) $\mathbf{U}$, $\mathbf{V}^{\mathbf{T}}$, $\tilde{\mathbf{\Lambda}}$ (h) $\mathbf{U}$, $\mathbf{V}^{\mathbf{T}}$, $\tilde{\mathbf{\Lambda}}$ (i) $\mathbf{L_1^{-1}}$ (j) $\mathbf{U_1^{-1}}$ (k) $\mathbf{H_{12}}$,$\mathbf{H_{21}^{T}}$,$\mathbf{L_2^{-1}}$,$\mathbf{U_2^{-1}}$

Figure 2: Sparsity patterns of the matrices resulted from different preprocessing methods on the Routing dataset (see Section 4.1 for the details of the dataset). Exact: exact method, Approx: approximate method, #nz: number of non-zero elements in the precomputed matrices. For B_LIN and NB_LIN, rank t is set to 500 and drop tolerance ξ is set to 0. Precomputed matrices are loaded into memory to speed up the query phase; for the reason, the number of non-zero entries in them determines the memory usage of each method and thus its scalability. Our exact method BEAR-Exact produces the smallest number of non-zero entries than all other methods including the approximate methods (1200× than Inversion and 6× than the second best method). Our approximate method BEAR-Approx further decreases the number of non-zero entries, depending on drop tolerance ξ.

Personalized PageRank. Personalized PageRank (PPR) [33] is an extension of RWR. PPR calculates the relevance of nodes according to the preference of each user, and is widely used for personalized search. A random surfer in PPR either jumps to a random node according to the probability distribution (user preference distribution) given by $\mathbf{q}$ (with probability c) or moves to a neighboring node (with probability $1-c$). PPR can be viewed as a generalized version of RWR with multiple seed nodes. Equations (1) and (2) can be directly applied to PPR with the modified $\mathbf{q}$.

2.2 Algorithms for RWR

We review two basic methods for RWR computation and four recent methods for addressing the limitations of the basic methods. We also point out a need for improvement which we will address in the following section. Since most applications require RWR scores for different seed nodes, whether at once or on demand, we separate the preprocessing phase, which occurs once, from the query phase, which occurs for each seed node.

Iterative method. The iterative method repeats updating $\mathbf{r}$ until convergence ($|\mathbf{r}^{(i)} - \mathbf{r}^{(i-1)}| < \epsilon$) by the following update rule:

$$\mathbf{r}^{(i)} \leftarrow (1-c)\tilde{\mathbf{A}}^{\mathbf{T}}\mathbf{r}^{(i-1)} + c\mathbf{q} \tag{3}$$

where the superscript i denotes the iteration number. If $0 < c < 1$, $\mathbf{r}^{(i)}$ is guaranteed to converge to a unique solution [27]. This method does not require preprocessing (one-time cost) but has expensive query cost which incurs a repeated matrix-vector multiplication. Thus, it is inefficient when RWR scores for many query nodes are required.

RPPR / BRPPR. Gleich et al. [18] propose restricted personalized PageRank (RPPR), which speeds up the iterative method by accessing only a part of a graph. This algorithm uses Equation 3 only for a subgraph and the nodes contained in it. The subgraph is initialized to a given seed node and grows as iteration proceeds. A node contained in the subgraph is on the boundary if its outgoing edges (in the original graph) are not contained in the subgraph, and the outgoing edges and outgoing neighbors of the node are added to the subgraph if the RWR score of the node (in the current iteration) is greater than a threshold ϵ_b. The algorithm repeats iterations until RWR scores converge. The RWR scores of nodes outside the subgraph are set to zero. Boundary-restricted personalized PageRank (BRPPR) [18] is a variant of the RPPR. It expands nodes on the boundary in decreasing order of their RWR scores (in the current iteration) until the sum of the RWR scores of nodes on the boundary becomes less than a threshold ϵ_b. Although these methods reduce the query cost of the iterative method significantly, they do not guarantee exactness.

Inversion. An algebraic method directly calculates $\mathbf{r}$ from Equation (2) as follows:

$$\mathbf{r} = c(\mathbf{I} - (1-c)\tilde{\mathbf{A}}^{\mathbf{T}})^{-1}\mathbf{q} = c\mathbf{H}^{-1}\mathbf{q}. \tag{4}$$

The matrix $\mathbf{H}$ is known to be invertible when $0 < c < 1$ [27]. Once $\mathbf{H}^{-1}$ is computed in the preprocessing phase, $\mathbf{r}$ can be obtained efficiently in the query phase. However, this is again impractical for large graphs because calculating $\mathbf{H}^{-1}$ is computationally expensive and $\mathbf{H}^{-1}$ is usually too dense to fit into memory as seen in Figure 2(a).

QR decomposition. To avoid the problem regarding $\mathbf{H}^{-1}$, Fujiwara et al. [17] decompose $\mathbf{H}$ using QR decomposition, and then use $\mathbf{Q}^{\mathbf{T}}(= \mathbf{Q}^{-1})$ and $\mathbf{R}^{-1}$ instead of $\mathbf{H}^{-1}$ as follows:

$$\mathbf{r} = c\mathbf{H}^{-1}\mathbf{q} = c\mathbf{R}^{-1}(\mathbf{Q}^{\mathbf{T}}\mathbf{q})$$

where $\mathbf{H} = \mathbf{QR}$. They also propose a reordering rule for $\mathbf{H}$ which makes $\mathbf{Q^T}$ and $\mathbf{R^{-1}}$ sparser. However, on the most datasets used in our experiments, QR decomposition results in dense matrices as shown in Figures 2(b) and 2(c); thus, its scalability is limited. This fact agrees with the claim made by Boyd et al. [10] that it is difficult to exploit sparsity in QR decomposition.

LU decomposition. To replace $\mathbf{H^{-1}}$, Fujiwara et al. [16] also exploit LU decomposition using the following rule:

$$\mathbf{r} = c\mathbf{H^{-1}q} = c\mathbf{U^{-1}}(\mathbf{L^{-1}q})$$

where $\mathbf{H} = \mathbf{LU}$. Prior to the decomposition, $\mathbf{H}$ is reordered based on nodes' degrees and community structure. This makes matrices $\mathbf{L^{-1}}$ and $\mathbf{U^{-1}}$ sparser as seen in Figures 2(d) and 2(e). We incorporate their idea into our method to replace inverse terms, which will be explained in detail in Section 3.

B_LIN / NB_LIN. Tong et al. [42] partition a given graph and divide $\mathbf{\tilde{A}^T}$ into $\mathbf{A_1}$ (inner-partition edges) and $\mathbf{A_2}$ (cross-partition edges). Then, they use a heuristic decomposition method with given rank t to approximate $\mathbf{A_2}$ with low-rank matrix $\mathbf{U\Sigma V}$ where $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ are $n \times t$, $t \times t$, and $t \times n$ matrices, respectively. In the query phase, they apply Sherman-Morrison lemma [35] to efficiently calculate $\mathbf{r}$ as follows:

$$\begin{aligned}\mathbf{r} &= c(\mathbf{I} - (1-c)\mathbf{\tilde{A}^T})^{-1}\mathbf{q}\\ &\approx c(\mathbf{I} - (1-c)\mathbf{A_1} - (1-c)\mathbf{U\Sigma V})^{-1}\mathbf{q}\\ &= c(\mathbf{A_1^{-1}q} + (1-c)\mathbf{A_1^{-1}U\tilde{\Lambda}VA_1^{-1}q})\end{aligned}$$

where $\mathbf{\tilde{\Lambda}} = (\mathbf{\Sigma}^{-1} - c\mathbf{VA_1^{-1}U})^{-1}$. To sparsify the precomputed matrices, near-zero entries whose absolute value is smaller than ξ are dropped. This method is called B_LIN, and its variant NB_LIN directly approximates $\mathbf{\tilde{A}^T}$ without partitioning it. Both methods do not guarantee exactness.

As summarized in Figure 2, previous preprocessing methods require too much space for preprocessed data or do not guarantee accuracy. Our proposed BEAR, explained in the following section, achieves both space efficiency and accuracy as seen in Figures 2(i) through 2(k).

3. PROPOSED METHOD

In this section, we describe BEAR, our proposed method for fast, scalable, and accurate RWR computation. BEAR has two versions: BEAR-Exact for exact RWR, and BEAR-Approx for approximate RWR. BEAR-Exact guarantees accuracy, while BEAR-Approx improves speed and space efficiency by sacrificing little accuracy. A pictorial description of BEAR is provided in Figure 3. BEAR exploits the following ideas:

- The adjacency matrix of real-world graphs can be reordered so that it has a large but easy-to-invert submatrix, such as a block-diagonal matrix as shown in the upper left part of Figure 4(b).
- A linear equation like Equation (2) is easily solved by block elimination using Schur complement if the matrix contains a large and easy-to-invert submatrix such as a block-diagonal one.
- Compared with directly inverting an adjacency matrix, inverting its LU-decomposed matrices is more efficient in terms of time and space.

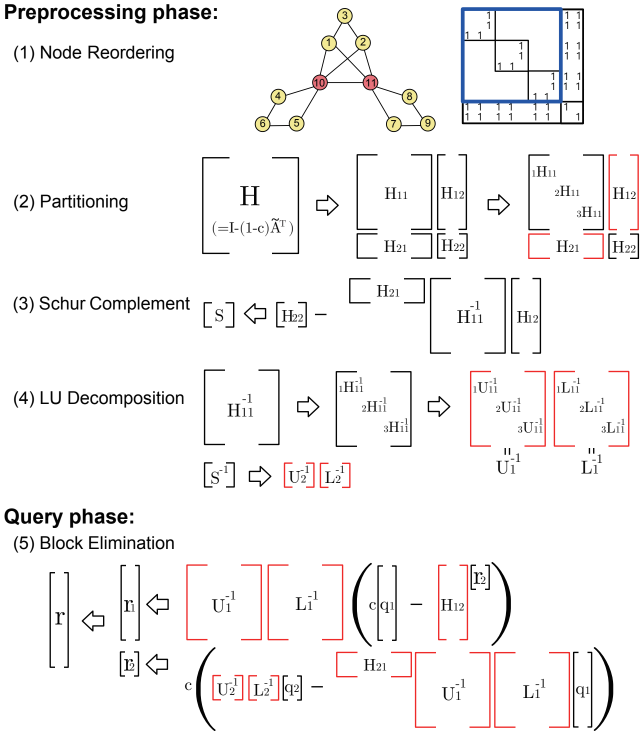


Figure 3: A pictorial description of BEAR. The output matrices of the preprocessing phase are red-bordered. (1) Reorder nodes so that the adjacency matrix has a large block-diagonal submatrix (the blue-bordered one). (2) Partition $\mathbf{H}$ into four blocks so that $\mathbf{H_{11}}$ corresponds to the submatrix. (3) Compute the Schur complement $\mathbf{S}$ of $\mathbf{H_{11}}$. (4) Since $\mathbf{S^{-1}}$ and the diagonal blocks of $\mathbf{H_{11}^{-1}}$ are likely to be dense, store the inverse of the LU decomposed matrices of $\mathbf{S}$ and $\mathbf{H_{11}}$ instead to save space. Notice that these can be computed efficiently in terms of time and space because $\mathbf{S}$ and the diagonal blocks of $\mathbf{H_{11}}$ are relatively small compared with $\mathbf{H}$. (5) Compute the RWR score vector $\mathbf{r}$ for a query vector $\mathbf{q}$ (the concatenation of $\mathbf{q_1}$ and $\mathbf{q_2}$) fast by utilizing the precomputed matrices.

Algorithms 1 and 2 represent the procedure of BEAR. Since RWR scores are requested for different seed nodes in real world applications, we separate the preprocessing phase (Algorithm 1), which is run once, from the query phase (Algorithm 2), which is run for each seed node. The exact method BEAR-Exact and the approximate method BEAR-Approx differ only at line 9 of Algorithm 1; the detail is in Section 3.1.4. To exploit their sparsity, all matrices considered are saved in a sparse matrix format, such as the compressed sparse column format [36], which stores only non-zero entries.

3.1 Preprocessing Phase

The overall preprocessing phase of BEAR is shown in Algorithm 1 and the details in the following subsections.

3.1.1 Node Reordering (lines 2 through 4)

In this part, we reorder $\mathbf{H}(= \mathbf{I} - (1-c)\mathbf{\tilde{A}^T})$ and partition it. Our objective is to reorder $\mathbf{H}$ so that it has a large but easy-to-invert submatrix such as a block-diagonal one. Any node reordering method (e.g., spectral clustering [32], cross association [11], shingle [14], etc.) can be used for this purpose; in this paper, we use a method which improves on SlashBurn [23] since it is the state-of-the-art method in concentrating the nonzeros of adjacency matrices

```
Algorithm 1: Preprocessing phase in BEAR
Input: graph: $G$, restart probability: $c$, drop tolerance: $\xi$
Output: precomputed matrices: $\mathbf{L_1^{-1}}$, $\mathbf{U_1^{-1}}$, $\mathbf{L_2^{-1}}$, $\mathbf{U_2^{-1}}$, $\mathbf{H_{12}}$,
    and $\mathbf{H_{21}}$
 1: compute $\tilde{\mathbf{A}}$, and $\mathbf{H} = \mathbf{I} - (1-c)\tilde{\mathbf{A}}^{\mathbf{T}}$
 2: find hubs using SlashBurn [23], and divide spokes into
    disconnected components by removing the hubs
 3: reorder nodes and $\mathbf{H}$ so that the disconnected components
    form a block-diagonal submatrix $\mathbf{H_{11}}$ where each block is
    ordered in the ascending order of degrees within the
    component
 4: partition $\mathbf{H}$ into $\mathbf{H_{11}}, \mathbf{H_{12}}, \mathbf{H_{21}}, \mathbf{H_{22}}$
 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: reorder the hubs in the ascending order of their degrees in $\mathbf{S}$
    and reorder $\mathbf{S}$, $\mathbf{H_{21}}$ and $\mathbf{H_{12}}$ according to it
 8: decompose $\mathbf{S}$ into $\mathbf{L_2}$ and $\mathbf{U_2}$ using LU decomposition and
    compute $\mathbf{L_2^{-1}}$ and $\mathbf{U_2^{-1}}$
 9: (BEAR-Approx only) drop entries whose absolute value is
    smaller than $\xi$ in $\mathbf{L_1^{-1}}$, $\mathbf{U_1^{-1}}$, $\mathbf{L_2^{-1}}$, $\mathbf{U_2^{-1}}$, $\mathbf{H_{12}}$, and $\mathbf{H_{21}}$
10: return $\mathbf{L_1^{-1}}$, $\mathbf{U_1^{-1}}$, $\mathbf{L_2^{-1}}$, $\mathbf{U_2^{-1}}$, $\mathbf{H_{12}}$, and $\mathbf{H_{21}}$
```

```
Algorithm 2: Query phase in BEAR
Input: seed node: $s$, precomputed matrices: $\mathbf{L_1^{-1}}$, $\mathbf{U_1^{-1}}$, $\mathbf{L_2^{-1}}$,
    $\mathbf{U_2^{-1}}$, $\mathbf{H_{12}}$, and $\mathbf{H_{21}}$
Output: relevance scores: $\mathbf{r}$
 1: create $\mathbf{q}$ whose $s$th entry is 1 and the others are 0
 2: partition $\mathbf{q}$ into $\mathbf{q_1}$ and $\mathbf{q_2}$
 3: compute $\mathbf{r_2} = c(\mathbf{U_2^{-1}}(\mathbf{L_2^{-1}}(\mathbf{q_2} - \mathbf{H_{21}}(\mathbf{U_1^{-1}}(\mathbf{L_1^{-1}}\mathbf{q_1})))))$
 4: compute $\mathbf{r_1} = \mathbf{U_1^{-1}}(\mathbf{L_1^{-1}}(c\mathbf{q_1} - \mathbf{H_{12}}\mathbf{r_2}))$
 5: create $\mathbf{r}$ by concatenating $\mathbf{r_1}$ and $\mathbf{r_2}$
 6: return $\mathbf{r}$
```

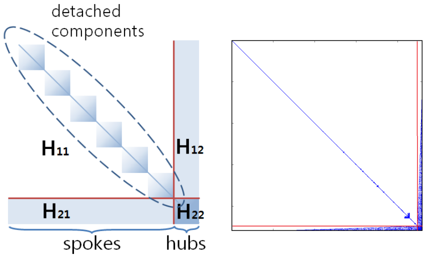
of graphs (more details in Appendix A). We first run SlashBurn on a given graph, to decompose the graph into hubs (high-degree nodes), and spokes (low-degree nodes which get disconnected from the giant connected component if the hubs are removed). Within each connected component containing spokes, we reorder nodes in the ascending order of degrees within the component. As a result, we get an adjacency matrix whose upper-left area (e.g., $\mathbf{H_{11}}$ in Figure 4(a)) is a large and sparse block diagonal matrix which is easily inverted, while the lower-right area (e.g., $\mathbf{H_{22}}$ in Figure 4(a)) is a small but dense matrix. Let n_1 denote the number of spokes and n_2 denote the number of hubs. After the reordering, we partition the matrix $\mathbf{H}$ into four pieces: $\mathbf{H_{11}}$ ($n_1 \times n_1$ matrix), $\mathbf{H_{12}}$ ($n_1 \times n_2$ matrix), $\mathbf{H_{21}}$ ($n_2 \times n_1$ matrix), and $\mathbf{H_{22}}$ ($n_2 \times n_2$ matrix), which correspond to the adjacency matrix representation of edges between spokes, from spokes to hubs, from hubs to spokes, and between hubs, respectively.

3.1.2 Schur Complement (lines 5 through 6)

In this part, we compute the Schur complement of $\mathbf{H_{11}}$, whose inverse is required in the query phase.

Definition 1 (Schur Complement [10]). *Suppose a square matrix $\mathbf{A}$ is partitioned into $\mathbf{A_{11}}$, $\mathbf{A_{12}}$, $\mathbf{A_{21}}$, and $\mathbf{A_{22}}$, which are $p \times p$, $p \times q$, $q \times p$, and $q \times q$ matrices, resp., and $\mathbf{A_{11}}$ is invertible. The Schur complement $\mathbf{S}$ of the block $\mathbf{A_{11}}$ of the matrix $\mathbf{A}$ is defined by*

$$\mathbf{S} = \mathbf{A_{22}} - \mathbf{A_{21}}\mathbf{A_{11}^{-1}}\mathbf{A_{12}}.$$



(a) Node reordering method (b) Result on the Routing datset

Figure 4: The node reordering method of BEAR and its result on the Routing dataset. BEAR reorders nodes so that edges between spokes form a large but sparse block-diagonal submatrix ($\mathbf{H_{11}}$). Diagonal blocks correspond to the connected components detached from the giant connected component when hubs are removed. Nodes in each block are sorted in the ascending order of their degrees within the component.

According to the definition, computing the Schur complement $\mathbf{S}$ ($n_2 \times n_2$ matrix) of $\mathbf{H_{11}}$ requires $\mathbf{H_{11}^{-1}}$. Instead of directly inverting $\mathbf{H_{11}}$, we LU decompose it into $\mathbf{L_1}$ and $\mathbf{U_1}$, then compute $\mathbf{L_1^{-1}}$ and $\mathbf{U_1^{-1}}$ instead (the reason will be explained in Section 3.1.3). Consequently, $\mathbf{S}$ is computed by the following rule:

$$\mathbf{S} = \mathbf{H_{22}} - \mathbf{H_{21}}(\mathbf{U_1^{-1}}(\mathbf{L_1^{-1}}\mathbf{H_{12}})). \tag{5}$$

3.1.3 LU Decomposition (lines 7 through 8)

The block elimination method, which will be explained in Section 3.2, requires $\mathbf{H_{11}^{-1}}$ and $\mathbf{S^{-1}}$ to solve Equation (2). Directly inverting $\mathbf{H_{11}}$ and $\mathbf{S}$, however, is inefficient since $\mathbf{S^{-1}}$ as well as each diagonal block of $\mathbf{H_{11}^{-1}}$ is likely to be dense. We avoid this problem by replacing $\mathbf{H_{11}^{-1}}$ with $\mathbf{U_1^{-1}}\mathbf{L_1^{-1}}$ as in Equation (5) and replacing $\mathbf{S^{-1}}$ with $\mathbf{U_2^{-1}}\mathbf{L_2^{-1}}$ where $\mathbf{L_2}$ and $\mathbf{U_2}$ denote the LU-decomposed matrices of $\mathbf{S}$. To compute $\mathbf{L_1^{-1}}$, $\mathbf{U_1^{-1}}$, $\mathbf{L_2^{-1}}$, and $\mathbf{U_2^{-1}}$ efficiently and make them sparser, we exploit the following observation [16] and lemma.

Observation 1. *Reordering nodes in the ascending order of their degrees speeds up the LU decomposition of an adjacency matrix and makes the inverse of the LU-decomposed matrices sparser.*

Lemma 1. *Suppose $\mathbf{A}$ is a non-singular block-diagonal matrix that consists of diagonal blocks, $\mathbf{A_1}$ through $\mathbf{A_b}$. Let $\mathbf{L}$ and $\mathbf{U}$ denote the LU-decomposed matrices of $\mathbf{A}$, and let $\mathbf{L_i}$ and $\mathbf{U_i}$ denote those of $\mathbf{A_i}$ for all i ($1 \le i \le b$). Then, $\mathbf{L^{-1}}$ and $\mathbf{U^{-1}}$ are the block-diagonal matrices that consist of $\mathbf{L_1^{-1}}$ through $\mathbf{L_b^{-1}}$ and $\mathbf{U_1^{-1}}$ through $\mathbf{U_b^{-1}}$, respectively.*

Proof. See Appendix B. □

These observation and lemma suggest for $\mathbf{H_{11}^{-1}}$ the reordering method we already used in Section 3.1.1. Since we computed $\mathbf{L_1^{-1}}$ and $\mathbf{U_1^{-1}}$ in Section 3.1.2, we only need to process $\mathbf{S}$. We first rearrange the hubs in the ascending order of their degrees within $\mathbf{S}$, which reorders $\mathbf{H_{12}}$, $\mathbf{H_{21}}$, and $\mathbf{H_{22}}$ as well as $\mathbf{S}$. Note that computing $\mathbf{S}$ after reordering the hubs, and reordering the hubs after computing $\mathbf{S}$ produce the same result. After decomposing $\mathbf{S}$ into $\mathbf{L_2}$ and $\mathbf{U_2}$, we compute $\mathbf{L_2^{-1}}$ and $\mathbf{U_2^{-1}}$.

3.1.4 Dropping Near-zero Entries (line 9, BEAR-Approx only)

The running time and the memory usage in the query phase of our method largely depend on the number of non-zero entries in $\mathbf{L_1^{-1}}$, $\mathbf{U_1^{-1}}$, $\mathbf{L_2^{-1}}$, $\mathbf{U_2^{-1}}$, $\mathbf{H_{12}}$, and $\mathbf{H_{21}}$. Thus, dropping near-zero entries in them saves time and memory space in the query phase although it sacrifices little accuracy of $\mathbf{r}$. BEAR-Approx drops entries whose absolute value is smaller than the drop tolerance ξ. The effects of different ξ values on accuracy, query time, and memory usage are empirically analyzed in Section 4. Note that, contrary to BEAR-Approx, BEAR-Exact guarantees the exactness of $\mathbf{r}$, which will be proved in Section 3.2, and still outperforms other exact methods in terms of time and space, which will be shown in Section 4.

3.2 Query Phase

In the query phase, BEAR computes the RWR score vector $\mathbf{r}$ w.r.t. a given seed node s by exploiting the results of the preprocessing phase. Algorithm 2 describes the overall procedure of the query phase.

The vector $\mathbf{q}=\begin{bmatrix}\mathbf{q_1}\\\mathbf{q_2}\end{bmatrix}$ 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 $\mathbf{q_1}$ and the length-n_2 vector $\mathbf{q_2}$. The exact RWR score vector $\mathbf{r}$ is computed by the following equation:

$$\mathbf{r}=\begin{bmatrix}\mathbf{r_1}\\\mathbf{r_2}\end{bmatrix}=\begin{bmatrix}\mathbf{U_1^{-1}}(\mathbf{L_1^{-1}}(c\mathbf{q_1}-\mathbf{H_{12}r_2}))\\ c(\mathbf{U_2^{-1}}(\mathbf{L_2^{-1}}(\mathbf{q_2}-\mathbf{H_{21}}(\mathbf{U_1^{-1}}(\mathbf{L_1^{-1}q_1})))))\end{bmatrix}. \tag{6}$$

To prove the correctness of the above equation, we use the block elimination method.

Lemma 2 (Block elimination [10]). *Suppose a linear equation* $\mathbf{Ax}=\mathbf{b}$ *is partitioned as*

$$\begin{bmatrix}\mathbf{A_{11}} & \mathbf{A_{12}}\\\mathbf{A_{21}} & \mathbf{A_{22}}\end{bmatrix}\begin{bmatrix}\mathbf{x_1}\\\mathbf{x_2}\end{bmatrix}=\begin{bmatrix}\mathbf{b_1}\\\mathbf{b_2}\end{bmatrix}$$

where $\mathbf{A_{11}}$ *and* $\mathbf{A_{22}}$ *are square matrices. If the submatrix* $\mathbf{A_{11}}$ *is invertible, and* $\mathbf{S}$ *is the Schur complement of the submatrix* $\mathbf{A_{11}}$ *in* $\mathbf{A}$,

$$\mathbf{x}=\begin{bmatrix}\mathbf{x_1}\\\mathbf{x_2}\end{bmatrix}=\begin{bmatrix}\mathbf{A_{11}^{-1}}(\mathbf{b_1}-\mathbf{A_{12}x_2})\\\mathbf{S^{-1}}(\mathbf{b_2}-\mathbf{A_{21}A_{11}^{-1}b_1})\end{bmatrix}. \tag{7}$$

Theorem 1 (The correctness of BEAR-Exact). *The* $\mathbf{r}$ *in Equation* (6) *is equal to the* $\mathbf{r}$ *in Equation* (2).

Proof. $\mathbf{H_{11}}$ *is invertible because its transpose is a strictly diagonally dominant matrix for* $0 < c < 1$ *[8]. Thus, by Lemma 2, the following holds for Equation* (2)*:*

$$\mathbf{r}=\begin{bmatrix}\mathbf{r_1}\\\mathbf{r_2}\end{bmatrix}=\begin{bmatrix}\mathbf{H_{11}^{-1}}(c\mathbf{q_1}-\mathbf{H_{12}r_2})\\\mathbf{S^{-1}}(c\mathbf{q_2}-\mathbf{H_{21}H_{11}^{-1}}(c\mathbf{q_1}))\end{bmatrix}.$$

Equation (6) *only replaces* $\mathbf{H_{11}^{-1}}$ *with* $\mathbf{U_1^{-1}L_1^{-1}}$ *and* $\mathbf{S^{-1}}$ *with* $\mathbf{U_2^{-1}L_2^{-1}}$ *where* $\mathbf{H_{11}}=\mathbf{L_1U_1}$ *and* $\mathbf{S}=\mathbf{L_2U_2}$. □

3.3 Complexity Analysis

In this section, we analyze the time and space complexity of BEAR. We assume that all the matrices considered are saved in a sparse format, such as the compressed sparse column format [36], which stores only non-zero entries, and that all the matrix operations exploit such sparsity by only considering non-zero entries. We also assume that the number of edges is greater than that of nodes (i.e., $m > n$) for simplicity. The maximum number of non-zero entries in each precomputed matrix is summarized in Table 2 where b denotes the number of diagonal blocks in $\mathbf{H_{11}}$, and n_{1i} denotes the number of nodes in the ith diagonal block. The maximum numbers of non-zero entries in $\mathbf{L_1^{-1}}$ and $\mathbf{U_1^{-1}}$ depend on the size of diagonal blocks because the LU-decomposed matrices and inverse of a block-diagonal matrix are also block-diagonal matrices with the same block sizes (see Lemma 1).

Table 2: Maximum number of non-zero entries in the precomputed matrices.

Matrix	Max nonzeros
$\mathbf{H_{12}}$ & $\mathbf{H_{21}}$	$O(\min(n_1n_2, m))$
$\mathbf{L_1^{-1}}$ & $\mathbf{U_1^{-1}}$	$O(\sum_{i=1}^{b} n_{1i}^2)$
$\mathbf{L_2^{-1}}$ & $\mathbf{U_2^{-1}}$	$O(n_2^2)$

3.3.1 Time Complexity

The time complexity of each step of BEAR is summarized in Table 3. In this section, we provide proofs for nontrivial analysis results starting from the following lemma:

Lemma 3 (Sparse matrix multiplication). *Suppose* $\mathbf{A}$ *and* $\mathbf{B}$ *are* $p\times q$ *and* $q\times r$ *matrices, respectively, and* $\mathbf{A}$ *has* $|\mathbf{A}|$ $(> p, q)$ *non-zero entries. Calculating* $\mathbf{C}=\mathbf{AB}$ *using sparse matrix multiplication takes* $O(|\mathbf{A}|r)$.

Proof. *Each non-zero entries in* $\mathbf{A}$ *is multiplied and then added up to* r *times.* □

Lemma 4. *It takes* $O(\sum_{i=1}^{b} n_{1i}^3)$ *to compute* $\mathbf{L_1^{-1}}$ *and* $\mathbf{U_1^{-1}}$.

Proof. *By Lemma 1, each block of* $\mathbf{H_{11}}$ *is processed separately. Since the LU decomposition of* $p\times p$ *matrix takes* $O(p^3)$ *[10], it takes* $O(\sum_{i=1}^{b} n_{1i}^3)$ *to LU decompose the diagonal blocks of* $\mathbf{H_{11}}$. *Since the inversion of* $p\times p$ *matrix takes* $O(p^3)$ *[10], it takes* $O(\sum_{i=1}^{b} n_{1i}^3)$ *to invert the decomposed blocks.* □

Lemma 5. *It takes* $O(n_2\sum_{i=1}^{b} n_{1i}^2+\min(n_1n_2^2, n_2m))$ *to compute* $\mathbf{S}=\mathbf{H_{22}}-\mathbf{H_{21}}(\mathbf{U_1^{-1}}(\mathbf{L_1^{-1}H_{12}}))$.

Proof. *By Lemma 3 and Table 2, it takes* $O(n_2\sum_{i=1}^{b} n_{1i}^2)$ *to compute* $\mathbf{R_1}=\mathbf{L_1^{-1}H_{12}}$, $O(n_2\sum_{i=1}^{b} n_{1i}^2)$ *to compute* $\mathbf{R_2}=\mathbf{U_1^{-1}R_1}$, $O(\min(n_1n_2^2, n_2m))$ *to compute* $\mathbf{R_3}=\mathbf{H_{21}R_2}$, *and* $O(n_2^2)$ *to compute* $\mathbf{S}=\mathbf{H_{22}}-\mathbf{R_3}$. □

Theorem 2. *Preprocessing phase in* BEAR *takes* $O(T(m+n\log n)+\sum_{i=1}^{b} n_{1i}^3+n_2\sum_{i=1}^{b} n_{1i}^2+n_2^3+\min(n_2^2n_1, n_2m))$

Proof. *See Lemma 4, Lemma 5, and Table 3.* □

Theorem 3. *Query phase in* BEAR *takes* $O(\sum_{i=1}^{b} n_{1i}^2+n_2^2+\min(n_1n_2, m))$

Proof. *Apply Lemma 3 and the results in Table 2 to each steps in* $\mathbf{r_2}=c(\mathbf{U_2^{-1}}(\mathbf{L_2^{-1}}(\mathbf{q_2}-\mathbf{H_{21}}(\mathbf{U_1^{-1}}(\mathbf{L_1^{-1}q_1})))))$ *and* $\mathbf{r_1}=\mathbf{U_1^{-1}}(\mathbf{L_1^{-1}}(c\mathbf{q_1}-\mathbf{H_{12}r_2}))$ *as in Lemma 5.* □

In real-world graphs, $\sum_{i=1}^{b} n_{1i}^2$ in the above results can be replaced by m since the number of non-zero entries in $\mathbf{L_1^{-1}}$ and $\mathbf{U_1^{-1}}$ is closer to $O(m)$ than $O(\sum_{i=1}^{b} n_{1i}^2)$ as seen in Table 4.

Table 3: Time complexity of each step of BEAR.

Line	Task	Time complexity
		Preprocessing phase (Algorithm 1)
2	run SlashBurn	$O(T(m + n \log n))$ [23]
3	reorder $\mathbf{H}$	$O(m + n + \sum_{i=1}^{b} n_{1i} \log n_{1i})$
5	compute $\mathbf{L_1^{-1}}$ and $\mathbf{U_1^{-1}}$	$O(\sum_{i=1}^{b} n_{1i}^3)$
6	compute $\mathbf{S}$	$O(n_2 \sum_{i=1}^{b} n_{1i}^2 + \min(n_1 n_2^2, n_2 m))$
8	compute $\mathbf{L_2^{-1}}$ and $\mathbf{U_2^{-1}}$	$O(n_2^3)$
9	drop near-zero entries	$O(\sum_{i=1}^{b} n_{1i}^2 + n_2^2 + min(n_1 n_2, m))$
Total		$O(T(m + n \log n) + \sum_{i=1}^{b} n_{1i}^3 + n_2 \sum_{i=1}^{b} n_{1i}^2 + n_2^3 + \min(n_2^2 n_1, n_2 m))$
		Query phase (Algorithm 2)
3	compute $\mathbf{r_2}$	$O(\sum_{i=1}^{b} n_{1i}^2 + n_2^2 + \min(n_1 n_2, m))$
4	compute $\mathbf{r_1}$	$O(\sum_{i=1}^{b} n_{1i}^2 + \min(n_1 n_2, m))$
Total		$O(\sum_{i=1}^{b} n_{1i}^2 + n_2^2 + \min(n_1 n_2, m))$

3.3.2 Space Complexity

THEOREM 4. BEAR *requires* $O(\sum_{i=1}^{b} n_{1i}^2 + \min(n_1 n_2, m) + n_2^2)$ *memory space for precomputed matrices:* $\mathbf{H_{12}}$, $\mathbf{H_{21}}$, $\mathbf{L_1^{-1}}$, $\mathbf{U_1^{-1}}$, $\mathbf{L_2^{-1}}$, *and* $\mathbf{U_2^{-1}}$.

PROOF. *See Table 2.* □

For the same reason, as in the time complexity, $\sum_{i=1}^{b} n_{1i}^2$ in the above result can be replaced by m in real-world graphs.

Theorems 2, 3, and 4 imply that BEAR works efficiently when the given graph is divided into small pieces (small $\sum_{i=1}^{b} n_{1i}^2$ and $\sum_{i=1}^{b} n_{1i}^3$) by removing a small number of hubs (small n_2), which is true in many real world graphs [2, 23].

3.4 Application to RWR Variants

Our BEAR method is easily applicable to various RWR variants since BEAR does not assume any unique property of RWR contrary to other methods [16, 44]. In this section, we show how BEAR can be applied to three of such variants.

Personalized PageRank. As explained in Section 2.1, personalized PageRank (PPR) selects a restart node according to given probability distribution. PPR can be computed by replacing $\mathbf{q}$ in Algorithm 2 with the probability distribution.

Effective importance. Effective importance (EI) [9] is the degree-normalized version of RWR. It captures the local community structure and adjusts RWR's preference towards high-degree nodes. We can compute EI by dividing each entry of $\mathbf{r}$ in Algorithm 2 by the degree of the corresponding node.

RWR with normalized graph Laplacian. Instead of row-normalized adjacency matrix, Tong et al. [42] use the normalized graph Laplacian. It outputs symmetric relevance scores for undirected graphs, which are desirable for some applications. This score can be computed by replacing $\tilde{\mathbf{A}}$ in Algorithm 1 with $\mathbf{D^{-1/2}AD^{-1/2}}$ where $\mathbf{A}$ is an unnormalized adjacency matrix and $\mathbf{D}$ is a diagonal matrix whose (i, i)th entry is the degree of ith node.

4. EXPERIMENTS

To evaluate the effectiveness of our exact method BEAR-EXACT, we design and conduct experiments which answer the following questions:

- **Q1. Preprocessing cost (Section 4.2).** How much memory space do BEAR-EXACT and its competitors require for their precomputed results? How long does this preprocessing phase take?
- **Q2. Query cost (Section 4.3).** How quickly does BEAR-EXACT answer an RWR query compared with other methods?
- **Q3. Effects of network structure (Section 4.4).** How does network structure affect the preprocessing time, query time, and space requirements of BEAR-EXACT?

For our approximate method BEAR-APPROX, our experiments answer the following questions:

- **Q4. Effects of drop tolerance (Section 4.5).** How does drop tolerance ξ affect the accuracy, query time, and space requirements of BEAR-APPROX?
- **Q5. Comparison with approximate methods (Section 4.6).** Does BEAR-APPROX provide a better trade-off between accuracy, time, and space compared with its competitors?

4.1 Experimental Settings

Machine. All the experiments are conducted on a PC with a 4-core Intel i5-4570 3.2GHz CPU and 16GB memory.

Data. The graph data used in our experiments are summarized in Table 4. A brief description of each real-world dataset is presented in Appendix C. For synthetic graphs, we use R-MAT [12] with different p_{ul}, the probability of an edge falling into the upper-left partition. The probabilities for the other partitions are set to $(1 - p_{ul})/3$, respectively.

Implementation. We compare our methods with the iterative method, RPPR [18], BRPPR [18], inversion, LU decomposition [16], QR decomposition [17], B_LIN [42], and NB_LIN [42], all of which are explained in Section 2.2. Methods only applicable to undirected graphs [3] or top-k search [20, 44] are excluded. All the methods including BEAR-EXACT and BEAR-APPROX are implemented using MATLAB, which provides a state-of-the-art linear algebra package. In particular, our implementation of NB_LIN and that of RPPR optimize their open-sourced implementations[1,2] in terms of preprocessing speed and query speed, respectively. The binary code of our method and several datasets are available at `http://kdmlab.org/bear`.

[1] http://www.cs.cmu.edu/~htong/pdfs/FastRWR_20080319.tgz

[2] http://www.mathworks.co.kr/matlabcentral/fileexchange/11613-pagerank

Table 4: Summary of real-world and synthetic datasets. $|\mathbf{M}|$ denotes the number of non-zero entries in the matrix $\mathbf{M}$. A brief description of each real-world dataset is presented in Appendix C.

dataset	n	m	n_2	$\sum_{i=1}^{b} n_{1i}^2$	$\lvert\mathbf{H}\rvert$	$\lvert\mathbf{H_{12}}\rvert + \lvert\mathbf{H_{21}}\rvert$	$\lvert\mathbf{L_1^{-1}}\rvert + \lvert\mathbf{U_1^{-1}}\rvert$	$\lvert\mathbf{L_2^{-1}}\rvert + \lvert\mathbf{U_2^{-1}}\rvert$
Routing	22,963	48,436	572	678,097	119,835	72,168	86,972	271,248
Co-author	31,163	120,029	4,464	1,364,443	271,221	118,012	202,482	18,526,862
Trust	131,828	841,372	10,087	3,493,773	972,627	317,874	318,461	81,039,727
Email	265,214	420,045	1,590	566,435	684,170	358,458	554,681	1,149,462
Web-Stan	281,903	2,312,497	16,017	420,658,754	2,594,400	1,423,993	26,191,040	55,450,105
Web-Notre	325,729	1,497,134	12,350	77,441,937	1,795,408	611,408	5,912,673	12,105,579
Web-BS	685,230	7,600,595	50,005	717,727,201	8,285,825	4,725,657	25,990,336	547,936,698
Talk	2,394,385	5,021,410	19,152	3,272,265	7,415,795	3,996,546	4,841,878	246,235,838
Citation	3,774,768	16,518,948	1,143,522	71,206,030	20,293,715	9,738,225	6,709,469	408,178,662
R-MAT (0.5)	99,982	500,000	17,721	1,513,855	599,982	184,812	204,585	260,247,890
R-MAT (0.6)	99,956	500,000	11,088	1,258,518	599,956	145,281	205,837	104,153,333
R-MAT (0.7)	99,707	500,000	7,029	705,042	599,707	116,382	202,922	43,250,097
R-MAT (0.8)	99,267	500,000	4,653	313,848	599,267	104,415	199,576	19,300,458
R-MAT (0.9)	98,438	500,000	3,038	244,204	598,438	107,770	196,873	8,633,841

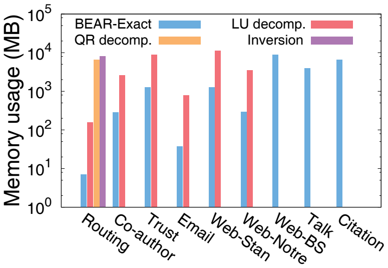


Figure 5: Space for preprocessed data. BEAR-Exact requires the least amount of space for preprocessed data on all the datasets. Due to its space efficiency, only BEAR-Exact successfully scales to the largest Citation dataset (with 3.8M nodes) without running out of memory.

Parameters. We set the restart probability c to 0.05 as in the previous work [42][3]. We set k of SlashBurn to $0.001n$ (see Appendix A for the meaning of k), which achieves a good trade-off between running time and reordering quality. The convergence threshold ϵ of the iterative method is set to 10^{-8}, which gives accurate results. For B_LIN and NB_LIN, we use the heuristic decomposition method proposed in their paper, which is much faster with little difference in accuracy compared with SVD in our experiments. The number of partitions in B_LIN, the rank in B_LIN and NB_LIN, and the convergence threshold of RPPR and BRPPR are tuned for each dataset, which are summarized in Table 5 in Appendix D.

4.2 Preprocessing Cost

We compare the preprocessing cost of BEAR-Exact with that of other exact methods. Figures 1(a) and 5 present the preprocessing time and space requirements of the methods except the iterative method, which does not require preprocessing. Only BEAR-Exact successfully preprocesses all the datasets, while others fail due to their high memory requirements.

Preprocessing time is measured in wall-clock time and includes time for SlashBurn (in BEAR-Exact) and community detection (in LU decomposition). BEAR-Exact requires the least amount of time, which is less than an hour, for all the datasets as seen in Figure 1(a). Especially, in the graphs with distinct hub-and-spoke structure, BEAR-Exact is up to 12× faster than the second best one. This fast preprocessing of BEAR-Exact implies that it is better at handling frequently changing graphs.

To compare space efficiency, we measure the amount of memory required for precomputed matrices of each method. The precomputed matrices are saved in the compressed sparse column format [36], which requires memory space proportional to the number of non-zero entries. As seen in Figure 5, BEAR-Exact requires up to 22× less memory space than its competitors in all the datasets, which results in the superior scalability of BEAR-Exact compared with the competitors.

4.3 Query Cost

We compare BEAR-Exact with other exact methods in terms of query time, which means time taken to compute $\mathbf{r}$ for a given seed node. Although time taken for a single query is small compared with preprocessing time, reducing query time is important because many applications require RWR scores for different query nodes (e.g., all nodes) or require the real-time computation of RWR scores for a query node.

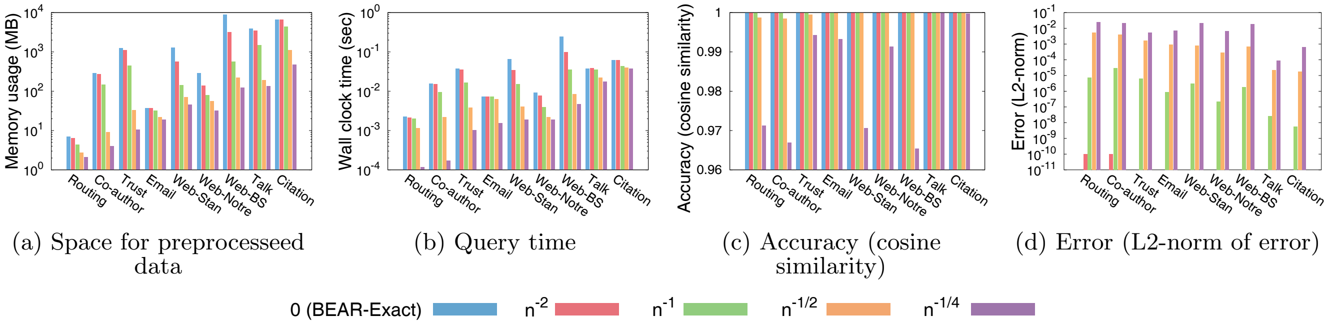
Figure 1(b) shows the result where y axis represents average query time for 1000 random seed nodes. Only BEAR-Exact and the iterative method run successfully on all the graphs, while the others fail on large graphs due to their high space requirements. BEAR-Exact outperforms its competitors in all the datasets except the smallest one, which is the only dataset that the inversion method can scale to. BEAR-Exact is up to 8× faster than LU decomposition, the second best one, and in the Talk dataset, it is almost 300× faster than the iterative method, which is the only competitor. Although BEAR-Exact requires a preprocessing step which is not needed by the iterative method, for real world applications where RWR scores for many query nodes are required, BEAR-Exact outperforms the iterative method in terms of total running time.

Furthermore, BEAR-Exact also provides the best performance in personalized PageRank, where the number of seeds is greater than one. See Figure 11 of Appendix E.1 for the query time of BEAR-Exact with different number of seeds, and see Figure 10 of Appendix E.1 for the comparison of the query time of BEAR-Exact with that of others in personalized PageRank.

4.4 Effects of Network Structure

The complexity analysis in Section 3.3 indicates that the performance of BEAR-Exact depends on the structure of

[3] In this work, c denotes (1 - restart probability)



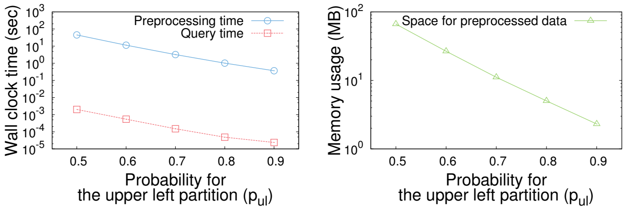
(a) Space for preprocesseed data

(b) Query time

(c) Accuracy (cosine similarity)

(d) Error (L2-norm of error)

Figure 6: Effects of drop tolerance on the performance of BEAR-Approx. Drop tolerance changes from 0 to $n^{-1/4}$. As drop tolerance increases, space requirements and query time are significantly improved, while accuracy, measured by cosine similarity and L2-norm of error, remains high.



(a) Preprocessing time and Query time

(b) Space for preprocessed data

Figure 7: Effects of network structure. BEAR-Exact becomes fast and space-efficient on a graph with distinct hub-and-spoke structure (a graph with high p_{ul}).

a given graph. Specifically, the analysis implies that BEAR-Exact is fast and space-efficient on a graph with strong hub-and-spoke structure where the graph is divided into small pieces by removing a small number of hubs. In this experiment, we empirically support this claim using graphs with similar sizes but different structures.

Table 4 summarizes four synthetic graphs generated using R-MAT [12] with different p_{ul}, the probability of an edge falling into the upper-left partition. As p_{ul} increases, hub-and-spoke structure becomes stronger as seen from the number of hubs (n_2) and the size of partitions ($\sum_{i=1}^{b} n_{1i}^2$) of each graph. Figure 7 shows the performance of BEAR-Exact on theses graphs. Preprocessing time, query time, and space required for preprocessed data decline rapidly with regard to p_{ul}, which is coherent with what the complexity analysis implies.

4.5 Effects of Drop Tolerance

As explained in Section 3.1.4, BEAR-Approx improves the speed and space efficiency of BEAR-Exact by dropping near-zero entries in the precomputed matrices although it loses the guarantee of accuracy. In this experiment, we measure the effects of different drop tolerance values on the query time, space requirements, and accuracy of BEAR-Approx. We change the drop tolerance, ξ, from 0 to $n^{-1/4}$ and measure the accuracy using cosine similarity[4] and L2-norm of error[5] between $\mathbf{r}$ computed by BEAR-Exact and $\hat{\mathbf{r}}$ computed by BEAR-Approx with the given drop tolerance.

Figure 6 summarizes the results. We observe that both the space required for preprocessed data and the query time of BEAR-Approx significantly decrease compared with those of BEAR-Exact, while the accuracy remains high. When ξ is set to n^{-1}, BEAR-Approx requires up to 16× less space and 7× less query time than BEAR-Exact, while cosine similarity remains higher than 0.999 and L2-norm remains less than 10^{-4}. Similarly, when ξ is set to $n^{-1/4}$, BEAR-Approx requires up to 117× less space and 90× less query time than BEAR-Exact, while cosine similarity remains higher than 0.96 and L2-norm remains less than 0.03.

4.6 Comparison with Approximate Methods

We conduct performance comparison among BEAR-Approx and other state-of-the-art approximate methods. Dropping near-zero entries of precomputed matrices is commonly applied to BEAR-Approx, B_LIN, and NB_LIN, and provides a trade-off between accuracy, query time, and storage cost. We analyze this trade-off by changing drop tolerance from 0 to $n^{-1/4}$. Likewise, we analyze the trade-off between accuracy and time provided by RPPR and BRPPR by changing the threshold to expand nodes, θ_b, from 10^{-4} to 0.5. RPPR and BRPPR do not require space for preprocessed data. Accuracy is measured using cosine similarity[4] and L2-norm of error[5] as in Section 4.5.

Figure 8 summarizes the result on two datasets. In Figure 8(a), the points corresponding to BEAR-Approx lie in the upper-left zone, indicating that it provides a better trade-off between accuracy and time. BEAR-Approx with $\xi = n^{-1/4}$ achieves 254× speedup in query time compared with other methods with the similar accuracy. It also preserves accuracy (> 0.97), while other methods with similar query time produce almost meaningless results.

BEAR-Approx also provides the best trade-off between accuracy and space requirements among preprocessing methods as seen in Figure 8(b). BEAR-Approx with $\xi = n^{-1/4}$ saves 50× on storage compared with B_LIN with $\xi = 0$ while providing higher accuracy (> 0.97). NB_LIN with $\xi = 0$ requires 7× more space compared with BEAR-Approx with $\xi = n^{-1/4}$ despite its lower accuracy (< 0.71).

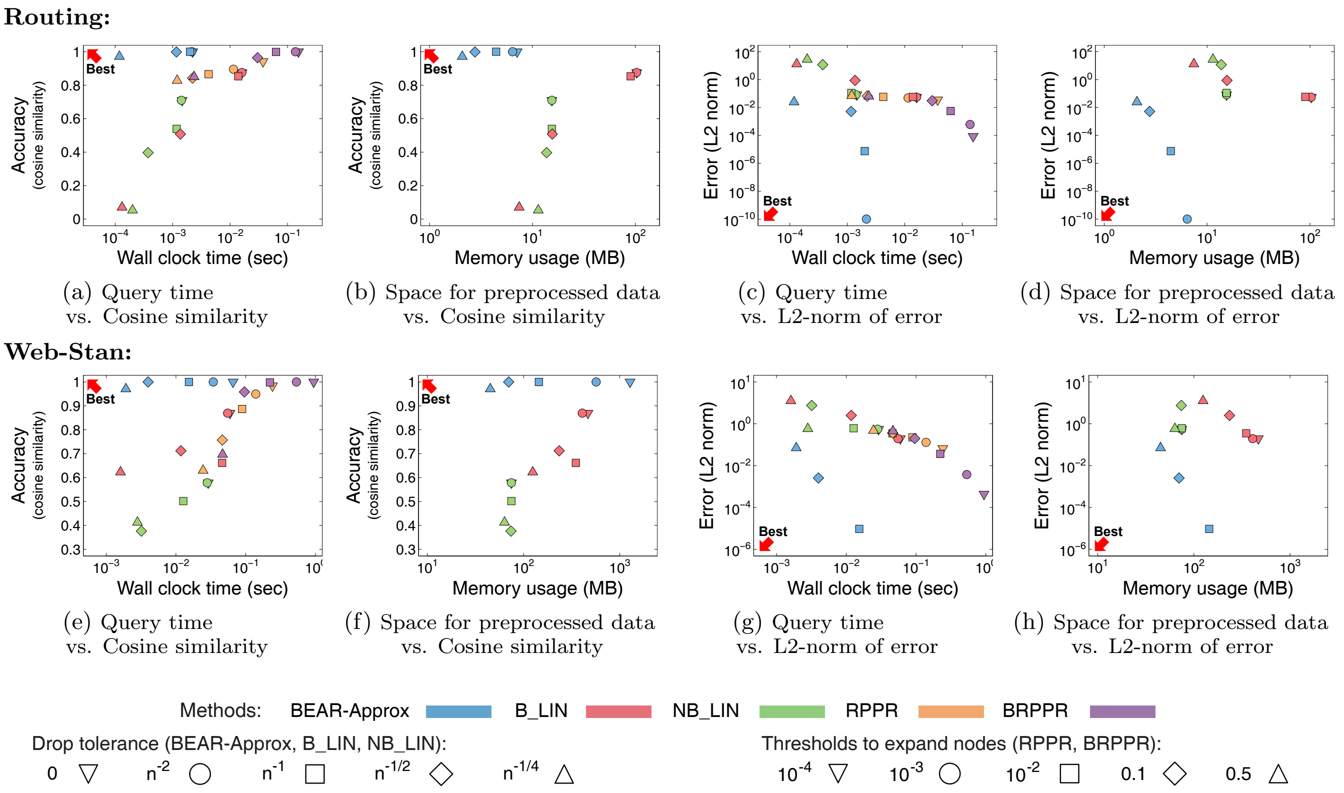
Experiments using L2-norm (Figures 8(c) and 8(d)) show similar tendencies. BEAR-Approx lies in the lower-left zone, which indicates that it provides the best trade-off. The results on other datasets are given in Figure 13 of Appendix E.3. The preprocessing times of the approximate methods are also compared in Figure 12 of Appendix E.2.

5. RELATED WORK

In this section, we review related work, which can be categorized into four parts: (1) relevance measures and applica-

[4] $(\mathbf{r}\cdot\hat{\mathbf{r}})/(||\mathbf{r}||\,||\hat{\mathbf{r}}||)$, ranging from -1 (dissimilar) to 1 (similar)

[5] $||\hat{\mathbf{r}} - \mathbf{r}||$, ranging from 0 (no error) to $||\hat{\mathbf{r}}|| + ||\mathbf{r}||$ (max error bound)



(a) Query time vs. Cosine similarity

(b) Space for preprocessed data vs. Cosine similarity

(c) Query time vs. L2-norm of error

(d) Space for preprocessed data vs. L2-norm of error

(e) Query time vs. Cosine similarity

(f) Space for preprocessed data vs. Cosine similarity

(g) Query time vs. L2-norm of error

(h) Space for preprocessed data vs. L2-norm of error

Figure 8: Trade-off between time, space, and accuracy provided by approximate methods. The above four subfigures show the results on the Routing dataset, and the below ones show the results on the Web-Stan dataset. The colors distinguish the methods, and the shapes distinguish the drop tolerance (for BEAR-Approx, B_LIN, and NB_LIN) or the threshold to expand nodes (for RPPR and BRPPR). RPPR and BRPPR do not appear in the subfigures in the second and fourth columns because they do not require space for preprocessed data. In the left four subfigures, upper-left region indicates better performance, while in the right ones, lower-left region indicates better performance. Notice that BEAR-Approx provides 1) the best trade-off between accuracy and query speed, and 2) the best trade-off between accuracy and space requirements, among preprocessing methods.

tions, (2) approximate methods for RWR, (3) top-k search for RWR, and (4) preprocessing methods for RWR.

Relevance measures and applications. There are various relevance (proximity) measures based on random walk, e.g., penalized hitting probability (PHP) [45], effective importance (EI) [9], discounted hitting time (DHT) [38], truncated hitting time (THT) [37], random walk with restart (RWR) [34], effective conductance (EC) [15], ObjectRank [7], SimRank [22], and personalized PageRank (PPR) [33]. Among these measures, RWR has received much interests and has been applied to many applications including community detection [3, 19, 43, 46], ranking [42], link prediction [6], graph matching [41, 25], knowledge discovery [26], anomaly detection [39], content based image retrieval [21], and cross modal correlation discovery [34]. Andersen et al. [3] proposed a local community detection algorithm which utilizes RWR to find a cut with small conductance near a seed node. This algorithm was used to explore the properties of communities in large graphs since it is fast and returns tight communities [28]. Considerable improvements of the algorithm have been made regarding seed finding [19, 43] and inner connectivity [46]. Backstrom et al. [6] proposed a link prediction algorithm called supervised random walk, which is a variant of RWR. In the algorithm, transition probabilities are determined as a function of the attributes of nodes and edges, and the function is adjusted through supervised learning. Tong et al. [41] proposed G-Ray, which finds best-effort subgraph matches for a given query graph in a large and attributed graph. They formulated a goodness function using RWR to estimate the proximity between a node and a subgraph. Kasneci et al. [26] exploited RWR as a measure of node-based informativeness to extract an informative subgraph for given query nodes. Sun et al. [39] utilized RWR for neighborhood formulation and abnormal node detection in bipartite graphs. He et al. [21] employed RWR to rank retrieved images in their manifold ranking algorithm. Pan et al. [34] proposed a method adopting RWR to compute the correlations between a query image node and caption nodes.

Approximate methods for RWR. The iterative method, which comes from the definition of RWR, is not fast enough in real world applications where RWR scores for different query nodes need to be computed. To overcome this obstacle, approximate approaches have been proposed. Sun et al. [39] observed that the relevance scores for a seed node are highly skewed, and many real-world graphs have a block-wise structure. Based on these observations, they performed random walks only on the partition containing the seed node and assigned the proximities of zero to the other nodes outside the partition. Instead of using a precomputed partition, Gleich et al. [18] proposed methods which adaptively determine the part of a graph used for RWR computation in the query phase as explained in Section 2.2. Andersen et al. [3] also proposed an approximate method based on local information. This method, however, is only applicable to undirected graphs. Tong et al. [42] proposed approximate approaches called B_LIN and NB_LIN. They achieved higher accuracy than previous methods by applying a low-rank approximation to cross-partition edges instead of ignoring them. However, our proposed BEAR-Approx outperforms them by providing a better trade-off between accuracy, time, and space.

Top-k search for RWR. Several recent works focus on finding the k most relevant nodes of a seed node instead of calculating the RWR scores of every node. Gupta et al. [20] proposed the basic push algorithm (BPA), which finds top-k nodes for personalized PageRank (PPR) in an efficient way. BPA precomputes relevance score vectors w.r.t. hub nodes and uses them to obtain the upper bounds of PPR scores. Fujiwara et al. [16] proposed K-dash, which computes the RWR scores of top-k nodes exactly. It computes the RWR scores efficiently by exploiting precomputed sparse matrices and pruning unnecessary computations while searching for the top-k nodes. Wu et al. [44] showed that many random walk based measures have the no-local-minimum (or no-local-maximum) property, which means that each node except for a given query node has at least one neighboring node having lower (or higher) proximity. Based on this property, they developed a unified local search method called Fast Local Search (FLoS), which exactly finds top-k relevant nodes in terms of measures satisfying the no-local-optimum property. Furthermore, Wu et al. showed that FLoS can be applied to RWR, which does not have the no-local-optimum property, by utilizing its relationship with other measures. However, top-k RWR computation is insufficient for many data mining applications [3, 6, 19, 21, 39, 41, 43, 46]; on the other hand, BEAR computes the RWR scores of all nodes.

Preprocessing methods for RWR. As seen from Section 2.2, the computational cost of RWR can be significantly reduced by precomputing $\mathbf{H}^{-1}$. However, matrix inversion does not scale up. That is, for a large graph, it often results in a dense matrix that cannot fit to memory. For this reason, alternative methods have been proposed. NB_LIN, proposed by Tong et al. [41], decomposes the adjacency matrix using low-rank approximation in the preprocessing phase and approximates $\mathbf{H}^{-1}$ from these decomposed matrices in the query phase using Sherman-Morrison Lemma [35]. Its variant B_LIN uses this technique only for cross-partition edges. These methods require less space but do not guarantee accuracy. Fujiwara et al. inverted the results of LU decomposition [16] or QR decomposition [17] of $\mathbf{H}$ after carefully reordering nodes. Their methods produce sparser matrices that can be used in place of $\mathbf{H^{-1}}$ in the query phase but still have limited scalability. Contrary to the earlier methods, our proposed BEAR-Exact offers both accuracy and space efficiency.

In addition to the approaches described above, distributed computing is another promising approach. Andersen et al. [4] proposed a distributed platform to solve linear systems including RWR. In order to reduce communication cost, they partitioned a graph into overlapping clusters and assigned each cluster to a distinct processor.

6. CONCLUSION

In this paper, we propose BEAR, a novel algorithm for fast, scalable, and accurate random walk with restart computation on large graphs. BEAR preprocesses the given graph by reordering and partitioning the adjacency matrix and uses block elimination to compute RWR from the preprocessed results. We discuss the two versions of BEAR: BEAR-Exact and BEAR-Approx. The former guarantees accuracy, while the latter is faster and more space-efficient with little loss of accuracy. We experimentally show that the preprocessing phase of the exact method BEAR-Exact takes up to 12× less time and requires up to 22× less memory space than that of other preprocessing methods guaranteeing accuracy, which makes BEAR-Exact enjoy the superior scalability. BEAR-Exact also outperforms the competitors in the query phase: it is up to 8× faster than other preprocessing methods, and in large graphs where other preprocessing methods run out of memory, is almost 300× faster than its only competitor, the iterative method. The approximate method BEAR-Approx consistently provides a better trade-off between accuracy, time, and storage cost compared with other approximate methods. Future research directions include extending BEAR to support frequently changing graphs [24].

Acknowledgment

This work was supported by the IT R&D program of MSIP/IITP [10044970, Development of Core Technology for Human-like Self-taught Learning based on Symbolic Approach], and Basic Science Research Program through the NRF of Korea funded by the MSIP [2013005259].

7. REFERENCES

[1] L. A. Adamic and E. Adar. Friends and neighbors on the web. *Social Networks*, 25(3):211–230, 2003.

[2] R. Albert, H. Jeong, and A.-L. Barabasi. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, 2000.

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

[4] R. Andersen, D. F. Gleich, and V. Mirrokni. Overlapping clusters for distributed computation. In *WSDM*, pages 273–282, 2012.

[5] 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.

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

[7] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[8] S. Banerjee and A. Roy. *Linear Algebra and Matrix Analysis for Statistics*. CRC Press, 2014.

[9] P. Bogdanov and A. Singh. Accurate and scalable nearest neighbors in large networks based on effective importance. In *CIKM*, pages 1009–1018, 2013.

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

[11] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully automatic cross-associations. In *KDD*, pages 79–88, 2004.

[12] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. In *SDM*, volume 4, pages 442–446, 2004.

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

[14] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *KDD*, pages 219–228, 2009.

[15] P. G. Doyle and J. L. Snell. Random walks and electric networks. *AMC*, 10:12, 1984.

[16] 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.

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

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

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

[20] M. Gupta, A. Pathak, and S. Chakrabarti. Fast algorithms for topk personalized pagerank queries. In *WWW*, pages 1225–1226, 2008.

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

[22] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *KDD*, pages 538–543, 2002.

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

[24] U. Kang and C. Faloutsos. Big graph mining: algorithms and discoveries. *SIGKDD Explorations*, 14(2):29–36, 2012.

[25] U. Kang, H. Tong, and J. Sun. Fast random walk graph kernel. 2012.

[26] G. Kasneci, S. Elbassuoni, and G. Weikum. Ming: mining informative entity relationship subgraphs. In *CIKM*, pages 1653–1656, 2009.

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

[28] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[29] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *J. Amer. Soc. Inf. Sci. and Tech*, 58(7):1019–1031, 2007.

[30] 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.

[31] Z. Lin, M. R. Lyu, and I. King. Matchsim: a novel neighbor-based similarity measure with maximum neighborhood matching. In *CIKM*, pages 1613–1616, 2009.

[32] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *NIPS*, 2:849–856, 2002.

[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] W. W. Piegorsch and G. Casella. Inverting a sum of matrices. *SIAM Review*, 32(3):470–470, 1990.

[36] W. H. Press. *Numerical recipes 3rd edition: The art of scientific computing.* Cambridge university press, 2007.

[37] P. Sarkar and A. W. Moore. A tractable approach to finding closest truncated-commute-time neighbors in large graphs. In *UAI*, pages 335–343, 2007.

[38] P. Sarkar and A. W. Moore. Fast nearest-neighbor search in disk-resident graphs. In *KDD*, pages 513–522, 2010.

[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, B. Gallagher, and T. Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *KDD*, pages 737–746, 2007.

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

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

[44] 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.

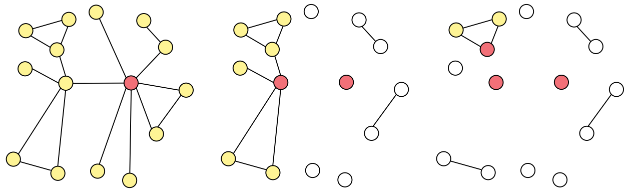
[45] C. Zhang, L. Shou, K. Chen, G. Chen, and Y. Bei. Evaluating geo-social influence in location-based social networks. In *CIKM*, pages 1442–1451, 2012.

[46] Z. A. Zhu, S. Lattanzi, and V. Mirrokni. A local algorithm for finding well-connected clusters. In *ICML*, pages 396–404, 2013.

APPENDIX

A. DETAILS OF SLASHBURN

SlashBurn [23, 30] is a node reordering method for a graph so that the nonzeros of the resulting adjacency matrix are concentrated. For the purpose, SlashBurn first removes k hub nodes (high-degree nodes) from the graph so that the graph is divided into the giant connected component (GCC) and the remaining disconnected components. Then, SlashBurn reorders nodes such that the hub nodes get the highest node ids, the nodes in the disconnected components get the lowest node ids, and the nodes in the GCC get the ids in the middle. The above procedure repeats on the GCC until the size of GCC becomes smaller than k. When SlashBurn finishes, the adjacency matrix of the graph contains a large



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

Figure 9: Hub selection in SlashBurn when $k = 1$. Hub nodes are colored red, nodes in the giant connected component are colored yellow, and nodes in disconnected components are colored white.

and sparse block diagonal matrix in the upper left area, as shown in Figure 4(b). Figure 9 illustrates this process when $k = 1$.

B. THE PROOF OF LEMMA 1

PROOF. Let $\mathbf{L}'$ and $\mathbf{U}'$ be the block-diagonal matrices that consist of $\mathbf{L_1}$ through $\mathbf{L_b}$ and $\mathbf{U_1}$ through $\mathbf{U_b}$, respectively. Then, $\mathbf{L} = \mathbf{L}'$ and $\mathbf{U} = \mathbf{U}'$ because: (1) $\mathbf{L}'$ is a unit lower triangular matrix; (2) $\mathbf{U}'$ is an upper triangular matrix; (3) $\mathbf{L}'\mathbf{U}'$ is the block-diagonal matrix that consists of $\mathbf{L_1U_1}$ through $\mathbf{L_bU_b}$ which is equal to $\mathbf{A}$; and (4) $\mathbf{L}'$ and $\mathbf{U}'$ satisfying (1)$\sim$(3) are the unique LU decomposition of $\mathbf{A}$ [8]. The multiplication of $\mathbf{L}$ and $\mathbf{L^{-1}}$ defined in Lemma 1 results in the block-diagonal matrix that consist of $\mathbf{L_1L_1^{-1}}$ through $\mathbf{L_bL_b^{-1}}$ which is an identical matrix. Likewise, the multiplication of $\mathbf{U}$ and $\mathbf{U^{-1}}$ defined in Lemma 1 results in an identical matrix. □

C. EXPERIMENTAL DATASETS

Below, we provide a short description of the real-world datasets used in Section 4.

- **Routing**[6]**.** The structure of the Internet at the level of autonomous systems. This data was reconstructed from BGP tables collected by the University of Oregon Route Views Project[7].
- **Co-author**[8]**.** The co-authorship network of scientists who posted preprints on the Condensed Matter E-Print Archive[9]. This data include all preprints posted between Jan. 1, 1995 and June. 30, 2003.
- **Trust**[10]**.** A who-trust-whom online social network taken from Epinions.com[11], a general consumer review site. Members of the site decide whether to trust each other, and this affects which reviews are shown to them.
- **Email**[12]**.** An email network taken from a large European research institution. This data include all incoming and outgoing emails of the research institution from October 2003 to May 2005.
- **Web-Notre**[13]**.** The hyperlink network of web pages from University of Notre Dame in 1999.
- **Web-Stan**[14]**.** The hyperlink network of web pages from Stanford University in 2002.
- **Web-BS**[15]**.** The hyperlink network of web pages from University of California, Berkeley and Stanford University in 2002.
- **Talk**[16]**.** A who-talks to-whom network taken from Wikipedia[17], a free encyclopedia written collaboratively by volunteers around the world. This data include all the users and discussion (talk) from the inception of Wikipedia to January 2008.
- **Citation**[18]**.** The citation network of utility patents granted in U.S. between 1975 and 1999.

D. PARAMETER SETTINGS FOR B_LIN, NB_LIN, RPPR, AND BRPPR

Table 5: Parameter values of B_LIN, NB_LIN, RPPR, and BRPPR used for each dataset. $\#p$ denotes the number of partitions, t denotes the rank, and ϵ denotes the convergence threshold.

dataset	B_LIN		NB_LIN	RPPR	BRPPR
	$\#p$	t	t	ϵ	ϵ
Routing	200	200	100	10^{-4}	10^{-4}
Co-author	200	500	1,000	10^{-4}	10^{-4}
Trust	100	200	1,000	10^{-4}	10^{-5}
Email	1,000	100	200	10^{-3}	10^{-5}
Web-Stan	1,000	100	100	10^{-3}	10^{-4}
Web-Notre	500	100	200	10^{-4}	10^{-5}
Web-BS	2,000	100	100	10^{-3}	10^{-5}
Talk	-	-	200	10^{-3}	10^{-6}
Citation	-	-	100	10^{-4}	10^{-5}

For each dataset, we determine the number of partitions ($\#p$) and the rank (t) of B_LIN among $\{100, 200, 500, 1000, 2000\}$ and $\{100, 200, 500, 1000\}$, resp., so that their pair provides the best trade-off between query time, space for preprocessed data, and accuracy. Likewise, the rank (t) of NB_LIN is determined among $\{100, 200, 500, 1000\}$, and the convergence threshold (ϵ) of RPPR and BRPPR is determined among $\{10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$. The parameter values used for each dataset are summarized in Table 5. B_LIN runs out of memory on the Talk dataset and the Citation dataset regardless of parameter values used.

E. ADDITIONAL EXPERIMENTS

E.1 Query Time in Personalized PageRank

We measure the query time of exact methods when the number of seeds is greater than one, which corresponds to personalized PageRank. We change the number of seeds (non-zero entries in $\mathbf{q}$) from 1 to 1000. As seen in Figure 10, BEAR-EXACT is the fastest method regardless of the number of seeds in all the datasets except the smallest Routing dataset The query time of the inverse method increases rapidly as the number of seeds increases. It increases by 210× on the Routing dataset, while the query time of BEAR-EXACT increases by 3×. Figure 11 summarizes the effect of the number of seeds on the query time of

[6] http://www-personal.umich.edu/~mejn/netdata/as-22july06.zip

[7] http://www.routeviews.org/

[8] http://www-personal.umich.edu/~mejn/netdata/cond-mat-2003.zip

[9] http://arxiv.org/archive/cond-mat

[10] http://snap.stanford.edu/data/soc-sign-epinions.html

[11] http://www.epinions.com/

[12] http://snap.stanford.edu/data/email-EuAll.html

[13] http://snap.stanford.edu/data/web-NotreDame.html

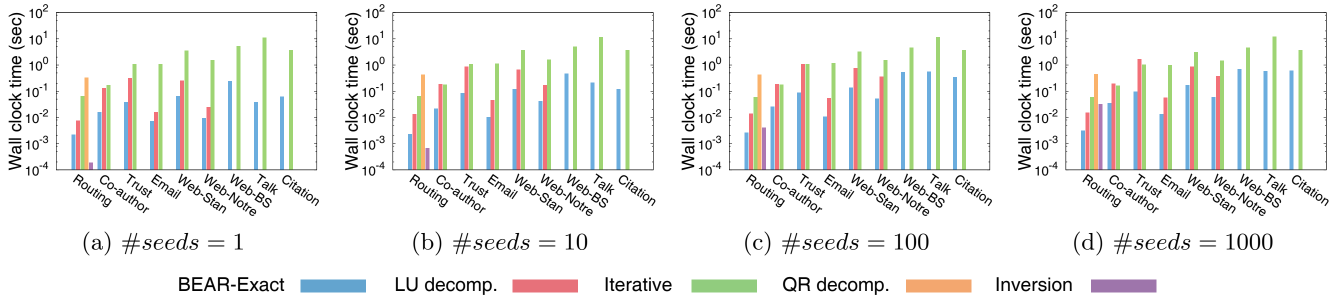
[14] http://snap.stanford.edu/data/web-Stanford.html

[15] http://snap.stanford.edu/data/web-BerkStan.html

[16] http://snap.stanford.edu/data/wiki-Talk.html

[17] http://www.wikipedia.org/

[18] http://snap.stanford.edu/data/cit-Patents.html



(a) #*seeds* = 1 (b) #*seeds* = 10 (c) #*seeds* = 100 (d) #*seeds* = 1000

Figure 10: Query time of exact methods with different number of seeds. If a method cannot scale to a dataset, the corresponding bar is omitted in the graphs. BEAR-Exact is the fastest method regardless of the number of seeds in all the datasets except the smallest Routing dataset.

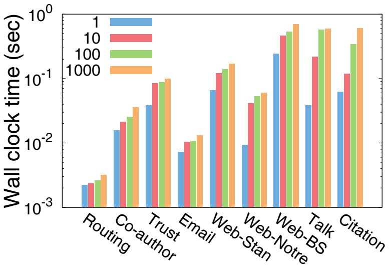


Figure 11: Effect of the number of seeds on the query time of BEAR-Exact. Query time increases as the number of seeds increases, but the rate of increase slows down.

BEAR-Exact. In most datasets, the effect of the number of seeds diminishes as the number of seeds increases. The query time increases by up to 16× depending on the number of seeds.

E.2 Preprocessing Time of Approximate Methods

Figure 12 presents the preprocessing time of approximate methods. Preprocessing time is measured in wall-clock time and includes time for SlashBurn (in BEAR-Approx) and community detection (in B_LIN). B_LIN cannot scale to the Talk dataset and the Citation dataset because it runs out of memory while inverting the block diagonal matrices. The relative performances among the methods depend on the structure of graphs, as summarized in Table 4. BEAR-Approx tends to be faster than the competitors on graphs with a small number of hubs (such as the Routing and Email datasets) and slower on those with a large number of hubs (such as the Web-BS and Citation datasets).

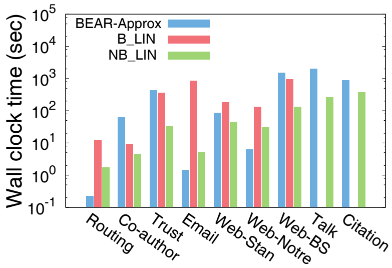
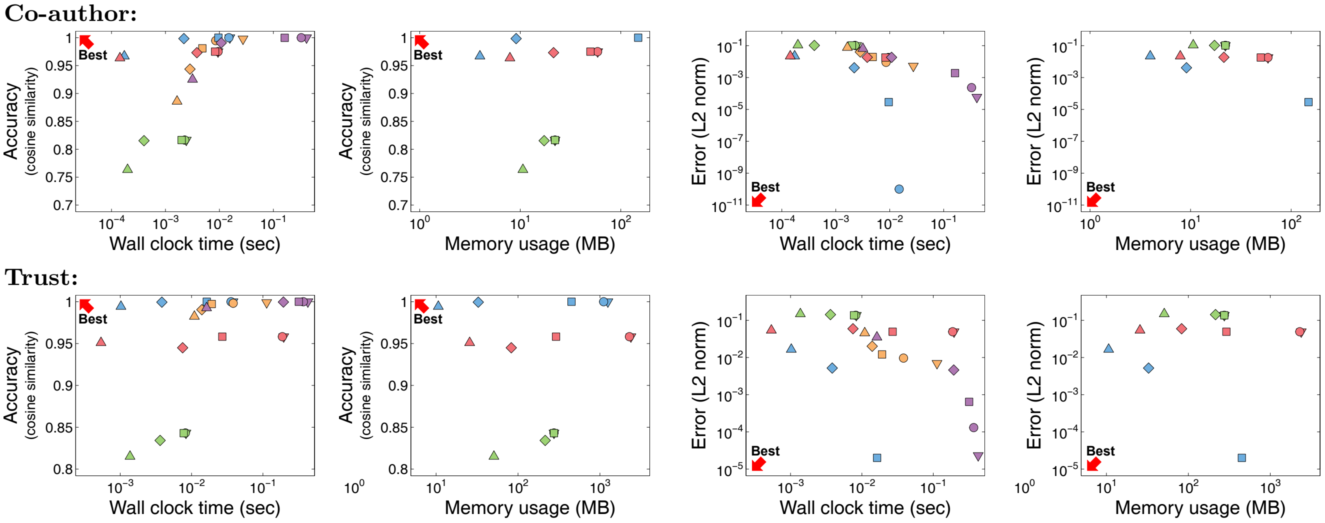


Figure 12: Preprocessing time of approximate methods. The relative performances among the methods depend on the characteristics of data.

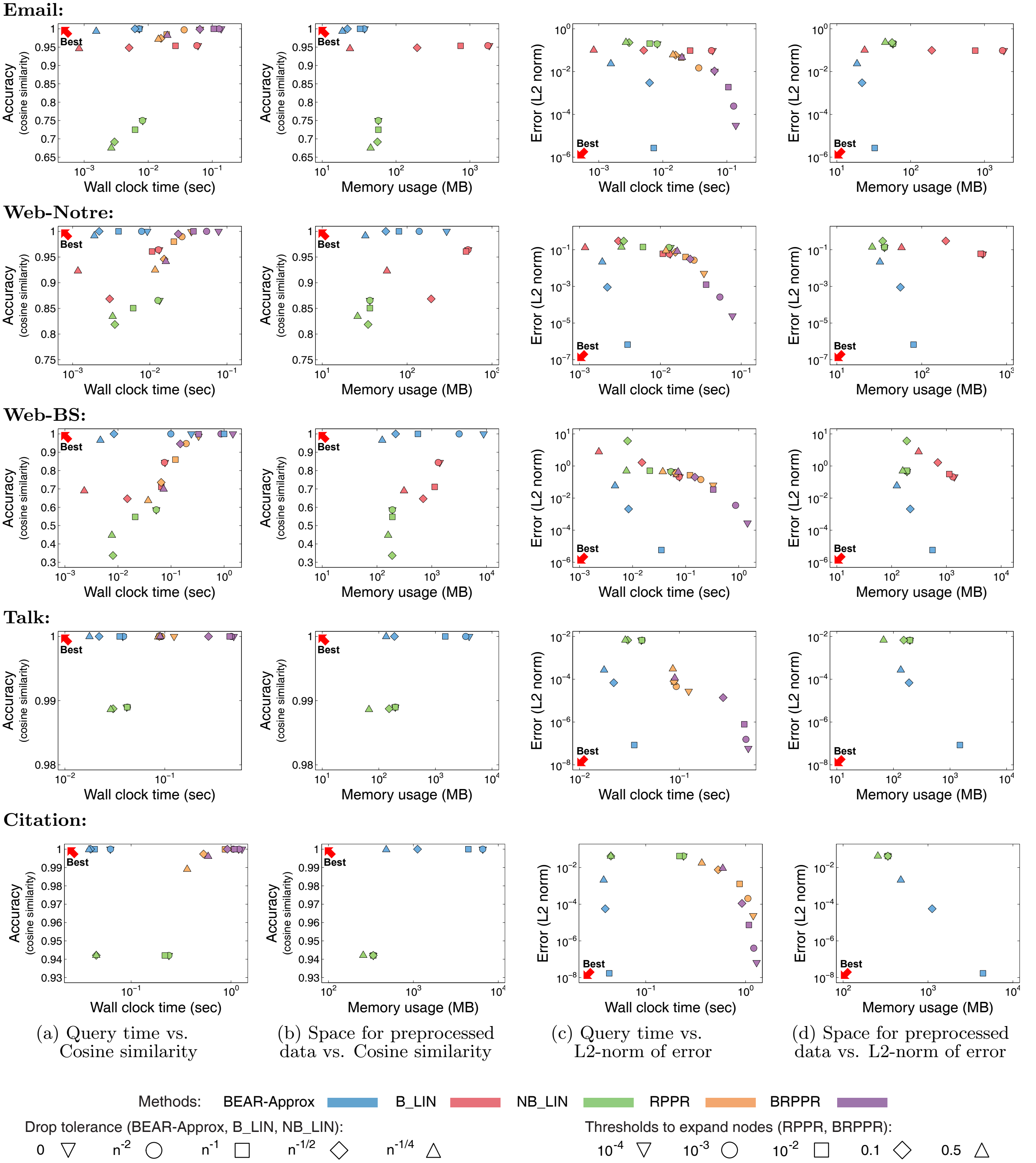
E.3 Comparison with Approximate Methods

Figure 13 shows the result of the experiments described in Section 4.6 on other datasets. In most of the datasets, BEAR-Approx gives the best trade-off between accuracy and time. It also provides the best trade-off between accuracy and storage among all preprocessing methods.


Co-author:

Trust:

(continues on the next page)



(a) Query time vs. Cosine similarity

(b) Space for preprocessed data vs. Cosine similarity

(c) Query time vs. L2-norm of error

(d) Space for preprocessed data vs. L2-norm of error

Figure 13: Trade-off between time, space, and accuracy provided by approximate methods. The subfigures in each row show the results on the corresponding dataset. The colors distinguish the methods, and the shapes distinguish the drop tolerance (for BEAR-Approx, B_LIN, and NB_LIN) or the threshold to expand nodes (for RPPR and BRPPR). RPPR and BRPPR do not appear in the subfigures in the second and fourth columns because they do not require space for preprocessed data. In the subfigures in the left two columns, upper-left region indicates better performance, while in the subfigures in the right two columns, lower-left region indicates better performance. BEAR-Approx is located more closely to those regions than the competitors in most of the datasets.