Efficient Personalized PageRank with Accuracy Assurance

Yasuhiro Fujiwara†, Makoto Nakatsuji‡, Takeshi Yamamuro†,
Hiroaki Shiokawa†, Makoto Onizuka†
†NTT Software Innovation Center, ‡NTT Service Evolution Laboratories
†3-9-1 Midori-cho Musashino-shi, Tokyo, Japan, ‡1-1 Hikarinooka Yokosuka-shi, Kanagawa, Japan
{fujiwara.yasuhiro, nakatsuji.makoto, yamamuro.takeshi,
shiokawa.hiroaki, onizuka.makoto}@lab.ntt.co.jp

ABSTRACT

Personalize PageRank (PPR) is an effective relevance (proximity) measure in graph mining. The goal of this paper is to efficiently compute single node relevance and top-k/highly relevant nodes without iteratively computing the relevances of all nodes. Based on a "random surfer model", PPR iteratively computes the relevances of all nodes in a graph until convergence for a given user preference distribution. The problem with this iterative approach is that it cannot compute the relevance of just one or a few nodes. The heart of our solution is to compute single node relevance accurately in non-iterative manner based on sparse matrix representation, and to compute top-k/highly relevant nodes exactly by pruning unnecessary relevance computations based on upper/lower relevance estimations. Our experiments show that our approach is up to seven orders of magnitude faster than the existing alternatives.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data mining

General Terms

Algorithms, Theory

Keywords

Personalized PageRank, Relevance computation

1. INTRODUCTION

Graphs arise in a wide range of application domains such as the Internet, social networks, biological networks and more [28]. In these graphs, personalized search has become an increasingly important topic in recent years [18]. One of the most important aspects of personalized search is to measure the relevance (proximity) of nodes according to the user preference distribution[1]. Examples of personalized search are "To what degree is a web page relevant to user's bookmark pages?", "Which books have the top ten relevance scores with a view to being recommended to a user in accordance with his/her shopping logs?", and so on.

Many relevance measures have been reported in the literature [5, 13, 14, 15, 21, 22]. Among the relevance measures, *personalized PageRank (PPR)* [10, 23] has attracted considerable attention due to its effectiveness and solid theoretical foundation. Similar to PageRank [1, 8, 27, 30], PPR is defined with respect to a "random surfer model" and needs to be computed iteratively. Informally speaking, PPR is the stationary distribution of random walks; at each step in a random walk, it randomly selects an outgoing edge from the current node, and, with a certain probability, it jumps to a node in accordance with a given preference distribution.

However, this iterative approach has a serious weakness: it is not effective if the user needs only single node relevance or top-k/highly relevant nodes. This is because its iterative computation approach updates the relevance scores of all nodes that are determined in the previous iteration. If m and t are the number of edges and the number of iteration steps until convergence, respectively, it requires $O(mt)$ time; its computational cost is excessive for large graphs.

1.1 Problem motivation

We address three problems. With our approach, many applications can be processed more efficiently without sacrificing application efficacy. The first problem is to compute the PPR relevance score of a single given node for a preference distribution. This problem is formalized as follows:

Problem 1 (Single-node relevance computation).
Given: *Query node x and query distribution* $\mathbf{d}$.
Find: *Relevance score of node x according to query distribution* $\mathbf{d}$.

The second problem is to identify the top-k relevant nodes for a given preference distribution.

Problem 2 (Top-k nodes identification).
Given: *The number of required answer nodes, K, and query distribution* $\mathbf{d}$.
Find: *Top K nodes with the highest relevance scores with respect to query distribution* $\mathbf{d}$.

We also solve the problem of finding all nodes whose relevance scores are higher than a given threshold:

Problem 3 (Highly relevant node detection).
Given: *The threshold, ϵ, and query distribution* $\mathbf{d}$.
Find: *Nodes whose relevance scores are higher than ϵ with respect to query distribution* $\mathbf{d}$.

These three problems have to be resolved to realize the following applications.

[1] Called as "preference vector" in [23].

KDD'12, August 12–16, 2012, Beijing, China.

Graph analysis (Problem 1). In graph analysis, there is a need for a tool that can analyze the relative importance of each node in a graph with respect to user preferences [22]. This tool reveals the context of interactive exploration of graph data. The PPR-based approach, suggested by White et al., can effectively measure the relative importance of each node in accordance with user preference [36]. By picking up several researchers in a co-authorship network who belong to the same university, this approach reveals that they have low importance scores if they study different research disciplines.

Recommendation (Problem 2). Recommendation systems aim to provide personalized items that are interesting for users. A popular approach is to use the collaborative filtering technique [20]. However, this technique does not directly utilize information about latent user interests in recommendations. PPR-based methods can be used to capture user's possible interest expansion [26]. This method uses PPR to expand the user's latent interests estimated by LDA [6]. And it computes the top-k relevant items for the recommendations. This method outperforms previous approaches such as ItemRank [15] and L^+ [11].

Link prediction (Problem 3). The link prediction problem, attempting to infer which new interactions among social network members are likely to occur in the near future, is an active research topic [37]. The PPR-based approach by Liven-Nowell et al. can answer this problem [25]; the probability of future interactions between members is computed by PPR. If two members have many friends in common, they have high mutual PPR scores, and they are more likely to interact. By identifying those pairs with high PPR scores, this approach provides better link predictions than the random prediction approach.

Our proposed method can also be used in other applications such as person name disambiguation [32], document clustering [2], and protein-protein interaction analysis [35], even though we omit details due to the space limitations.

1.2 Contribution

To the best of our knowledge, our approach is the first solution to comprehensively handle all the three problems that theoretically guarantees exactness unlike previous approximate approaches [3, 4, 10, 17, 23]. To achieve high efficiency, we use a sparse matrix representation and compute single node relevance in a non-iterative manner. Our approach also efficiently identifies the top-k/highly relevant nodes without computing the relevance scores of unnecessary nodes by estimating upper/lower relevance scores. Our approach has the following attractive characteristics:

- **Fast**: The proposed approach, by utilizing the above ideas, is significantly faster than existing approaches.
- **Exact**: Our approach does not sacrifice accuracy; it returns the exact outputs for the three problems.
- **Easy to deploy**: Our approach itself does not require any parameter. Thus it provides the user with a simple solution to PPR-based applications.

While PPR has been used in many applications, it has been difficult to utilize due to its high computational cost. However, by providing exact solutions in a highly efficient manner, the proposed approach will allow many more PPR-based applications to be developed in the future.

The remainder of this paper is organized as follows: Section 2 overviews the background of this paper. Section 3 introduces the details of our approach. Section 4 reviews the experiments conducted and the results gained. Section 5 details case-studies of our approach. Section 6 describes related work. Section 7 provides our brief conclusion.

Table 1: Definition of main symbols.

Symbol	Definition
n	Number of nodes
m	Number of edges
x	Query node for a single node relevance computation
K	Number of answer nodes for top-k identification
ϵ	Threshold in finding highly relevant nodes
c	Restart probability
s_u	Relevance or steady-state probability of node u
$\mathbf{s}$	$n \times 1$ relevance vector
$\mathbf{d}$	$n \times 1$ query distribution vector where $\sum d_i = 1$
$\mathbf{A}$	$n \times n$ column normalized adjacent matrix
$\mathbf{P}$	$n \times n$ permutation matrix
$\mathbf{Q}$	$n \times n$ orthogonal matrix in QR decomposition
$\mathbf{R}$	$n \times n$ upper triangular matrix in QR decomposition
$\lvert\mathbf{R}\rvert$	Number of non-zero elements in matrix $\mathbf{R}$

2. PRELIMINARY

We formally define the notations and introduce the background of this paper. Table 1 lists the main symbols and their definitions. Measuring the relevance scores of nodes according to a preference distribution can be achieved by using PPR. In PPR, starting from one of *seed nodes*[2] whose preference distribution scores are not zero, a random walk is performed by iteratively following an edge to another node at each step. Additionally, at every step, there is a probability, c, of returning to a seed node according to the preference distribution. Let $\mathbf{s}$ be an $n \times 1$ vector where n is the number of nodes, the u-th element s_u denotes the steady-state probability that the random walk is at node u. Let $\mathbf{d}$ be an $n \times 1$ column normalized vector of the preference distribution of seed nodes, i.e. $\sum d_i = 1$, and $\mathbf{A}$ be the $n \times n$ column normalized adjacency matrix of the graph. The steady-state probability for each node can be obtained by recursively applying the following equation until convergence:

$$\mathbf{s} = (1 - c)\mathbf{A}\mathbf{s} + c\mathbf{d} \tag{1}$$

The steady-state probabilities, $\mathbf{s}$, give the long term visit rate of each node given the bias indicated by query distribution $\mathbf{d}$. Therefore, s_u can be considered as the relevance measure of node u with respect to given distribution $\mathbf{d}$.

But this method is not effective to compute the relevance score of a single node or identify the nodes with top-k/high relevance since it recursively updates the relevance scores of all nodes. It needs $O(mt)$ time until convergence. This incurs excessive computational cost for large graphs.

3. PROPOSED METHOD

In this section, we describe the proposed approach. The main advantage of our approach is that it can efficiently compute single node relevance as well as top-k/highly relevant nodes without loss in accuracy.

3.1 Overview

Computing single node relevance. As described in Section 2, exact relevance scores can be obtained by the recursive approach. This approach iteratively updates the relevance scores of all nodes; it cannot compute the relevance of

[2] In the special case of only a single seed node, PPR is equivalent to random walk with restart [13].

just one single node. To avoid iterative processing, we utilize matrix computations and compute single node relevance in non-iterative style. The matrices can be obtained from Equation (1). However, this approach can incur high computational cost if the matrices are dense [31]. We propose an approach to overcome this problem in Section 3.2.

In a precomputation step, we permute nodes of the adjacent matrix, and compute the QR decomposition [19] of the matrix. From the resulting matrices, we then compute the inverse matrices. By properly permuting the nodes, the inverse matrices have a sparse structure. The permutation can be obtained by solving an **NP**-complete problem as described in Section 3.2. Thus, we can compute single node relevance efficiently from the sparse data structure. Since QR decomposition is not an approximate method, we can compute exact relevance scores.

Identifying top-k nodes. When a user queries top-k relevant nodes for a preference distribution, the iterative approach computes the relevance scores of all nodes and finds the answer nodes. However, this approach requires high computational cost. For top-k identification, our approach computes exact relevance scores for just the answer-likely nodes to avoid unnecessary relevance computations as described in Section 3.3.

To find top-k nodes, we estimate the relevance scores of each node at $O(1)$ time. If a node is estimated to be an answer-likely node, we compute its exact relevance. The advantage of this idea is to find top-k nodes exactly although it uses estimations. Exact answer results are obtained by estimating upper bounding relevance scores. This means that we can safely discard unpromising nodes along with their estimated relevance scores at low computational cost.

Finding highly relevant nodes. Although the upper bounding estimation approach can discard most unlikely nodes, it still relies on exact relevance computations to guarantee the exactness of answer results. In section 3.4, we show our approach that reduces the cost of exact computations for finding highly relevant nodes whose scores exceed ϵ.

To find highly relevant nodes, we estimate lower bounding relevance scores as well as upper bounding relevance scores. Our approach to find highly relevant nodes is based on the property that, if the lower bounding relevance of a node is more than ϵ, the exact relevance of the node must be greater than ϵ. Therefore, the node must be an answer node. This idea reinforces the idea of upper bounding estimation by effectively pruning the exact relevance computations. Since the exact relevance computations are limited to a small number, we can efficiently find highly relevant nodes.

3.2 Computing single node relevance

In this section, we describe our approach to efficiently computing the relevance score of a query node for a given distribution. Section 3.2.1 introduces a definition of PPR that allows it to be rewritten as non-iterative matrix computation by using QR decomposition, and then shows our non-iterative approach. In Section 3.2.2, we show that obtaining sparse matrices is an **NP**-complete problem, and we then show our solution for the problem.

3.2.1 Non-iterative relevance computation

We show here how to rewrite Equation (1) into non-iterative matrix form. In our relevance computation, we permute nodes in the graph. That is, let $\mathbf{P}$ be a permutation matrix, the original adjacent matrix $\mathbf{A}$ is transformed into matrix $\mathbf{A}'$ in the form of $\mathbf{A}' = \mathbf{P}\mathbf{A}\mathbf{P}^T$ where matrix $\mathbf{P}^T$ is the transpose of matrix $\mathbf{P}$ [19]. The $n \times n$ permutation matrix $\mathbf{P}$ is an orthogonal matrix where every row and column contains precisely a single 1 with 0s everywhere else, and $P_{ij} = 1$ indicates that j-th row is permutated into the i-th row. Our approach to obtaining matrix $\mathbf{P}$ is described in Section 3.2.2.

By utilizing matrix $\mathbf{P}$, Equation (1) can be rewritten in the following matrix form since $\mathbf{I} = \mathbf{P}^T\mathbf{I}\mathbf{P}$, $\mathbf{P}^{-1} = \mathbf{P}^T$, and $\mathbf{A} = \mathbf{P}^T\mathbf{A}\mathbf{P}$ where $\mathbf{P}^{-1}$ is the inverse matrix of $\mathbf{P}$ [19]:

$$\begin{aligned}\mathbf{s} &= c\{\mathbf{I}-(1-c)\mathbf{A}\}^{-1}\mathbf{d} = c\{\mathbf{P}^T\mathbf{I}\mathbf{P}-(1-c)\mathbf{P}^T\mathbf{A}'\mathbf{P}\}^{-1}\mathbf{d} \\ &= c\mathbf{P}^T\{\mathbf{I}-(1-c)\mathbf{A}'\}^{-1}\mathbf{P}\mathbf{d}\end{aligned} \quad (2)$$

To compute query node relevance, we perform QR decomposition of the matrix $\mathbf{I}-(1-c)\mathbf{A}'$, i.e., $\mathbf{Q}\mathbf{R} = \mathbf{I}-(1-c)\mathbf{A}'$ where matrices $\mathbf{Q}$ and $\mathbf{R}$ are an orthogonal matrix and upper triangular matrix, respectively [19]. Formally, we compute the relevance of node x for the given distribution $\mathbf{d}$ from precomputed $\mathbf{Q}^T$ and $\mathbf{R}^{-1}$ as follows:

Definition 1 (Relevance computation). *Let $n \times n$ matrix $\mathbf{F} = c\mathbf{P}^T\mathbf{R}^{-1}$, $n \times 1$ vector $\mathbf{g} = \mathbf{Q}^T\mathbf{P}\mathbf{d}$ and $1 \times n$ vector $\mathbf{f}_i$ be the i-th row vector in matrix $\mathbf{F}$. The relevance of node x, s_x, is computed as follows:*

$$s_x = \mathbf{f}_x \cdot \mathbf{g} \quad (3)$$

For Definition 1, we have the following theorem:

Theorem 1 (Relevance computation). *Equation (3) outputs exactly the same result as Equation (1).*

Proof From Equation (2), we have the following equation since $\mathbf{Q}^T = \mathbf{Q}^{-1}$ [19]:

$$\mathbf{s} = c\mathbf{P}^T(\mathbf{Q}\mathbf{R})^{-1}\mathbf{P}\mathbf{d} = c\mathbf{P}^T\mathbf{R}^{-1}\mathbf{Q}^T\mathbf{P}\mathbf{d} = \mathbf{F}\cdot\mathbf{g}$$

Therefore, s_x, x-th element of $n \times 1$ column vector $\mathbf{s}$, can be computed as the inner product of vectors $\mathbf{f}_x$ and $\mathbf{g}$. □

Equation (3) implies that we can compute node relevance as the inner product of vectors $\mathbf{f}_x$ and $\mathbf{g}$ in non-iterative style. Equation (3) also implies that, in the relevance computation, we first compute vector $\mathbf{g} = \mathbf{Q}^T\mathbf{P}\mathbf{d}$ from vector $\mathbf{d}$ since it is given by the user.

We provide the following theorem to describe the time complexity of our relevance computation:

Theorem 2 (Relevance computation cost). *Let $|\mathbf{f}_x|$ and $|\mathbf{Q}|$ be the number of non-zero elements in $\mathbf{f}_x$ and $\mathbf{Q}$, respectively, it requires $O(|\mathbf{f}_x| + |\mathbf{Q}|)$ time to compute the relevance score of a single node by our non-iterative approach.*

Proof Our approach computes vector $\mathbf{g}$ to obtain the relevance from the given distribution $\mathbf{d}$. Since vector $\mathbf{P}\mathbf{d}$ is obtained by permutating vector $\mathbf{d}$, it needs $O(|\mathbf{Q}|)$ time to compute vector $\mathbf{g} = \mathbf{Q}^T\mathbf{P}\mathbf{d}$. Since it takes $O(|\mathbf{f}_x| + |\mathbf{Q}|)$ time to compute the inner product of the vectors $\mathbf{f}_x$ and $\mathbf{g}$, it requires $O(|\mathbf{f}_x| + |\mathbf{Q}|)$ time. □

Theorem 2 implies that the numbers of non-zero elements in matrices $\mathbf{R}^{-1}$ and $\mathbf{Q}$ should be reduced to achieve high efficiency since $\mathbf{F} = c\mathbf{P}^T\mathbf{R}^{-1}$ and $\mathbf{g} = \mathbf{Q}^T\mathbf{P}\mathbf{d}$. Note that the number of non-zero elements in $\mathbf{P}$ is trivially n [19]. In particular, the number of non-zero elements in matrix $\mathbf{Q}$ should be reduced since vector $\mathbf{g}$ is computed according to the user's query. In the next section, we show our approach to obtain the sparse matrices $\mathbf{R}^{-1}$ and $\mathbf{Q}$.

3.2.2 Sparse matrix problem

As shown in Definition 1, we can compute relevance scores in non-iterative style if we precompute matrices $\mathbf{R}^{-1}$ and $\mathbf{Q}$. However, this approach can incur high computational cost if the matrices have dense structures even if the original graph has sparse structure. In many real graphs, the number of edges is much smaller than the squared number of nodes, i.e. $m \ll n^2$ [29]. In our approach, we permute nodes in the graph to obtain sparse $\mathbf{R}^{-1}$ and $\mathbf{Q}$. Unfortunately, determining the node permutation to yield the sparse matrices is an **NP**-complete problem.

Theorem 3 (Sparse matrix problem). *Setting the nodes permutation that minimizes non-zero elements in the matrices* $\mathbf{R}^{-1}$ *and* $\mathbf{Q}$ *is* **NP***-complete.*

Proof We prove the theorem by a reduction from the *minimum fill-in* problem [38]. We transform instances of this problem into instances of the *sparse matrices* problem as follows: For the graph of the *minimum fill-in* problem, we create matrix $\mathbf{A}$. For the node elimination ordering, we create the permutation matrix $\mathbf{P}$, and we create matrix $\mathbf{R}^{-1}$ and $\mathbf{Q}$ for the chordal graph. Thus, it is easy to show that there exists a solution to the *minimum fill-in* problem with the minimum number of edge additions iff there exists a solution to the *sparse matrices* problem with the minimum increase in non-zero elements in the inverse matrices. Thus, the *sparse matrices* problem is trivial in **NP**. □

We introduce an approach for the problem. At the beginning, we show how to compute matrices $\mathbf{Q}$ and $\mathbf{R}^{-1}$ to describe our approach in detail. Let $\mathbf{q}_i$ be the i-th column vector of matrix $\mathbf{Q}$ and $\mathbf{w}_i$ be the i-th column vector of matrix $\mathbf{W} = \mathbf{I} - (1-c)\mathbf{A}'$, matrices $\mathbf{Q}$ and $\mathbf{R}$ can be computed by Gram-Schmidt orthogonalization as follows [19]:

$$\mathbf{q}_i = \mathbf{q}'_i / ||\mathbf{q}'_i||,\ \mathbf{q}'_i = \begin{cases} \mathbf{w}_i & (i = 1) \\ \mathbf{w}_i - \sum_{j=1}^{i-1} (\mathbf{w}_i \cdot \mathbf{q}_j)\mathbf{q}_j & (i \neq 1) \end{cases} \quad (4)$$

$$R_{ij} = \begin{cases} 0 & (i > j) \\ ||\mathbf{q}'_i|| & (i = j) \\ \mathbf{w}_j \cdot \mathbf{q}_i & (i < j) \end{cases} \quad (5)$$

where $||\mathbf{q}'_i||$ is the norm of $\mathbf{q}'_i$ [19]. Elements in matrix $\mathbf{R}^{-1}$ can be computed as follows by backward substitution [31]:

$$R^{-1}_{ij} = \begin{cases} 0 & (i > j) \\ 1/R_{ij} & (i = j) \\ -1/R_{ii} \sum_{k=i+1}^{j} R_{ik} R^{-1}_{kj} & (i < j) \end{cases} \quad (6)$$

Equation (4) and (5) imply that column vectors of $\mathbf{Q}$ and $\mathbf{R}$ are obtained from left to right by computing orthogonal column vectors and inner products from matrix $\mathbf{W}$. Equation (6) shows that the column vectors of $\mathbf{R}^{-1}$ are obtained from right to left and elements in column vectors from bottom to top. Equations (4), (5), and (6) indicate that matrices $\mathbf{Q}$ and $\mathbf{R}$ are computed from matrix $\mathbf{W}$ which is obtained from matrix $\mathbf{A}'(= \mathbf{PAP}^T)$.

We can obtain sparse matrix $\mathbf{Q}$ based on the following observation: if column vector $\mathbf{w}_i$ has a single non-zero element whose left elements in the matrix $\mathbf{W}$ are all zero, column vector $\mathbf{w}_i$ is orthogonal to all its left column vectors in matrix $\mathbf{Q}$. This is because the column vector $\mathbf{w}_i$ is linearly independent [19], thus matrix $\mathbf{Q}$ has a sparse structure.

We obtain sparse matrix $\mathbf{R}^{-1}$ based on the following two observations: (1) the right/bottom elements of matrix $\mathbf{R}^{-1}$ would be sparse if the corresponding right/bottom elements of matrix $\mathbf{R}$ are sparse, and (2) the right/bottom elements of matrix $\mathbf{R}$ are expected to be zero if the corresponding right/bottom elements of matrix $\mathbf{W}$ are zero.

Algorithm 1 Node permutation

Input: $\mathbf{A}$, adjacent matrix of the graph; n, the number of nodes
Output: $\mathbf{P}$, permutation matrix
1: $\mathbf{P} = \mathbf{0}$;
2: $\mathcal{P} = \emptyset$;
3: **for** $i = 1$ to n **do**
4: $\quad \mathcal{U} = \text{argmin}(e(u)|u \in \mathcal{V} \backslash \mathcal{P})$;
5: $\quad v = \text{argmax}(deg(u)|u \in \mathcal{U})$;
6: $\quad P_{iv} = 1$;
7: $\quad$ append node v to $\mathcal{P}$;
8: **end for**
9: **return** $\mathbf{P}$;

Algorithm 1 depicts our permutation algorithm for obtaining the sparse matrices. Our algorithm uses the above observations on matrices $\mathbf{Q}$ and $\mathbf{R}^{-1}$. In this algorithm, $\mathcal{V}$ and $\mathcal{P}$ indicate the node set in the graph and the permutated node set, respectively. $deg(u)$ is the number of edges incident to node u, i.e. degree of node u. $e(u)$ is the number of edges that are *not* incident to node set $\mathcal{P}$ from node u. Our algorithm first sets the permutation matrix $\mathbf{P}$ to a zero matrix, and initializes node set $\mathcal{P}$ (lines 1-2). From the unpermutated set of nodes, $\mathcal{V} \backslash \mathcal{P}$, it finds the set of nodes, $\mathcal{U}$, whose numbers of edges are $\min(e(u)|u \in \mathcal{V} \backslash \mathcal{P})$ based on the observation on matrix $\mathbf{Q}$ (line 4). If the right/bottom elements of matrix $\mathbf{A}'$ are zero, matrix $\mathbf{R}^{-1}$ has a sparse structure. Therefore, it identifies node v that has the maximum number of edges to make the left/top elements of matrix $\mathbf{A}'$ dense (line 5). It then sets an element in matrix $\mathbf{P}$ to determine the permutation (lines 6-7). These procedures continue until all nodes are permutated.

This approach can effectively reduce the number of non-zero elements for $\mathbf{R}^{-1}$ and, especially for, $\mathbf{Q}$ as shown in Section 4. Thus, we can efficiently compute a single node relevance score since $\mathbf{F} = c\mathbf{P}^T\mathbf{R}^{-1}$ and $\mathbf{g} = \mathbf{Q}^T\mathbf{Pd}$. This approach has another advantage: we can efficiently precompute matrices $\mathbf{Q}$ and $\mathbf{R}^{-1}$ as demonstrated in Section 4. This is because the efficiency of the precomputation is greatly enhanced by reducing the number of non-zero elements.

3.3 Identifying top-k nodes

We introduce an algorithm that finds top-k nodes by estimating upper bounding scores. We compute the estimations in the process of finding the top-k nodes. Our solution is based on lower bounding estimation which is described in Section 3.3.1. The details of the upper bounding estimation approach are introduced in Section 3.3.2. We show our top-k node identification algorithm in Section 3.3.3.

3.3.1 Lower bounding estimation

So as to find the top-k nodes, we estimate lower bounding relevance scores from the numbers of minimum hops from the seed nodes. Let h_u be the number of hops from the seed nodes to node u, and let $\mathcal{H}(i)$ be the set of nodes that are i hops away from the seed nodes; nodes in $\mathcal{H}(i)$ form layer i. $\mathcal{H}(0)$ is equivalent to the set of the seed nodes. We formally estimate the lower bounding relevance scores as follows:

Definition 2 (Lower bound). *The following equation defines the relevance estimation of node* u, $\underline{s}_u$*:*

$$\underline{s}_u = \begin{cases} cd_u & (u \in \mathcal{H}(0)) \\ (1-c) \sum_{v \in \mathcal{H}(h_u - 1)} A_{uv} \underline{s}_v & (u \notin \mathcal{H}(0)) \end{cases} \quad (7)$$

This equation implies that (1) if node u is a seed node, its estimation is obtained from the restart probability and its preference score, and (2) if node u is not a seed node, its estimation is computed from that of its upper layer nodes, where $\mathcal{H}(0)$ forms the top layer, with their transition probabilities. It requires $O(n+m)$ time to compute the estimations of all nodes. This is because the estimations are yielded from a single breadth-first search that is rooted on the seed nodes, and breadth-first search needs $O(n+m)$ time [9].

We introduce the following lemma to show the property of the above estimation approach:

Lemma 1 (Lower bound). *For any node in $\mathcal{H}(i)$, $\underline{s}_u \leq s_u$ holds in the graph.*

Proof We prove that Definition 2 gives lower bounding estimations by using mathematical induction [16].

Initial step: We show the statement holds for any node in $\mathcal{H}(0)$. As described in Section 2, a random walk specifies seed nodes when (1) it jumps to the seed nodes with probability c or (2) outgoing edge of a current node is incident to the seed nodes. In the case of (1), a random walk jumps to node u of the seed nodes with probability cd_u. Therefore, steady-state probability of node u cannot be lower than cd_u. That is, $\underline{s}_u \leq s_u$ holds for any node in $\mathcal{H}(0)$.

Inductive step: We assume that $\underline{s}_v \leq s_v$ holds for any node in $\mathcal{H}(i-1)$. We prove the statement holds for any node in $\mathcal{H}(i)$. For a node u in $\mathcal{H}(i)$, the following inequality holds from Equation (1) since $\mathcal{H}(i-1)$ is a subset of all node sets in the graph (i.e. $\mathcal{H}(i-1) \subset \mathcal{V}$) and $c, d_u \geq 0$:

$$s_u = (1-c)\sum_{v \in \mathcal{V}} A_{uv} s_v + c d_u \geq (1-c) \sum_{v \in \mathcal{H}(i-1)} A_{uv} \underline{s}_v = \underline{s}_u$$

This completes the inductive step. Therefore, Definition 2 gives lower bounding estimations. □

Therefore, for all nodes in the graph, we can effectively obtain lower bounding estimations by Definition 2.

3.3.2 Upper bounding estimation

In this section, we detail our upper bounding estimation approach which can prune unnecessary exact computations. The upper bounding estimation is based on the lower bounding estimation introduced in Section 3.3.1. To find the top-k nodes, nodes are explored one by one, and we compute the upper bounding estimation and exact relevance score for each node. Let u_i be the i-th explored node, the definition of the upper bounding estimation is as follows:

Definition 3 (Upper bound). *The relevance estimation of i-th explored node u_i, $\overline{s}_{u_i}$, is defined as follows:*

$$\overline{s}_{u_i} = \begin{cases} 1 - \sum_{j=2}^{n} \underline{s}_{u_j} & (i = 1) \\ \underline{s}_{u_i} - s_{u_{i-1}} + \overline{s}_{u_{i-1}} & (i \neq 1) \end{cases} \quad (8)$$

If $i = 1$, it needs $O(n)$ time to compute the estimation of a node from already computed lower bounding estimations. Otherwise, we can incrementally compute the estimation of a node at the cost of $O(1)$ time since $\underline{s}_{u_i}$, $s_{u_{i-1}}$, and $\overline{s}_{u_{i-1}}$ are already computed before computing $\overline{s}_{u_i}$.

In our approach, nodes are explored in descending order of their lower bounding estimations. There are two reasons for this approach. The first is that a node is expected to have high exact relevance score if its lower bounding relevance is high. Thus, we can efficiently find top-k nodes. The second reason is that, if an upper bounding estimation is lower than the K-th highest exact relevance of candidate nodes, our algorithm terminates the process based on the following property of the upper bounding estimation:

Algorithm 2 Top-k node identification

Input: **d**, query distribution; K, number of answer nodes
Output: $\mathcal{V}_a$, set of answer nodes
1: $\theta = 0$;
2: $\mathcal{V}_e = \emptyset$;
3: $\mathcal{V}_a = \emptyset$;
4: append K dummy nodes to $\mathcal{V}_a$;
5: compute the lower bounding estimations of all nodes;
6: **while** $\mathcal{V}_e \neq \mathcal{V}$ **do**
7: $u = \text{argmax}(\underline{s}_v | v \in \mathcal{V} \backslash \mathcal{V}_e)$;
8: compute the upper bounding estimation of node u;
9: **if** $\overline{s}_u < \theta$ **then**
10: **return** $\mathcal{V}_a$;
11: **else**
12: compute exact relevance of node u for distribution **d**;
13: **if** $s_u > \theta$ **then**
14: $v = \text{argmin}(s_w | w \in \mathcal{V}_a)$;
15: remove node v from $\mathcal{V}_a$;
16: append node u to $\mathcal{V}_a$;
17: $\theta = \min(s_w | w \in \mathcal{V}_a)$;
18: **end if**
19: **end if**
20: append node u to $\mathcal{V}_e$;
21: **end while**
22: **return** $\mathcal{V}_a$;

Lemma 2 (Upper bound). *$s_{u_i} \leq \overline{s}_{u_i} \leq \overline{s}_{u_{i-1}}$ holds for i-th explored node u_i if nodes are explored in descending order of their lower bounding estimations.*

Proof We first prove $s_u \leq \overline{s}_u$. From Equation (8) and Lemma 1, the following inequality holds since node relevance scores are steady-state probabilities:

$$\begin{aligned} \overline{s}_u &= \underline{s}_{u_i} - s_{u_{i-1}} + \overline{s}_{u_{i-1}} = \underline{s}_{u_i} - s_{u_{i-1}} + \underline{s}_{u_{i-1}} - s_{u_{i-2}} + \ldots + \overline{s}_{u_1} \\ &= \sum_{j=2}^{i} \underline{s}_{u_j} - \sum_{j=1}^{i-1} s_{u_j} + 1 - \sum_{j=2}^{n} \underline{s}_{u_j} = 1 - \sum_{j=1}^{i-1} s_{u_j} - \sum_{j=i+1}^{n} \underline{s}_{u_j} \\ &\geq 1 - \sum_{j=1}^{i-1} s_{u_j} - \sum_{j=i+1}^{n} s_{u_j} = 1 - \sum_{j \neq i}^{n} s_{u_j} = s_{u_i} \end{aligned}$$

We next prove that $\overline{s}_{u_i} \leq \overline{s}_{u_{i-1}}$ holds. From Equation (8), we have $\overline{s}_{u_{i-1}} - \overline{s}_{u_i} = s_{u_{i-1}} - \underline{s}_{u_i} \geq \underline{s}_{u_{i-1}} - \underline{s}_{u_i} \geq 0$. This completes the proof. □

Lemma 2 indicates that, if the upper bounding estimation of an explored node is lower than the K-th highest relevance of the candidate nodes, all other unexplored nodes have lower relevance scores than that of the K-th node. Therefore, we can safely terminate the algorithm.

3.3.3 Top-k node algorithm

We show the algorithm that finds the top-k nodes by the estimation approach. In Algorithm 2, θ, $\mathcal{V}_a$, and $\mathcal{V}_e$ indicate the K-th highest relevance among the candidate nodes, the set of candidate/answer nodes, and the set of explored nodes, respectively. The algorithm sets the candidate nodes by appending K dummy nodes whose relevance scores are all 0 (line 4), and computes the lower bounding estimations (line 5). It then finds the node with the highest lower bounding estimation from the unexplored node set, $\mathcal{V} \backslash \mathcal{V}_e$, and computes the upper bounding estimation of the node (lines 7-8). If the upper bounding estimation is lower than θ, the node and all other unexplored nodes cannot be the

Algorithm 3 Finding highly relevant nodes

Input: **d**, query distribution; ϵ, threshold
Output: $\mathcal{V}_a$, set of answer nodes
1: $\mathcal{V}_e = \emptyset$;
2: $\mathcal{V}_a = \emptyset$;
3: compute the lower bounding estimations of all nodes;
4: **while** $\mathcal{V}_e \neq \mathcal{V}$ **do**
5: $u = \text{argmax}(\underline{s}_v | v \in \mathcal{V} \backslash \mathcal{V}_e)$;
6: compute the upper bounding estimation of node u;
7: **if** $\overline{s}_u \leq \epsilon$ **then**
8: **return** $\mathcal{V}_a$;
9: **else if** $\underline{s}_u > \epsilon$ **then**
10: append node u to $\mathcal{V}_a$;
11: **else**
12: compute exact relevance of node u for distribution **d**;
13: **if** $s_u > \theta$ **then**
14: append node u to $\mathcal{V}_a$;
15: **end if**
16: **end if**
17: append node u to $\mathcal{V}_e$;
18: **end while**
19: **return** $\mathcal{V}_a$;

answer nodes (Lemma 2). Therefore, it terminates the process (lines 9-10). Otherwise, the node may be an answer node, so it computes the exact relevance of the node (line 12). If the computed relevance is higher than θ, it updates the candidate set, $\mathcal{V}_a$, and θ (lines 13-18). Finally, it returns the candidate set, $\mathcal{V}_a$, as the answer nodes (line 22).

We introduce the following theorem for Algorithm 2:

Theorem 4 (Exactness in top-k identification). *Algorithm 2 finds top-k nodes exactly.*

Proof If θ_K are the answer nodes with K-th highest relevance scores, the upper bounding estimations of the answer nodes cannot be lower than θ_K (Lemma 2). The algorithm prunes a node if the upper bounding estimation of the node is lower than θ. Since the K-the highest relevance score of the candidate nodes cannot be more than that of the answer nodes (i.e. $\theta \leq \theta_K$), the answer nodes cannot be pruned by the algorithm. And, if a node have a relevance score less than θ, the node is pruned with its exact/estimated relevance by the algorithm. □

We discuss here the time complexity of Algorithm 2.

Theorem 5 (Computational cost for top-k nodes). *Algorithm 2 requires $O(n \log n + m + |\mathbf{F}| + |\mathbf{Q}|)$ time to find top-k nodes.*

Proof The algorithm first computes lower bounding estimations for all nodes by single breadth-first search which takes $O(n + m)$ time. Next, it finds the node of the highest lower bounding estimation and computes the upper bounding estimation of the node. This requires $O(n \log n)$ and $O(n)$ time, respectively, if binomial heap [9] is used and none of the nodes are pruned. It takes $O(|\mathbf{F}| + |\mathbf{Q}|)$ time to compute exact relevance scores of all nodes since $\sum_{i=1}^{n} |\mathbf{f}_i| + |\mathbf{Q}| = |\mathbf{F}| + |\mathbf{Q}|$ (Theorem 2). Therefore, it needs $O(n \log n + m + |\mathbf{F}| + |\mathbf{Q}|)$ time to find the top-k nodes. □

3.4 Finding highly relevant nodes

In this section, we describe our approach to finding nodes whose relevance scores are higher than threshold ϵ. All proofs in this section are omitted due to the space limitations. To find highly relevant nodes, we utilize the lower bounding estimations to avoid unnecessary relevance computations; if the lower bounding estimation of a node is higher than ϵ, the node must be an answer node. Therefore, we prune the exact relevance computation of the node.

Nodes are explored in descending order of their lower bounding estimations. And the upper bounding estimations are utilized to skip unnecessary relevance computations as in top-k search. However, as shown in Definition 3, we cannot compute the upper bounding estimations of node u_i without the exact relevance of node u_{i-1}. In that case, we compute the upper bounding estimation of node u_i as follows:

Definition 4 (Upper bound w/o exact relevances). *The relevance estimation of node u_i, $\overline{s}_{u_i}$, is computed as follows without the exact relevance computation of node u_{i-1}:*

$$\overline{s}_{u_i} = \underline{s}_{u_i} - \underline{s}_{u_{i-1}} + \overline{s}_{u_{i-1}} \tag{9}$$

We introduce the following lemma to show the property of the above estimation approach:

Lemma 3 (Upper bound w/o exact relevances). *For i-th explored node u_i, $s_{u_i} \leq \overline{s}_{u_i} \leq \overline{s}_{u_{i-1}}$ holds, and it requires $O(1)$ time to compute the estimation if $i \neq 1$ in the estimation obtained by Definition 4.*

Algorithm 3 depicts our approach for finding highly relevant nodes. It first computes the lower bounding estimations for all nodes (line 3). It next finds the node with the highest lower bounding estimation and computes the upper bounding relevance of the node (lines 5-6). If the upper bounding estimation is not higher than ϵ, the node and remaining unexplored nodes cannot be answer nodes. Therefore, it terminates the process (lines 7-8). If the lower bounding estimation is higher than ϵ, its exact relevance must be higher than ϵ. So, we add the node to the answer node set (lines 9-10). Otherwise, the node can be answer node, so its exact relevance is computed (lines 11-16).

We have the following theorem for Algorithm 3:

Theorem 6 (Finding highly relevant nodes). *Algorithm 3 requires $O(n \log n + m + |\mathbf{F}| + |\mathbf{Q}|)$ time to exactly find highly relevant nodes.*

4. EXPERIMENTAL EVALUATION

We performed experiments to demonstrate the efficiency of the proposed approach against the iterative and approximate approaches. We used following three public datasets for the experiments:

- Social[3]: This is a graph taken from Slashdot.org[4]. The network contains interaction links between the users of Slashdot. There are 82,168 nodes and 948,464 edges.
- Routing[5]: We used routing information of the Internet. This graph was constructed from BGP tables posted by the University of Oregon Route Views Project[6]. This graph has 22,963 nodes and 48,436 edges.
- Web[7]: This dataset was taken from web pages in University of Notre Dame[8]. In this dataset, nodes represent pages from domain nd.edu and edges represent hyperlinks between them. There are 325,729 nodes and 1,497,135 edges.

[3] http://snap.stanford.edu/data/soc-Slashdot0902.html
[4] http://slashdot.org/
[5] http://www-personal.umich.edu/~mejn/netdata/as-22july06.zip
[6] http://routeviews.org/
[7] http://vlado.fmf.uni-lj.si/pub/networks/data/ND/NDnets.htm
[8] http://www.nd.edu/

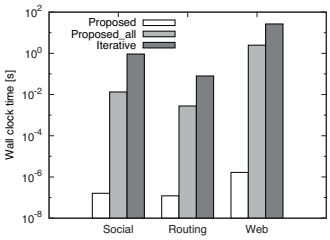

Figure 1: Relevance computation time.

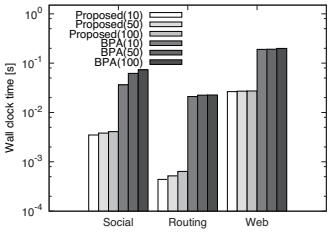

Figure 2: Efficiency of top-k nodes identification.

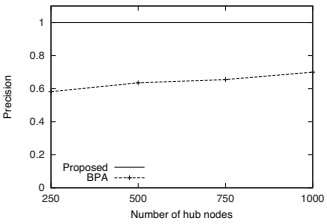

Figure 3: Accuracy versus number of hub nodes.

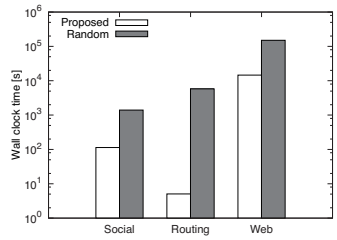

Figure 4: Precomputation time.

Table 2: Ratio of non-zero elements.

	Data set		
	Social	*Routing*	*Web*
$\lvert\mathbf{Q}\rvert/n^2$	3.40×10^{-6}	4.88×10^{-6}	2.76×10^{-4}
$\lvert\mathbf{R}^{-1}\rvert/n^2$	9.25×10^{-3}	2.40×10^{-3}	1.93×10^{-2}

We set the restart probability $c = 0.9$ as [13] did, and randomly selected ten nodes as seed nodes where the preference scores were equally set to same score. All the experiments were conducted on a Linux 2.26 GHz Intel Xeon server with 144GB of main memory. GCC 4.1.2 was used to implement all approaches.

4.1 Relevance computation time

We assessed relevance computation time for the proposed approach and the iterative approach since the none of previous approximate approaches can compute relevance exactly. To make the comparison fair, we also evaluated the relevance computation time of our approach for all nodes even though our approach can compute just single node relevance.

Figure 1 shows the computation time of the proposed approach for a single node relevance and all nodes relevances, referred to as "Proposed" and "Proposed_all", respectively. "Iterative" indicates the results of the iterative approach. Table 2 shows the ratio of non-zero elements in $\mathbf{Q}$ and $\mathbf{R}^{-1}$ to the squared number of nodes, n^2.

Figure 1 indicates that our approach is up to seven orders of magnitude faster than the iterative approach. The proposed approach is up to 70 times faster even if it computes the relevance scores of all nodes. As shown in Theorem 2, the proposed approach requires $O(|\mathbf{f}_x| + |\mathbf{Q}|)$ time to compute a single node relevance; the number of non-zero elements have an effect on the efficiency of relevance computations in our approach since $\mathbf{F} = c\mathbf{P}^T\mathbf{R}^{-1}$ and $\mathbf{g} = \mathbf{Q}^T\mathbf{P}\mathbf{d}$. Our permutation approach, as described in Section 3.2.2, is designed to obtain sparse matrices $\mathbf{Q}$ and $\mathbf{R}^{-1}$. As shown in Table 2, we can reduce the number of non-zero elements for $\mathbf{R}^{-1}$ and, especially for, $\mathbf{Q}$. That is, $|\mathbf{Q}| \ll |\mathbf{R}^{-1}| \ll n^2$. Therefore, we can compute relevance scores efficiently for the given distribution $\mathbf{d}$ since vector $\mathbf{g}$ is obtained from $\mathbf{d}$. However, as described in Section 2, the iterative approach recursively updates the relevance scores of all nodes until convergence. This leads to high computational cost. Therefore, we can compute relevance more efficiently than the iterative approach.

4.2 Computation time for top-k nodes

We performed experiments to demonstrate the effectiveness of our approach to find top-k nodes in a comparison to Basic Push Algorithm [17]. This algorithm approximately identifies the top-k nodes. It exploits precomputed relevance scores of all nodes from hub nodes to estimate the upper relevance bound. This implies it theoretically guarantees that the recall[9] of its answer results is always 1; all of the answer nodes output by the iterative approach are contained in those output by this algorithm. Other approximate approaches such as [3, 4] do not have this property. This also implies the number of answer nodes yielded by Basic Push Algorithm can be more than K. Hub nodes are selected according to degrees.

Figure 2 shows the efficiency of each approach where the number of hub nodes is set to $1,000$. In this figure, the results of the proposed approach and Basic Push Algorithm are referred to as "Proposed(K)" and "BPA(K)", respectively, where K is the number of answer nodes. Figure 3 shows the precision with the various numbers of hub nodes for Social dataset in finding the top ten nodes. Precision is the fraction of answer nodes among top-k results by each approach that match those of the iterative approach. We additionally confirmed similar results for the other datasets even though we omit the results due to space limitations.

As shown in Figure 2, our approach can find the top-k nodes up to 50 times faster than the Basic Push Algorithm. In the worst case, the proposed approach computes the relevance scores of all nodes if none of the nodes are pruned. However, the number of relevance computations is limited to small numbers due to the effectiveness of the estimation approach. It takes $O(n)$ times to compute the estimations for all nodes. Thus our approach can efficiently prune unnecessary relevance computations.

Figure 3 indicates that our approach outputs the same results as the iterative approach. Figure 3 also shows that Basic Push Algorithm can find top-k nodes more accurately as the number of the hub nodes increase since this algorithm computes the upper bounding estimations from the hub nodes[10]. However, this algorithm cannot find the top-k nodes exactly while our approach outputs only exact answers. Moreover, the precomputation time of the Basic Push Algorithm is proportional to the number of hub nodes since the relevance scores of all nodes from all hub nodes must be precomputed by the iterative approach until convergence. Figure 2 and 3 show that our approach is superior to this algorithm in both speed and accuracy.

4.3 Precomputation time

Our approach precomputes the matrices $\mathbf{Q}$ and $\mathbf{R}^{-1}$ to compute node relevance scores. Thus we evaluated the effectiveness of our permutation approach in terms of precom-

[9] Recall is the fraction of answer nodes extracted by the iterative approach that were successfully extracted by this algorithm.

[10] We confirmed that this algorithm more efficiently finds answer nodes as the number of the hub nodes increases for the same reason.

(1) Proposed (2) Monte Carlo(250) (3) Monte Carlo(1,000,000)

Figure 5: Highly relevant cities for San Diego, Denver, Orlando, and New York as detected by each approach.

putation time. In Figure 4, "Random" represents the results achieved when nodes are premutated in random order.

Figure 4 indicates that our permutation approach enhances the precomputation time as well as reducing the number of non-zero elements. Our approach is up to $1,200$ times faster than the random permutation approach. Since the matrices $\mathbf{Q}$ and $\mathbf{R}^{-1}$ have sparse data structures in our approach, we can effectively reduce computations of non-zero matrices elements in Equation (4), (5), and (6). Therefore, we can efficiently obtain the matrices $\mathbf{Q}$ and $\mathbf{R}^{-1}$.

5. CASE-STUDIES

Our approach can exactly find nodes whose relevances are higher than ϵ. To show the applicability of our approach, we demonstrate the results of case-studies on a real dataset. Because none of the previous approaches, including Basic Push Algorithm, are designed to find highly relevant nodes, we compared our approach to the state-of-the-art approximate method by Avrachenkov et al. [3]. This approach uses Monte Carlo method to compute approximate relevance scores by performing multiple random walks. The results of case-studies revealed that our approach can obtain reasonable results more efficiently than the Monte Carlo method.

For this evaluation, we used the city dataset[11] consisting of $71,959$ direct flights between 456 cities in North America. Flights and cities correspond to edges and nodes. Edge weights in the adjacency matrix are proportional to city proximities. We identified the nodes whose relevance scores exceeded 5×10^{-4}. We set San Diego, Denver, Orlando, and New York as the seed nodes. The preference scores of the seed nodes were equally set to the same score where the restart probability, c, was set to 0.9, the same as in the previous section. Figure 5 shows the results. The number of random walks in the Monte Carlo approach was set to 250 and $1,000,000$ in Figure 5-(2) and 5-(3), respectively. In Figure 5, circles and triangles represent seed nodes and highly relevant nodes, respectively.

Figure 5-(1) shows highly relevant cities for San Diego, Denver, Orlando, and New York as determined by the proposed approach. These cities are all large ones in the United States. These cities have big airline hubs from which almost all cities can be directly accessed. Since PPR computes random walks from seed nodes, highly connected nodes from other nodes are expected to be highly relevant nodes if large cities are selected as seed nodes. As expected, relevant cities were Los Angeles, Phoenix, Colorado Springs, Daytona Beach, Philadelphia, and Washington, D.C. which also have big airline hubs. That is, we can detect cities with airline hubs from among large cities. Note that the results of our approach equal those of the iterative approach.

[11] http://www.psi.toronto.edu/index.php?q=affinity propagation

As shown in Figure 5-(2), if the number of random walks is 250, the Monte Carlo approach outputs different results from the proposed approach except for Washington, D.C. and Colorado Springs. The process time of the Monte Carlo approach was 21 microseconds while that of our approach was 20.5 microseconds; they are almost the same. As shown in Figure 5-(3), if the number of random walks is increased, the Monte Carlo approach has the same outputs as the proposed approach except for Baltimore. However, this leads to high computational time since the computational cost of the Monte Carlo approach is proportional to the number of random walks [3]. The Monte Carlo approach took 80.7 milliseconds with this setting; the proposed approach is more than $3,900$ times faster.

The result of our approach is reasonable, and is consistent with our intuition. While our approach achieves high efficiency, it is guaranteed to output the same results as the iterative approach. This indicates that our approach can be another option for the research community in utilizing PPR.

6. RELATED WORK

Several approximate approaches have been proposed focusing on efficient computation for PPR. However, none of them guarantees the exact answer result; their outputs can differ from that of the iterative approach. Therefore, it is difficult for these approximate approaches to enhance the quality of real applications. None of them can be taken as a unified scheme that comprehensively solves the three problems described in Section 1.

Jeh et al. suggested a framework that, for nodes that belong to a highly linked subset of hub nodes, provides a scalable solution for PPR [23]. The approach is based on the observation that relevance scores for a given distribution can be approximated as a linear combination of the relevance scores from a single node in hub nodes. They compute approximate relevance scores for a given distribution using the relevance scores precomputed from hub nodes. Fogaras et al. proposed a Monte Carlo algorithm for PPR [10]. To compute approximate relevance scores, they use fingerprints of several nodes; a fingerprint of a node is a random walk starting from the node. They precompute fingerprints by simulating random walks and approximate relevance distributions from the ending nodes of these random walks. They approximate relevance scores by exploiting the linearity property of PPR. Avrachenkov et al. and Bahmani et al. independently improved the realization of the Monte Carlo method [3, 4]. Basic Push Algorithm is an approach that finds top-k nodes approximately but efficiently for PPR [17]. The approach uses precomputed relevance scores of hub nodes to estimate the upper relevance bound. However, their approach outputs ineligible nodes for top-k search, i.e. those that have

low relevance scores as shown in Section 4. Basic Push Algorithm, proposed Chakrabarti et al., efficient approximate approach for ER graphs [7].

Random walk with restart (RWR) [13] is a special case of PPR in that it has only a single seed node; many papers have targeted the enhancement of RWR efficiency. Sun et al. proposed an approximate approach for RWR [33]. They used METIS [24] to partition a graph into several disjoint subgraphs, and they performed RWR only on the subgraph that contains the seed node. All nodes outside the subgraph are simply assigned RWR relevance scores of 0. In other words, their approach outputs a local estimation of RWR. Their approach is based on the observation that some of the nodes that are just a few hops away from the seed node have high RWR relevance scores. Based on the same observation, K-dash finds top-k nodes efficiently for RWR [12]. By using upper bounding estimations, it prunes unnecessary relevance computations of nodes that are far from the seed nodes. However, these two approaches do not support the case that there are multiple seed nodes as is true in PPR. Tong et al. showed that eigen-value decomposition [19] can be utilized to approximate RWR relevance scores [34]. They also showed the proof of an error bound of the approach. However, their proof is based on the assumption that the normalized graph Laplacian [39] can be exploited to represent a graph. Furthermore, it is impractical to compute eigen-value decomposition of a large graph since it is extremely time-consuming as they themselves pointed out.

7. CONCLUSIONS

We addressed the problems of computing single node relevance and finding the top-k/highly relevant nodes efficiently and exactly for PPR. Our approach is to compute single node relevance from sparse matrices in a non-iterative style and to prune unnecessary relevance computations by upper/lower relevance estimations. Experiments show that the proposed approach can achieve high efficiency without sacrificing the accuracy of answer results. PPR is fundamental in many applications. The proposed approach allows many applications to be implemented more efficiently, and helps to improve the effectiveness of future applications.

8. REFERENCES

[1] R. Andersen, F. R. K. Chung, and K. J. Lang. Local Graph Partitioning Using PageRank Vectors. In *FOCS*, pages 475–486, 2006.

[2] K. Avrachenkov, V. Dobrynin, D. Nemirovsky, S. K. Pham, and E. Smirnova. PageRank Based Clustering of Hypertext Document Collections. In *SIGIR*, pages 873–874, 2008.

[3] K. Avrachenkov, N. Litvak, D. Nemirovsky, E. Smirnova, and M. Sokol. Quick Detection of Top-k Personalized PageRank Lists. In *WAW*, pages 50–61, 2011.

[4] B. Bahmani, A. Chowdhury, and A. Goel. Fast Incremental and Personalized PageRank. *PVLDB*, 4(3):173–184, 2010.

[5] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based Keyword Search in Databases. In *VLDB*, pages 564–575, 2004.

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[7] S. Chakrabarti, A. Pathak, and M. Gupta. Index Design and Query Processing for Graph Conductance Search. *VLDB J.*, 20(3):445–470, 2011.

[8] Y.-Y. Chen, Q. Gan, and T. Suel. Local Methods for Estimating PageRank Values. In *CIKM*, pages 381–389, 2004.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2009.

[10] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós. Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments. *Internet Mathematics*, 2(3), 2005.

[11] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. Random-walk Computation of Similarities Between Nodes of a Graph with Application to Collaborative Recommendation. *IEEE Trans. Knowl. Data Eng.*, 19(3):355–369, 2007.

[12] 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.

[13] B. Gallagher, H. Tong, T. Eliassi-Rad, and C. Faloutsos. Using Ghost Edges for Classification in Sparsely Labeled Networks. In *KDD*, pages 256–264, 2008.

[14] B. Gao, T.-Y. Liu, W. Wei, T. Wang, and H. Li. Semi-supervised Ranking on Very Large Graphs with Rich Metadata. In *KDD*, pages 96–104, 2011.

[15] M. Gori and A. Pucci. Itemrank: A Random-walk Based Scoring Algorithm for Recommender Engines. In *IJCAI*, pages 2766–2771, 2007.

[16] D. S. Gunderson. *Handbook of Mathematical Induction: Theory and Applications*. Chapman and Hall/CRC, 2010.

[17] M. S. Gupta, A. Pathak, and S. Chakrabarti. Fast Algorithms for Top-k Personalized PageRank Queries. In *WWW*, pages 1225–1226, 2008.

[18] A. Harpale, Y. Yang, S. Gopal, D. He, and Z. Yue. Citedata: a New Multi-faceted Dataset for Evaluating Personalized Search Performance. In *CIKM*, pages 549–558, 2010.

[19] D. A. Harville. *Matrix Algebra From a Statistician's Perspective*. Springer, 2008.

[20] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An Algorithmic Framework for Performing Collaborative Filtering. In *SIGIR*, pages 230–237, 1999.

[21] V. Hristidis, L. Raschid, and Y. Wu. Scalable Link-based Personalization for Ranking in Entity-relationship Graphs. In *WebDB*, 2011.

[22] G. Jeh and J. Widom. SimRank: a Measure of Structural-context Similarity. In *KDD*, pages 538–543, 2002.

[23] G. Jeh and J. Widom. Scaling Personalized Web Search. In *WWW*, pages 271–279, 2003.

[24] G. Karypis, V. Kumar, and V. Kumar. Multilevel K-way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed Computing*, 48:96–129, 1998.

[25] D. Liben-Nowell and J. M. Kleinberg. The Link Prediction Problem for Social Networks. In *CIKM*, pages 556–559, 2003.

[26] Q. Liu, E. Chen, H. Xiong, and C. H. Q. Ding. Exploiting User Interests for Collaborative Filtering: Interests Expansion via Personalized Ranking. In *CIKM*, pages 1697–1700, 2010.

[27] F. McSherry. A Uniform Approach to Accelerated PageRank Computation. In *WWW*, pages 575–582, 2005.

[28] G. Namata, S. Kok, and L. Getoor. Collective Graph Identification. In *KDD*, pages 87–95, 2011.

[29] M. Newman, A.-L. Barabasi, and D. J. Watts. *The Structure And Dynamics of Networks*. Princeton University Press, 2006.

[30] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[31] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition*. Cambridge University Press, 2007.

[32] E. Smirnova, K. Avrachenkov, and B. Trousse. Using Web Graph Structure for Person Name Disambiguation. In *CLEF*, 2010.

[33] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood Formation and Anomaly Detection in Bipartite Graphs. In *ICDM*, pages 418–425, 2005.

[34] H. Tong, C. Faloutsos, and J.-Y. Pan. Fast Random Walk with Restart and Its Applications. In *ICDM*, pages 613–622, 2006.

[35] K. Voevodski, S.-H. Teng, and Y. Xia. Spectral Affinity in Protein Networks. *BMC Systems Biology*, 3(1):112, 2009.

[36] S. White and P. Smyth. Algorithms for Estimating Relative Importance in Networks. In *KDD*, pages 266–275, 2003.

[37] J. Yang and J. Leskovec. Patterns of Temporal Variation in Online Media. In *WSDM*, pages 177–186, 2011.

[38] M. Yannakakis. Computing the Minimum Fill-in is NP-complete. *SIAM. J. on Algebraic and Discrete Methods*, 2(1):77–79, 1981.

[39] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with Local and Global Consistency. In *NIPS*, 2003.