Personalized PageRank to a Target Node, Revisited

Hanzhi Wang
hanzhi_wang@ruc.edu.cn
School of Information
Renmin University of China

Zhewei Wei∗
zhewei@ruc.edu.cn
GaoingSchool of Artificial Intelligence
Renmin University of China

Junhao Gan
junhao.gan@unimelb.edu.au
School of Computing and Information Systems
University of Melbourne

Sibo Wang
swang@se.cuhk.edu.hk
Department of Systems Engineering and Engineering Management
The Chinese University of Hong Kong

Zengfeng Huang
huangzf@fudan.edu.cn
School of Data Science
Fudan University

ABSTRACT

Personalized PageRank (PPR) is a widely used node proximity measure in graph mining and network analysis. Given a source node \( s \) and a target node \( t \), the PPR value \( \pi(s, t) \) represents the probability that a random walk from \( s \) terminates at \( t \), and thus indicates the bidirectional importance between \( s \) and \( t \). The majority of the existing work focuses on the single-source queries, which asks for the PPR value of a given source node \( s \) and every node \( t \in V \). However, the single-source query only reflects the importance of each node \( t \) with respect to \( s \). In this paper, we consider the single-target PPR query, which measures the opposite direction of importance for PPR. Given a target node \( t \), the single-target PPR query asks for the PPR value of every node \( s \in V \) to a given target node \( t \). We propose RBS, a novel algorithm that answers approximate single-target queries with optimal computational complexity. We show that RBS improves three concrete applications: heavy hitters PPR query, single-source SimRank computation, and scalable graph neural networks. We conduct experiments to demonstrate that RBS outperforms the state-of-the-art algorithms in terms of both efficiency and precision on real-world benchmark datasets.

CCS CONCEPTS

• Mathematics of computing → Graph algorithms; • Information systems → Data mining.

KEYWORDS

Personalized PageRank, single-target query, graph mining

∗Zhewei Wei is the corresponding author. Work partially done at Beijing Key Laboratory of Big Data Management and Analysis Methods, and at Key Laboratory of Data Engineering and Knowledge Engineering, MOE, Renmin University of China.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

KDD ’20, August 23–27, 2020, Virtual Event, USA
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7998-4/20/08...
https://doi.org/10.1145/3394486.3403108

1 INTRODUCTION

Personalized PageRank (PPR), as a variant of PageRank [46], focuses on the relative significance of a target node with respect to a source node in a graph. Given a directed graph \( G = (V, E) \) with \( n \) nodes and \( m \) edges, the PPR value \( \pi(s, t) \) of a target node \( t \) with respect to a source node \( s \) is defined as the probability that an \( \alpha \)-discounted random walk from \( s \) terminates at \( t \). Here an \( \alpha \)-discounted random walk represents a random traversal that, at each step, either terminates at the current node with probability \( \alpha \), or moves to a random out-neighbor with probability \( 1 - \alpha \). For a given source node \( s \), the PPR value of each node \( t \) is \( \sum_{s \in V} \pi(s, t) = 1 \), and thus \( \pi(s, t) \) reflects the significance of node \( t \) with respect to the source node \( s \). On the other hand, PPR to a target node can be related to PageRank: the summation of PPR from each node \( s \in V \) to a given target node \( t \) is \( \sum_{s \in V} \pi(s, t) \). Large \( \pi(s, t) \) also shows the great contribution \( s \) made for \( t \)’s PageRank, the overall importance of \( t \). Therefore, \( \pi(s, t) \) indicates bidirectional importance between \( s \) and \( t \).

PPR has widespread applications in the area of data mining, including web search [23], spam detection [4], social networks [21], graph neural networks [29, 63], and graph representation learning [45, 54, 64], and thus has drawn increasing attention during the past years. Studies on PPR computations can be broadly divided into four categories: 1) single-pair query, which asks for the PPR value of a given source node \( s \) and a given target node \( t \); 2) single-source query, which asks for the PPR value of a given source node \( s \) to every node \( t \in V \) as the target; 3) single-target query, which asks for the PPR value of every node \( s \in V \) to a given target node \( t \); 4) all-pairs query, which asks for the PPR value of each pair of nodes. While single-pair and single-source queries have been extensively studied [37, 39, 59, 61], single-target PPR query is less understood due to its hardness. In this paper, we study the problem of efficiently computing the single-target PPR query with error guarantee. We demonstrate that this problem is a primitive of both practical and theoretical interest.
Table 1: Complexity of single-source and single-target PPR queries.

|                  | Single-Source PPR | Single-Target PPR |
|------------------|-------------------|-------------------|
|                  | O(\frac{1}{\alpha}) | O(\frac{1}{\alpha}) |
| Relative error   | \tilde{O} \left( \frac{1}{\alpha} \right) | \tilde{O} \left( \frac{1}{\alpha} \right) |
| Additive error   | \tilde{O} \left( \frac{1}{\alpha^2} \right) | \tilde{O} \left( \frac{1}{\alpha^2} \right) |

1.1 Motivations and Concrete Applications

We first give some concrete applications of the single-target PPR query. We will elaborate on how to use our single-target PPR algorithm to improve the complexity for these applications in Section 5.

Approximate heavy hitters in PPR. The heavy hitters PPR problem [58] asks for all nodes \( s \in V \) such that \( \pi(s, t) > \phi \cdot \pi(t) \) with a given node \( t \) and a parameter \( \phi \). As opposite to the single-source PPR query, which asks for the important nodes for a given source node \( s \), heavy hitters PPR query asks for the nodes \( s \in V \) for which \( t \) is important. The motivation of the heavy hitters PPR query is to consider the opposite direction of importance as a promising approach to enhance the effectiveness of recommendation. Intuitively, the single-target query is a generalization of heavy hitters PPR query.

Approximate single-source SimRank. SimRank is a widely used node similarity measure proposed by Jeh and Widom[22]. Compared with PPR, SimRank is symmetric and thus is of independent interest in various graph mining tasks [35, 32, 34, 41, 52]. A large number of works [14, 24, 30, 31, 33, 35, 43, 50, 53, 66] focus on the single-source SimRank query, which asks for the SimRank similarity between a given node \( u \) and every other node \( v \in V \). Following [50], we can formulate SimRank in the framework of \( \alpha \)-discounted random walks. In particular, if we revert the direction of every edge in the graph, the SimRank similarity \( s(u, v) \) of node\( u \) and \( v \) equals to the probability that two \( \alpha \)-discounted random walks from \( u \) and \( v \) visit at the same node \( w \) with the same steps. As a result, it is shown in [60] that the bottleneck of the computational complexity of single-source SimRank depends on how fast we can compute the single-target PPR value for each node \( v \) and the target node \( w \in V \). Hence, by improving the complexity of single-target PPR query, we can also improve the performance of the state-of-the-art single-source SimRank algorithms.

Approximate PPR matrix and graph neural networks. In recent years, graph neural networks have drawn increasing attention due to their applications in various machine learning tasks. Graph neural networks focus on learning a low-dimensional latent representation for each node in the graph from the structural information and the node features. Many graph neural network algorithms are closely related to the approximate PPR matrix problem, which computes the approximate PPR value for every pair of nodes \( s, t \in V \). For example, a few unsupervised graph embedding methods, such as HOPE [45] and STRAP [64], suggest that directly computing and decomposing the approximate PPR matrix into low-dimensional vectors achieves satisfying performance in various downstream tasks. On the other hand, several recent works on semi-supervised graph neural networks, such as APPNP [28], PPRGo [63], and GDC [29], propose to use the (approximate) PPR matrix to smooth the node feature matrix. It is shown [29] that the approximate PPR matrix outperforms spectral methods, such as GCN [27] and GAT [55], in various applications.

The computation bottleneck for these graph learning algorithms is the computation of the approximate PPR matrix, as the power method takes at least \( O(n^2) \) time and space and is not scalable on large graphs. On the other hand, there are two alternative approaches to compute the approximate PPR matrix: issue a single-source query to every source node \( s \in V \) to compute \( \pi(s, t), t \in V \), or issue a single-target query to every target node \( t \in V \) to compute \( \pi(s, t), s \in V \). As we shall see in Section 5, the later approach is superior as it can provide the absolute error guarantee. Therefore, by proposing a faster single-target PPR algorithm, we also improve the computation time of the approximate PPR matrix. In particular, we show that our new single-target PPR algorithm computes the approximate PPR matrix in time sub-linear to the number of edges in the graphs, which significantly improves the scalability of various graph neural networks.

Theoretical motivations. Unlike the single-source PPR query, the complexity of the single-target PPR query remains an open problem. In particular, given a source node \( s \), it is known that a simple Monte-Carlo algorithm can approximately find all nodes \( t \in V \) such that \( \pi(s, t) > \delta \) with constant probability in \( \tilde{O}(1/\delta) \) time (see Section 2 for a detailed discussion), where \( \tilde{O} \) denotes the Big-Oh notation ignoring the log factors. Note that there are at most \( O(1/\delta) \) nodes \( t \) with \( \pi(s, t) > \delta \), which implies that there is a lower bound of \( O(1/\delta) \) and thus the simple Monte Carlo algorithm is optimal. On the other hand, given a random target node \( t \), the state-of-the-art single-target PPR algorithm finds all nodes \( s \in V \) with \( \pi(s, t) > \delta \) in \( \tilde{O}(d/\delta) \) time, where \( d \) is the average degree of the graph. Thus, there is an \( \tilde{O}(d) \) gap between the upper bound and lower bound for the single-target PPR problem. For dense graphs such as complete graphs, the \( \tilde{O}(d) \) gap is significant. Therefore, an interesting open problem is: is it possible to achieve the same optimal complexity as the single-source PPR query for the single-target PPR query?

1.2 Problem definition and Contributions

Problem definition. In this paper, we consider the problem of efficiently computing approximate single-target PPR queries. Following [10], the approximation quality is determined by relative or additive error. More specifically, we define approximate single-target PPR with additive error as follows.

Definition 1.1 (Approximate Single-Target PPR with additive error). Given a directed graph \( G = (V, E) \), a target node \( t \), an additive error bound \( \epsilon \), an approximate single-target PPR query with additive error returns an estimated PPR value \( \hat{\pi}(s, t) \) for each \( s \in V \), such
that
\[ |\hat{\pi}(s, t) - \pi(s, t)| \leq \varepsilon \] (1)
holds with a constant probability.

For single-target PPR query with relative error, we follow the definition of [10].

**Definition 1.2 (Approximate Single-Target PPR with relative error).** Given a directed graph \( G = (V, E) \), a target node \( t \), and a threshold \( \delta \), an approximate single-target PPR with relative error returns an estimated PPR value \( \hat{\pi}(s, t) \) for each \( s \in V \), such that for any \( \pi(s, t) > \delta \),
\[ |\hat{\pi}(s, t) - \pi(s, t)| \leq \frac{1}{10} \cdot \pi(s, t) \] (2)
holds with a constant probability.

Note that to simplify the presentation, we assume that the relative error parameter and success probability are constants following [10]. We can boost the success probability to arbitrarily close to \( 1 \) with the Median-of-Mean trick [12], which only adds a log factor to the running time. For these two types of error, we propose Randomized Backward Search (RBS), a unified algorithm that achieves optimal complexity for the single-target PPR query. We summarize the properties of the RBS algorithm as follows.

- Given a target node \( t \), RBS answers a single-target PPR query with constant relative error for all \( \pi(s, t) \geq \delta \) with constant probability using \( \tilde{O} \left( \frac{n \pi(t)}{\delta} \right) \) time. This result suggests that RBS achieves optimal time complexity for the single-target PPR query with relative error, as there may be \( O \left( \frac{n \pi(t)}{\delta} \right) \) nodes with \( \pi(s, t) \geq \delta \).
- Given a random target node \( t \), RBS answers a single-target PPR query with an additive error \( \varepsilon \) with constant probability using \( \tilde{O} \left( \frac{n \pi(t)}{\delta^2} \right) \) time. This query time complexity improves previous bound for single-target PPR query with additive error by a factor of \( \sqrt{\delta} \). Table 1 presents a detailed comparison between RBS and the state-of-the-art single-target PPR algorithm.

We demonstrate that the RBS algorithm improves the complexity of single-source SimRank computation, heavy hitters PPR query, and PPR-related graph neural networks in Section 5. We also conduct an empirical study to evaluate the performance of RBS. The experimental results show that RBS outperforms the state-of-the-art single-target PPR algorithm on real-world datasets.

## 2 PRELIMINARY

### 2.1 Existing Methods

**Power Method** is an iterative method for computing single-source and single-target PPR queries [46]. Recall that, at each step, an \( \alpha \)-discounted random walk terminates at the current node with \( \alpha \) probability or moves to a random out-neighbor with \( 1 - \alpha \) probability. This process can be expressed as the iteration formula with single-source PPR vector.
\[
\pi_s = (1 - \alpha) \pi_s \cdot P + \alpha \cdot e_t,
\] (3)
where \( \pi_s \) denotes the PPR vector with respect to a given source node \( s \), \( e_t \) denotes the one-hot vector with \( e_t(s) = 1 \), and \( P \) denotes the transition matrix where

| Table 2: Table of notations. |
|-----------------------------|
| Notation | Description |
| \( n, m \) | the numbers of nodes and edges in \( G \) |
| \( N_{in}(u), N_{out}(u) \) | the in/out neighbor set of node \( u \) |
| \( d_{in}(u), d_{out}(u) \) | the in/out degree of node \( u \) |
| \( \pi(s, t), \hat{\pi}(s, t) \) | the true and estimated PPR values of node \( s \) to \( t \) |
| \( \pi_{e}(s, t), \hat{\pi}_{e}(s, t) \) | the true and estimated \( \ell \)-hop PPR values of node \( s \) to \( t \) |
| \( \alpha \) | the teleport probability that a random walk terminates at each step |
| \( \varepsilon \) | the additive error parameter |
| \( \delta \) | the relative error threshold |
| \( d \) | the average degree, \( d = \frac{m}{n} \) |
| \( O \) | the Big-Oh notation ignoring the log factors |

\[
P(i, j) = \begin{cases} 
\frac{1}{d_{out}(v_i)}, & \text{if } v_j \in N_{out}(v_i), \\
0, & \text{otherwise.}
\end{cases}
\] (4)

Reversing this process, we can also compute single-target PPR values with the given target node \( t \). The iteration formula should be adjusted correspondingly:
\[
\pi_t = (1 - \alpha) \pi_t \cdot P^T + \alpha \cdot e_t.
\] (5)

Power Method can be used to compute the ground truths for the single-source and single-target query. After \( \ell = \log_{1-\alpha}(\varepsilon) \) iterations, the absolute error can be bounded by \( (1 - \alpha)^\ell \varepsilon \). Since each iteration takes \( O(m) \) time, it follows that the Power Method computes the approximate single-target PPR query with additive error in \( O \left( m \cdot \log \frac{1}{\varepsilon} \right) \) time. Note that the dependence on the error parameter \( \varepsilon \) is logarithmic, which implies that the Power Method can answer single-target PPR queries with high precision. However, the query time also linearly depends on the number of edges, which limits its scalability on large graphs.

**Backward Search** [38] is a local search method that efficiently computes the single-target PPR query on large graphs. Algorithm 1 illustrates the pseudo-code of Backward Search. We use residual \( r^h(u, t) \) to denote the probability mass to be distributed at node \( s \), and reserve \( \pi^h(s, t) \) denotes the probability mass that will stay at \( s \) permanently. For initialization, Backward Search sets \( r^h(u, t) = \pi^h(u, t) = 0 \) for all \( u \in V \), except for the residue \( r^h(t, t) = 1 \). In each push operation, it picks the node \( u \) with the largest residual \( r^h(u, t) \), and transfer a fraction of \( \alpha \) to \( \pi^h(u, t) \), the reserve of \( v \). Then the algorithm transfers the other \( (1 - \alpha) \) fraction to the in-neighbors of \( v \). For each in-neighbor \( u \) of \( v \), the residual \( r^h(u, t), u \in N_{in}(v) \) is incremented by \( \frac{(1 - \alpha)r^h(u, t)}{d_{out}(u)} \). After all in-neighbors are processed, the algorithm sets the residue of \( v \) to be 0. The process ends when the maximum residue descends below the error parameter \( \varepsilon \). Finally, Backward Search uses the reserve \( \pi^h(s, t) \) as the estimator for \( \pi(s, t), s \in V \). Backward Search utilizes the following property of the single-target PPR vector.

**Proposition 2.1.** Denote \( I(u = v) \) as the indicator variable such that \( I(u = v) = 1 \) if \( u = v \). For all \( s, t \in V \), \( \pi(s, t) \) satisfies that
\[
\pi(s, t) = \sum_{u \in N_{in}(t)} \frac{1 - \alpha}{d_{out}(u)} \cdot \pi(s, u) + \alpha \cdot I(s = t).
\] (6)
Utilizing this property, it is shown in [38] that the residues and reserves of Backward Search satisfies the following invariant:

\[ \pi(s, t) = \pi^b(s, t) + \sum_{u \in V} r^b(u, t) \cdot \pi(s, u). \]  

(7)

Note that when the Backward Search algorithm terminates, all residues \( r^b(u, t) \leq \epsilon. \) It follows that \( \pi^b(s, t) \leq \pi(s, t) \leq \pi^b(s, t) + \epsilon \sum_{u \in V} \pi(s, u) = \pi^b(s, t) + \epsilon, \) where \( \sum_{u \in V} \pi(s, u) = 1. \) Therefore, Backward Search ensures an additive error of \( \epsilon. \) It is shown in [38] that the running time of Backward Search is bounded by \( O\left(\sum_{u \in V} \frac{d_{\text{out}}(u) \cdot \pi(u, t)}{\epsilon}\right). \) We claim that the running time of Backward Search can also be bounded by \( O\left(\sum_{u \in V} \frac{d_{\text{out}}(u) \cdot \pi(u, t)}{\epsilon}\right). \) Due to space limitations we omit this proof, and refer the reader to the full version of the paper [1]. If the target node \( t \) is randomly selected, the complexity becomes \( O\left(\frac{d}{2}\right), \) where \( d \) is the average degree of the graph. For relative error, we can set \( \delta = \Theta(\epsilon) \) and obtain a worst-case complexity of \( O\left(\sum_{u \in V} \frac{d_{\text{out}}(u) \cdot \pi(u, t)}{\epsilon}\right) \) and an average complexity of \( O\left(\frac{d}{2}\right), \) respectively.

**Single-source algorithms.** The Monte-Carlo algorithm [14] computes the approximate single-source PPR query by sampling abundant random walks from source node \( s \) and using the proportion of the random walks that terminate at \( t \) as the estimator of \( \pi(s, t). \) According to Chernoff bound, the number of random walks required for an additive error \( \epsilon \) is \( O\left(\frac{d}{\epsilon}\right), \) while the number of random walks required to ensure constant relative error for all PPR larger than \( \delta \) is \( O\left(\frac{d}{\delta}\right). \) This simple method is optimal for single-source PPR queries with relative error, as there are at most \( O\left(\frac{d}{\epsilon}\right) \) nodes \( t \) with PPR \( \pi(s, t) \geq \delta. \) However, the Monte-Carlo algorithm does not work for single-target queries, as there lacks of a mechanism for sampling source nodes from a given target node. Moreover, it remains an open problem whether it is possible to achieve the same optimal \( O\left(\frac{d}{\epsilon}\right) \) complexity for the single-target query.

**Forward Search** [6] is the analog of Backward Search, but for single-source PPR queries. Similar to Backward Search, Forward Search uses residue \( r^f(s, u) \) to denote the probability mass to be distributed at node \( u, \) and \( r^f(s, u) \) to denote the probability mass that will stay at node \( u \) permanently. For initialization, Forward Search sets \( r^f(s, u) = \pi^f(s, u) = 0 \) for all \( u \in V, \) except for the residue \( r^f(s, s) = 1. \) In each push operation, it picks the node \( u \) with the largest residue/degree ratio \( \frac{r^f(s, u)}{d_{\text{out}}(u)}, \) and transfer a fraction of \( \alpha \) to \( \pi^f(s, u), \) the reserve of \( u. \) Then the algorithm transfers the other \( (1-\alpha) \) fraction to the out-neighbors of \( u. \) For each out-neighbor \( v \) of \( u, \) the residue \( r^f(s, v) \) is incremented by \( \frac{(1-\alpha) r^f(s, u)}{d_{\text{out}}(u)}. \) After all in-neighbors are processed, the algorithm sets the residue of \( u \) to 0. The process ends when the maximum residue/degree ratio descends below the error parameter \( \epsilon. \) Finally, Forward Search uses the reserve \( \pi^f(s, u) \) as the estimator for \( \pi(s, u), \) \( u \in V. \)

As shown in [6], Forward Search runs in \( O(1/\epsilon) \) time. However, the major problem with Forward Search is that it can only ensure an additive error of \( \epsilon d_{\text{out}}(u) \) for each PPR value \( \pi(s, u) \) on undirected graphs. Compared to the \( \epsilon \) error bound by Backward Search, this weak error guarantee makes Forward Search unfavorable when we need to compute the approximate PPR matrix in various graph neural network applications. We also note that there are a few works [57, 59, 61] that combines Forward Search, Backward Search and Monte-Carlo to answer single-single PPR queries. However, these methods are not applicable to the single-target PPR queries.

### 2.2 Other related work

PPR has been extensively studied for the past decades [5–9, 11, 13–20, 23, 26, 36, 37, 40, 42, 44, 48, 49, 51, 59, 62, 63, 65, 67–69]. Existing work has studied other variants of PPR queries. Research work on exact single-source queries [11, 26, 42, 46, 51, 69] aims at improving the efficiency and scalability of the Power Method. Research work on single-source top-\( k \) queries [36, 37, 40, 42, 56, 59, 69] focuses on (approximately) returning \( k \) nodes with the highest PPR values to a given source node. Single-pair PPR queries are studied by [14, 36, 37, 40, 56], which estimates the PPR value of a given pair of nodes. PPR computation has also been studied on dynamic graphs [9, 11, 44, 47, 48, 67, 68] and in the distributed environment [8, 19]. These studies, however, are orthogonal to our work. Table 2 summaries the notations used in this paper.

### 3 The RBS Algorithm

**High-level ideas.** In this section, we present Randomized Backward Search (RBS), an algorithm that achieves optimal query cost for the single-target query. Compared to the vanilla Backward Search (Algorithm 1), we employ two novel techniques. First of all, we decompose the PPR value \( \pi(s, t) \) into the \( \ell \)-hop Personalized PageRank \( \pi_{\ell}(s, t), \) which is defined as the probability that an \( \alpha \)-discounted random walk from \( s \) terminates at \( t \) exactly \( \ell \) steps. For different \( \ell, \) such events are mutually exclusive, and thus we can compute the original PPR value by \( \pi(s, t) = \sum_{\ell=0}^{\infty} \pi_{\ell}(s, t). \) Furthermore, we can truncate the summation to \( \hat{\pi}(s, t) = \sum_{\ell=0}^{L} \pi_{\ell}(s, t) \) where \( L = \log_{1-\alpha} \theta, \) which only adds a small additive error \( \theta \) to the final estimator and \( \hat{\theta} = \hat{\theta}(\epsilon). \) Secondly, we introduce randomization into the push operation of the Backward Search algorithm to reduce the query cost. Recall that in the vanilla Backward Search algorithm, each push operation transfers an \( (1-\alpha) \) fraction of the probability mass from the current node \( u \) to each of its in-neighbors. This operation is expensive as it touches all the in-neighbors of \( u \) and thus leads to the \( O(d) \) overhead. We avoid this complexity by pushing the probability mass to a small random subset of \( u \)’s in-neighbors. The probability for including an in-neighbor \( u \) depends on the random walk probability to reach \( u \), and uses \( r^f(s, u) \) to denote the probability mass to be distributed at node \( u, \) and \( r^f(s, u) \) to denote the probability mass that will stay at node \( u \) permanently.
on the out-degree of $u$. We show that this randomized push operation is unbiased and has bounded variance, which enables us to derive probabilistic bounds for both additive error and relative error.

**Sorted adjacency lists.** Before presenting our algorithm, we assume the in-adjacency list of each node is sorted in the ascending order of the out-degrees. More precisely, let $\{u_1, \ldots, u_d\}$ denote the in-adjacency list of $v$. We assume that $d_{\text{out}}(u_1) \leq \ldots \leq d_{\text{out}}(u_d)$. We claim that it is possible to sort all the in-adjacency lists in $O(m+n)$ time, which is asymptotically the same as reading the graph into the memory. This means we can pre-sort the graph as we read the graph into the main memory without increasing the asymptotic cost. More specifically, we construct a tuple $(u,v,d_{\text{out}}(u))$ for each edge $(u,v) \in E$, and use counting sort to sort $(u,v,d_{\text{out}}(u))$ tuples in the ascending order of $d_{\text{out}}(u)$. Since each $d_{\text{out}}(u)$ is bounded by $n$, and there are $m$ tuples, the cost of counting sort is bounded by $O(m+n).$ Finally, for each $u, v, d_{\text{out}}(u)$, we append $u$ to the end of $v$’s in-adjacency list. This preprocessing algorithm runs in $O(m+n)$ time, which is asymptotically the same as reading the graph structure.

**Algorithm description.** Algorithm 2 illustrates the pseudocode of the RBS algorithm. Consider a directed graph $G = (V,E)$, target node $t$, and a teleport probability $\alpha$. The algorithm takes in two additional parameters: error parameter $\theta$ and sampling function $\lambda(u)$. We can manipulate these two parameters to obtain the additive or relative error guarantees. Recall that $\theta$ denotes the Big-Oh notation ignoring the log factors. For single-target PPR query with additive error, we set $\theta$ to be $O(\epsilon)$, the maximum additive error allowed, and $\lambda(u) = \sqrt{d_{\text{out}}(u)}$. For relative error, we set $\theta = O(\delta)$, the threshold for constant relative error guarantee, and $\lambda(u) = 1$.

For initialization, we set the maximum number of hops $L$ to be $\log_{1-\alpha} \theta$ (line 1). Then we initialize the estimators $\hat{\pi}(s,t) = 0$ for $\forall s \in V$ and $\forall t \in \{0,L\}$ except for $\pi_0(t,t) = \alpha$ (Line 2-3). We iteratively push the probability mass from level 0 to $L-1$ (line 4). At level $\ell$, we pick a node $v \in V$ with non-zero estimator $\hat{\pi}_\ell(v,t)$, and push $\hat{\pi}_\ell(v,t)$ to a subset of its in-neighbors (line 5). More precisely, for an in-neighbor $u$ with out-degree $d_{\text{out}}(u) \leq \frac{\lambda(u)(1-\alpha)\hat{\pi}_\ell(v,t)}{\theta}$, we deterministically push a probability mass of $\frac{(1-\alpha)\hat{\pi}_\ell(v,t)}{\theta}$ to the $(\ell+1)$-hop estimator $\hat{\pi}_{\ell+1}(u,t)$ (lines 6-7). Recall that the in-neighbors of $v$ are sorted according to their out-degrees. Therefore, we can sequentially scan the in-adjacency list of $v$ until we encounter the first in-neighbor $u$ with $d_{\text{out}}(u) > \frac{\lambda(u)(1-\alpha)\hat{\pi}_\ell(v,t)}{\theta}$. For in-neighbors with higher out-degrees, we generate a random number $r$ from $(0,1)$, and push a probability mass of $\frac{\theta}{\lambda(u)}$ to $\hat{\pi}_{\ell+1}(u,t)$ for each in-neighbor $u$ with $d_{\text{out}}(u) \leq \frac{\lambda(u)(1-\alpha)\hat{\pi}_\ell(v,t)}{\theta}$ (lines 8-10). Similarly, we can sequentially scan the in-adjacency list of $v$ until we encounter the first in-neighbor $u$ with $d_{\text{out}}(u) > \frac{\lambda(u)(1-\alpha)\hat{\pi}_\ell(v,t)}{\theta}$. Finally, after all $L$ hops are processed, we return $\hat{\pi}(s,t) = \sum_{\ell=0}^{L} \hat{\pi}_\ell(s,t)$ for each $s \in V$.

![Figure 1: Sketch of Randomized Backward Search](image)

**Algorithm 2: Randomized Backward Search**

| Input: | Directed graph $G = (V,E)$ with sorted adjacency lists, target node $t \in V$, teleport probability $\alpha$, error parameter $\theta$, sampling function $\lambda(u)$ |
| --- | --- |
| Output: | Estimator $\hat{\pi}(s,t)$ for each $s \in V$ |
| 1. | $L \leftarrow \log_{1-\alpha} \theta$; |
| 2. | $\hat{\pi}_0(s,t) \leftarrow 0$ for $\ell = 0, \ldots, L, s \in V$; |
| 3. | $\pi_0(t,t) \leftarrow \alpha$; |
| 4. | for $\ell = 0$ to $L-1$ do |
| 5. | for each $v \in V$ with non-zero $\hat{\pi}_\ell(v,t)$ do |
| 6. | for each $u \in N_{in}(v)$ and $d_{\text{out}}(u) \leq \frac{\lambda(u)(1-\alpha)\hat{\pi}_\ell(v,t)}{\theta}$ do |
| 7. | $\hat{\pi}_{\ell+1}(u,t) \leftarrow \hat{\pi}_{\ell+1}(u,t) + \frac{(1-\alpha)\hat{\pi}_\ell(v,t)}{d_{\text{out}}(u)}$; |
| 8. | $r \leftarrow \text{rand}(0,1)$; |
| 9. | for each $u \in N_{in}(v)$ and $d_{\text{out}}(u) \leq \frac{\lambda(u)(1-\alpha)\hat{\pi}_\ell(v,t)}{\theta}$ do |
| 10. | $\hat{\pi}_{\ell+1}(u,t) \leftarrow \hat{\pi}_{\ell+1}(u,t) + \frac{\theta}{\lambda(u)}$ |
| 11. | return all non-zero $\hat{\pi}(s,t) = \sum_{\ell=0}^{L} \hat{\pi}_\ell(s,t)$ for each $s \in V$; |

**4 ANALYSIS**

In this section, we analyze the theoretical property of the RBS algorithm. Recall that for single-target PPR query with additive error, we set $\lambda(u)$ to be $\sqrt{d_{\text{out}}(u)}$ and $\theta = O(\epsilon)$ to be the error bound. For relative error, we set $\lambda(u) = 1$ and $\theta = O(\delta)$, the threshold for constant relative error guarantee. Recall that $\hat{O}$ denotes the Big-Oh notation ignoring the log factors. Theorem 4.1 and 4.2 provide the theoretical results of running time and error guarantee for the RBS algorithm with additive and relative error, respectively.

**Theorem 4.1.** By setting $\lambda(u) = 1$ and $\theta = O(\delta)$, Algorithm 2 answers the single-target PPR queries with a relative error threshold $\delta$ with high probability. The expected worst-case time cost is bounded by $\hat{O}\left(\frac{n \pi(t,t)}{\delta}\right)$. If the target node $t$ is chosen uniformly at random from $V$, the time cost becomes $\hat{O}\left(\frac{1}{\delta}\right)$. 

---
With the help of Lemma 4.3, 4.4, and 4.5, we are able to prove worst case. Hence, we do not extract $\lambda_0$ process from level $c$ return all $c_0$ in [58]. We define the following PPR matrix computation.

Lemma 4.3. Algorithm 2 returns an estimator $\hat{\pi}(s, t)$ for each $\pi(s, t)$ such that $E[\hat{\pi}(s, t)] = \pi(s, t)$ holds for $\forall s \in V$ and $\ell \in \{0, 1, 2, ..., L\}$.

Next, we bound the variance of the $\ell$-hop estimators.

Lemma 4.4. For any $s \in V$ and $\ell \in \{0, 1, 2, ..., L\}$, the variance of each estimator $\hat{\pi}(s, t)$ obtained by Algorithm 2 satisfies that:

1) If we set $\lambda(u) = 1$, then $\text{Var}[\hat{\pi}(s, t)] \leq 0\pi(s, t)$.
2) If we set $\lambda(u) = \sqrt{d_{out}(u)}$, then $\text{Var}[\hat{\pi}(s, t)] \leq a\tilde{o}(d_{out}(u))$.

The following lemma analyzes the expected query cost of the RBS algorithm.

Lemma 4.5. Let $C_{total}$ denote the total cost during the whole push process from level 0 to level (L−1), the expected time cost of algorithm 2 can be expressed as that:

$$E[C_{total}] \leq \frac{1}{a\tilde{o}} \sum_{u \in V} \lambda(u) \cdot \pi(u, t).$$

Note that $\lambda(u)$ is an adjustable sampling function that balances the variance and time cost. In our case, it is a function of node $u$. Hence, we do not extract $\lambda(u)$ from the last summation symbol. With the help of Lemma 4.3, 4.4, and 4.5, we are able to prove Theorem 4.1 and 4.2. For the sake of readability, we defer all proofs to the appendix.

5 APPLICATIONS

In this section, we discuss how the RBS algorithm improves the three concrete applications mentioned in Section 1: heavy hitters PPR query, single-source SimRank computation, and approximate PPR matrix computation.

Heavy hitters PPR computation. Following the definitions in [58], we define the $c$-approximate heavy hitter as follows.

Definition 5.1. ($c$-approximate heavy hitter). Given a real value $0 < \phi < 1$, a constant real value $0 < c < 1$, two nodes $s, t$ in $V$, we say that $s$ is:

- a $c$-absolute $\phi$-heavy hitter of $t$ if $\pi(s, t) > (1 + c)\phi \cdot n\pi(t)$;
- a $c$-permissible $\phi$-heavy hitter of $t$ if $(1 - c)\phi \cdot n\pi(t) \leq \pi(s, t) \leq (1 + c)\phi \cdot n\pi(t)$;
- not a $c$-$\phi$-heavy hitter of $t$, otherwise.

Given a target node $t$, a heavy hitter algorithm is required to return all $c$-absolute $\phi$-heavy hitters and to exclude all nodes that are not a $c$-$\phi$-heavy hitter of $t$. Wang et al. utilizes the traditional Backward Search to derive the $c$-approximate heavy hitter[58]. The time complexity is $O \left( \sum_{u \in V} \frac{d_{out}(u) \cdot \pi(u, t)}{\phi \cdot n\pi(t)} \right)$ in the worst case.

On the other hand, by running the RBS algorithm with relative error threshold $\delta = c\tilde{O}(t)$, we can obtain the $c$-approximate heavy hitter of node $t$ with high probability. By Theorem 4.1, the running time of RBS algorithm is bounded by $O \left( \frac{n\pi(t)}{\delta} \right)$. Note that this complexity is optimal up to log factors, as there may be $O \left( \frac{n}{\epsilon} \right)$ heavy hitters for node $t$. Therefore, RBS achieves optimal worst-case query complexity for the $c$-approximate heavy hitter problem.

Single-source SimRank computation. Recall that if we revert the direction of every edge in the graph, the SimRank similarity $s(u, v)$ of node $u$ and $v$ equals to the probability that two $\alpha$-discounted random walks from $u$ and $v$ visit at the same node $w$ with the same steps. PRSim [60] and SLING [53], two state-of-the-art SimRank algorithms, formulate the SimRank in terms of $\ell$-hop PPR: $s(u, v) = \frac{1}{(1 - \sqrt{\beta})} \sum_{\ell=0}^{\infty} \sum_{w \in V} \pi(s, t) \cdot \pi(s, w) \cdot \pi(t, w) \cdot \pi(t, w)$. For each $w, v \in V, \ell = 0, \ldots, \infty$, while PRSim [60] only precomputes an approximation of $\pi(s, w)$ for node $w$ with large PageRanks. Recall that the running time of the SimRank algorithm on a random node $t$ is $O \left( \frac{n}{\epsilon} \right)$. It follows that the total precomputation cost for SLING or PRSim is bounded by $O \left( \frac{n^2}{\epsilon^2} \right)$. However, according to Theorem 4.2, if we replace the Backward Search algorithm with the new RBS algorithm with additive error, the running time for a random target node $t$ is improved to $O \left( \frac{n\sqrt{d}}{\epsilon^2} \right)$. And thus the total precomputation time is improved to $O \left( \frac{n\sqrt{d}}{\epsilon^2} \right)$.

Approximate PPR matrix. As mentioned in Section 1, computing the approximate PPR matrix is the computational bottleneck for various graph embedding and graph neural network algorithms, such as HOPE [45], STRAP [64], PPRGo [63] and GDC [29]. However, computing the PPR matrix is costly; Applying the Power Method to $n$ nodes takes $O(nm)$ time, which is infeasible on large graphs. PPRGo [63] proposes to apply Forward Search to each source node $s \in V$ to construct the approximate PPR matrix; While STRAP [64] employs Backward Search to each target node $t \in V$ to compute $\pi(s, t), s \in V$ and then put $\pi(s, t)$ into an inverted list indexed by $s$. For the former approach, recall that the Forward Search only guarantees an additive error of $c\tilde{O}(t)$ for the estimator of $\pi(s, t)$ [6], which is undesirable for nodes with high degrees. On the other hand, the latter approach incurs a running time of $O \left( \frac{n^2}{\epsilon^2} \right)$ for computing an approximate PPR matrix with additive error $\epsilon$. By replacing the Backward Search algorithm with the new RBS algorithm with additive error, we can improve the complexity to $O \left( \frac{n\sqrt{d}}{\epsilon^2} \right)$, which is sub-linear to the number of edges $m$.

6 EXPERIMENTS

This section experimentally evaluates the performance of RBS against state-of-the-art methods. Section 6.1 presents the empirical study for single-target PPR queries. Section 6.2 applies RBS to three concrete applications to show its effectiveness. The information of the datasets we used is listed in table 3. All datasets are obtained...
6.1 Single-Target Query

Metrics and experimental setup. For a given query node, we apply the Power Method [46] with $L = [\log_{10}^{-\alpha}(10^{-7})]$ iterations to obtain the ground truths of the single-target queries based on the following formula: $\hat{\pi}(u,v) = (1-\alpha)\hat{\pi}(u)P + \alpha \cdot \epsilon$. To evaluate the additive error, we consider $\text{MaxAdditiveErr}$, which is defined as:

$$\text{MaxAdditiveErr} = \max_{v_i \in V} |\hat{\pi}(v_i, t) - \hat{\pi}(v_i, t)|$$

where $\hat{\pi}(v_i, t)$ is the estimator of PPR value $\pi(v_i, t)$. For relative error, there lacks a practical metric that evaluates the relative error threshold $\delta$. Hence, we consider the $\text{Precision}@k$, which evaluates the quality of the single-target top-$k$ queries. More precisely, let $V_k$ and $\hat{V}_k$ denote the set containing the nodes with single-target top-$k$ queries returned by the ground truth and the approximation methods, respectively. $\text{Precision}@k$ is defined as the percentage of nodes in $\hat{V}_k$ that coincides with the actual top-$k$ results $V_k$. In our experiment, we set $k = 50$. On each dataset, we sample 100 target query nodes according to their degrees and report the averages of the $\text{MaxAdditiveErr}$ and $\text{Precision}@k$ for each evaluated methods.

Results. We evaluate the performance of RBS against Backward Search [38] for the single-target PPR query. For RBS, we set $\lambda(u) = \sqrt{\text{deg}_{out}(u)}$ and $\theta = \epsilon$ for additive error, $\lambda(u) = 1$ and $\theta = \delta$ for relative error. For Backward Search (BS), we set $\epsilon = \delta$ for relative error. We vary the additive error parameter $\epsilon$ and relative threshold $\delta$ from 0.1 to $10^{-6}$ in both experiments. Following previous work [56], we set the decay factor $\alpha = 0.2$.

Figure 2 shows the tradeoffs between the $\text{MaxAdditiveErr}$ and the query time for the additive error experiments. Figure 3 presents the tradeoffs between $\text{Precision}@k$ and the query time for the relative error experiments. In general, we observe that under the same approximation quality, RBS outperforms BS by orders of magnitude in terms of query time. We also observe that RBS offers a more significant advantage over BS when we need high-quality estimators. In particular, to obtain an additive error of $10^{-6}$ on IT, we observe a 100x query time speedup for RBS. From Figure 3, we also observe that the precision of RBS with relative error approaches 1 more rapidly, which concurs with our theoretical analysis.

6.2 Applications

We now evaluate RBS in three concrete applications: heavy hitters query, single-source SimRank computation and approximate PPR matrix approximation.

Heavy hitters PPR query. Recall that in section 5, the heavy hitter of target node $t$ is defined as node $s$ with $\pi(s, t) > \phi \cdot n \pi(t)$, where $\phi$ is a parameter. Following [58], we fix $\phi = 10^{-5}$ in our experiments. Since the number of true heavy hitters is oblivious to the algorithm, we use the $F1$ score instead of precision to evaluate the approximate algorithms, where the $F1$ score is defined as $F1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$. We set $\lambda(u) = 1$ and $\theta = \delta$ for RBS, and $\epsilon = \delta$ for BS, and set $\delta$ from 0.1 to $10^{-6}$ to illustrate how the $F1$ score varies with the query time.

In general, RBS returns a higher $F1$ score than BS does, given the same amount of query time. In particular, to achieve an $F1$ score of 1 on the IC dataset, RBS requires a query time that is 80x less than BS does. The results suggest that by replacing BS with RBS, we can improve the performance of heavy hitters PPR queries.

Single-source SimRank computation. As claimed in section 5, SimRank can be expressed in terms of PPR values as below:

$$s(u, v) = \frac{1}{1 - \gamma \lambda(u) \theta} \sum_{t=0}^{\infty} \sum_{w \in V} \pi(t, w) \pi(u, t) \eta(w)$$

The state-of-the-art single-source SimRank methods, SLING [53] and PRSim [60], pre-compute the $t$-hop PPR values $\pi(v, w)$ with the Backward Search algorithm and store them in index. We take PRSim as example to show the benefit from replacing BS with RBS.

Following [60], we evaluate the tradeoffs between the preprocessing time with $\text{MaxAdditiveErr}@50$, the maximum additive error of single-source top-50 SimRank values for a given query node. PRSim has one error parameter $\epsilon$. We vary it in $0.5, 0.1, 0.05, 0.01, 0.005$ to plot the tradeoffs. We sample 100 query nodes uniformly and use the pooling method in [60] to derive the actual top-50 SimRank values for each query node, and return the average of the $\text{MaxAdditiveErr}@50$ for each approximate method. Figure 5 illustrates the tradeoffs between the preprocessing time with $\text{MaxAdditiveErr}@50$. We observe that by replacing BS with RBS, we can achieve a significantly lower preprocessing time without increasing the approximation quality of the single-source SimRank results. This suggests RBS also outperforms BS for computing $t$-hop PPR to a target node.

Approximate PPR matrix. An approximate PPR matrix consists of PPR estimators for all pairs of nodes, and is widely used in graph learning. Recall that PPRGo [63] proposes to apply Forward Search (FS) to each source node $s \in V$ to construct the approximate PPR matrix, while STRAP [64] employs Backward Search (BS) to each target node $t \in V$ to compute $\pi(s, t)$, $s \in V$ and then put $\pi(s, t)$ into an inverted list indexed by $s$. We evaluate RBS against BS and FS in terms of additive error and running time for computing the approximate PPR matrix.

Due to the scalability limitation of the Power Method, we conduct this experiment on two small datasets: GQ and AS. We set $\lambda(u) = \sqrt{\text{deg}_{out}(u)}$ and $\theta = \epsilon$ for RBS, and vary the additive error parameter $\epsilon$ in RBS from 0.1 to $10^{-6}$. Similarly, we vary the parameter $\epsilon$ of BS from 0.1 to $10^{-6}$. Figure 6 shows the tradeoffs between $\text{MaxAdditiveErr}$ of PPR values of all node pairs and the running time for the three methods. We first observe that given the same error budget, BS outperforms FS in terms of running time. This result concurs with our theoretical analysis that FS guarantees an additive error of $\epsilon \text{deg}_{out}(t)$ while BS guarantees an additive error of $\epsilon \cdot \text{deg}(t)$. Therefore, it may be worthy of taking the extra step to convert the single-target PPR results into inverted lists indexed by the source nodes. On the other hand, by replacing BS with RBS, we can further improve the tradeoffs between the running time and the approximation quality, which demonstrates the superiority of ours.
We present an algorithm RBS to compute approximate single-target PPR query, single-source SimRank computation, and scalable graph neural networks. The experiments suggest that RBS outperforms PPR query, single-source SimRank computation, and scalable graph neural networks.

7 CONCLUSION

In this paper, we study the single-target PPR query, which measures the importance of a given target node $t$ to every node $s$ in the graph. We present an algorithm RBS to compute approximate single-target PPR query with optimal computational complexity. We show that RBS improves three concrete applications in graph mining: heavy hitters PPR query, single-source SimRank computation, and scalable graph neural networks. The experiments suggest that RBS outperforms the state-of-the-art algorithms in terms of both efficiency and precision on real-world benchmark datasets. For future work, we note that a few works combine the Backward Search algorithm with the Monte-Carlo algorithm to obtain near-optimal query cost for single-pair queries \cite{37,40}. An interesting open problem is whether we can replace the Backward Search algorithm with RBS to further improve the complexity of these algorithms.

8 ACKNOWLEDGEMENTS

This research is supported by National Natural Science Foundation of China (No. 61832017, No. 61972401, No. 61932001, No.U1936205), by Beijing Outstanding Young Scientist Program NO. BJWZYJH10219100020098, and by the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China under Grant 18XNLG2. Junhao Gan is supported by Australian Research Council (ARC) DECE RE19011118. Sibo Wang is also supported by Hong Kong RGC ECS Grant No. 24203419. Zengfeng Huang is supported by Shanghai Science and Technology Commission Grant No. 17JC1420200, and by Shanghai Sailing Program Grant No. 18YF1401200.

REFERENCES

\cite{1} \cite{2} \cite{3} \cite{4} \cite{5} \cite{6} \cite{7} \cite{8}
Manish S. Gupta, Amit Pathak, and Soumen Chakrabarti. Fast algorithms for rank computation in entity-relation graphs. In WWW, pages 571–580, 2007.

Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In IALP, pages 693–703, Springer, 2000.

Mustafa Coskun, Ananth Grama, and Mehmet Kusyuturk. Efficient processing of network proximity queries via chebyshev acceleration. In KDD, pages 1515–1524, 2016.

Dániel Fagras, Bálint Récz, Károly Csólagyári, and Tamás Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. Internet Mathematics, 2(3):333–358, 2005.

Yasuhiro Fujwara, Makoto Nakatsui, Makoto Onizuka, and Masaru Katsugrewa. Fast and exact top-k search for random walk with restart. PVLDB, 5(5):442–453, 2012.

Yasuhiro Fujwara, Makoto Nakatsui, Hiroaki Shiokawa, Takeshi Mishima, and Makoto Onizuka. Efficient ad-hoc search for personalized pagerank. In SIGMOD, pages 445–456, 2013.

Yasuhiro Fujwara, Makoto Nakatsui, Hiroaki Shiokawa, Takeshi Mishima, and Makoto Onizuka. Fast and exact top-k algorithm for pagerank. In AAAI, 2013.

Yasuhiro Fujwara, Makoto Nakatsui, Takashi Yamamuro, Hiroaki Shiokawa, and Makoto Onizuka. Efficient personalized pagerank with accuracy assurance. In KDD, pages 15–23, 2012.

Tao Guo, Xin Cao, Gao Cong, Jiaheng Lu, and Xuemin Lin. Distributed algorithms on exact personalized pagerank. In SIGMOD, pages 479–494, 2017.

Manish S. Gupta, Anil Pathak, and Soumen Chakrabarti. Fast algorithms for top-k personalized pagerank queries. In WWW, pages 1225–1226, 2008.

Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesht Sharma, Dong Wang, and Reza Zadeh. Whiff: The who to follow service at twitter. In WWW, pages 505–514, 2013.

Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In SIGKDD, pages 538–542, 2003.

Glen Jeh and Jennifer Widom. Scaling personalized web search. In WWW, pages 271–279, 2003.

Minhao Jiang, Ada Wai-Chee Fu, and Raymond Chi-Wing Wong. Probesim: scalable single-source and top-k simrank computations. In ICDE, pages 789–804, 2017.

Thomas N Kipf and Max Welling. Scalable convolutional networks for graph convolutional networks. ICLR, 2017.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Personalized embedding propagation: Combining neural networks on graphs with convolutional networks. In ICDM, pages 505–514, 2017.

Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning, 2019.

Mitsuru Kusumoto, Takamori Maehara, and Ken-ichi Katayama. Scalable similarity search for simrank. In SIGMOD, pages 325–336, 2014.

Pei Lei, Laks V S Lakshmanan, and Jeffrey Xu Yu. On top-k structural similarity search. In ICDE, pages 774–785, 2012.

Lina Li, Cuipeng Li, Chen Hong, and Xiaoyong Du. Mapreduce-based simrank computation and its application in social recommender system. In Big Data (BigData Congress), 2013 IEEE International Congress on, 2013.

Zhenguo Li, Yanfeng Sun, Lijun Luan, and Xin Sun. Enhancing Pagerank to rank microblogs. In WSDM, pages 235–244, 2013.

David Liben-Nowell and Jon M. Kleinberg. The link prediction problem for social networks. In ICBM, pages 556–559, 2003.

Yu Liu, Bolong Zheng, Xiaodong He, Zhewei Wei, Xiaokui Xiao, Kai Zheng, and Jiuhang Lu. Probesim: scalable single-source and top-k simrank computations on large graphs. In PVLDB, 10(9):937–948, 2017.

Ruoming Jin, Victor E Lee, and Hui Hong. Attractor ranking: ranking network role proximities. In KDD, pages 1517–1585, 2015.

Nikita Sinir and Jiawei Han. Scaling personalized pagerank with accuracy assurance. In WWW, pages 589–599, 2017.

Anton Tsitsulin, Davide Mottin, Pasquale Storelli, and Emmanuel Müller. Versatile graph embeddings from similarity measures. In WWW, pages 539–548, 2016.

Hongyang Zhang, Peter A Lofgren, and Ashish Goel. Approximate personalized pagerank estimation: From average-case to worst-case. In SIGKDD, pages 113–121, 2015.

Boyuan Tian and Xiaokui Xiao. SLING: A near-optimal index structure for simrank. In SIGMOD, pages 1598–1599, 2014.

Mustafa Coskun, Ananth Grama, and Mehmet Kusyuturk. Efficient method for billion-scale random walk with restart. In SIGMOD, pages 922–930, 2011.

Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. Theoretical Computer Science, 350(3):552–582, 2006.
A APPENDIX

A.1 Proof of Lemma 4.3

Proof. During a push operation through edge \((u, v)\) from level \(\ell\) to level \((\ell+1)\), we denote \(X_{\ell+1}(u, v)\) as \(\tilde{\pi}_{\ell+1}(u, t)\)’s increments caused by this push. According to Algorithm 2, \(X_{\ell+1}(u, v)\) is assigned as \(\frac{1}{\text{out}(u)}\) deterministically if \(\lambda u\langle 1-\alpha\tilde{\pi}_{\ell}(v, t)\rangle < \frac{\theta}{\lambda(u)}\). Otherwise, \(X_{\ell+1}(u, v)\) will be \(\frac{\theta}{\lambda(u)}\) with probability \(\frac{\lambda(u)(1-\alpha\tilde{\pi}_{\ell}(v, t))}{\lambda(u)+\frac{\theta}{\lambda(u)}\text{out}(u)}\), or 0 with the left probability. If we use \(\{\tilde{\pi}_{\ell}\}\) to denote the set of \(\tilde{\pi}_{\ell}(v, t)\) for all \(v \in V\), the expectation of \(X_{\ell+1}(u, v)\) conditioned on all estimators \(\{\tilde{\pi}_{\ell}\}\) can be derived that

\[
E[X_{\ell+1}(u, v) \mid \{\tilde{\pi}_{\ell}\}] = \frac{1}{\lambda(u)} \left[ \frac{1}{\lambda(u)} + \frac{\theta}{\lambda(u)} \right] = \frac{\theta}{\lambda(u)}.
\]

Because \(\tilde{\pi}_{\ell+1}(u, t) = \sum_{v \in \text{Nout}(u)} X_{\ell+1}(u, v)\), the conditional expectation of \(\tilde{\pi}_{\ell+1}(u, t)\) conditioned on the value of all estimators \(\{\tilde{\pi}_{\ell}\}\) at \(\ell\)-th level can be derived that

\[
E[\tilde{\pi}_{\ell+1}(u, t) \mid \{\tilde{\pi}_{\ell}\}] = \sum_{v \in \text{Nout}(u)} \frac{1}{\lambda(u)} (1-\alpha)\tilde{\pi}_{\ell}(v, t).
\]

Based on equation (9), we have

\[
E[\tilde{\pi}_{\ell+1}(u, t) \mid \{\tilde{\pi}_{\ell}\}] = \sum_{v \in \text{Nout}(u)} \frac{(1-\alpha)\tilde{\pi}_{\ell}(v, t)}{\text{out}(u)}. \tag{10}
\]

Because \(E[\tilde{\pi}_{\ell}(x, t)] = \pi_0(x, t)\) holds for all \(x \in V\) and \(i = 0\) in the initial state, because \(E[\tilde{\pi}_{\ell}(t, t)] = \pi_{t}(t, t)\) and \(E[\tilde{\pi}_{\ell}(u, t)] = \pi_{t}(u, t) = 0\) (\(u \neq t\)). Assume \(E[\tilde{\pi}_{\ell}(x, t)] = \pi_{t}(x, t)\) holds for all \(x \in V\) and \(i \leq \ell\). We can derive that

\[
E[\tilde{\pi}_{\ell+1}(u, t)] = \sum_{v \in \text{Nout}(u)} \frac{(1-\alpha)\pi_{t}(v, t)}{\text{out}(u)} = \pi_{t+1}(u, t),
\]

which testifies the unbiasedness. \(\square\)

A.2 Proof of Lemma 4.4

Proof. During each push operation, the randomness comes from the second scenario: the random variable \(X_{\ell+1}(u, v)\) is independent with each other because of the independent generation for the random number \(r\) in Algorithm 2. Applying \(\tilde{\pi}_{\ell+1}(u, t) = \sum_{v \in \text{Nout}(u)} X_{\ell+1}(u, v)\), the conditional variance of \(\tilde{\pi}_{\ell+1}(u, t)\) is followed that

\[
\text{Var} \left[ \tilde{\pi}_{\ell+1}(u, t) \mid \{\tilde{\pi}_{\ell}\} \right] = \sum_{v \in \text{Nout}(u)} \text{Var}[X_{\ell+1}(u, v) \mid \{\tilde{\pi}_{\ell}\}] \leq \frac{\alpha^2}{\lambda(u)} \sum_{v \in \text{Nout}(u)} \frac{(1-\alpha)\pi_{t}(v, t)}{\text{out}(u)}. \tag{11}
\]

By the total variance law, \(\text{Var} [\tilde{\pi}_{\ell+1}(u, t) \mid \{\tilde{\pi}_{\ell}\}] = \text{Var} \{\tilde{\pi}_{\ell+1}(u, t) \mid \{\tilde{\pi}_{\ell}\}\} + \text{Var} [E[\tilde{\pi}_{\ell+1}(u, t) \mid \{\tilde{\pi}_{\ell}\}]\). Based on equation (11) and the unbiasedness of \(\tilde{\pi}_{\ell+1}(u, t)\), we have

\[
\text{Var} [E[\tilde{\pi}_{\ell+1}(u, t) \mid \{\tilde{\pi}_{\ell}\}]] \leq \frac{\alpha^2}{\lambda(u)} \sum_{v \in \text{Nout}(u)} \frac{(1-\alpha)\pi_{t}(v, t)}{\text{out}(u)} = \frac{\alpha^2}{\lambda(u)} \pi_{t+1}(u, t).
\]

Meanwhile, applying equation (10), we can derive

\[
\text{Var}[E[\tilde{\pi}_{\ell+1}(u, t) \mid \{\tilde{\pi}_{\ell}\}]] = \frac{(1-\alpha)^2}{\text{out}(u)} \cdot \text{Var} \sum_{v \in \text{Nout}(u)} \tilde{\pi}_{\ell}(v, t).
\]

The convexity of variance implies that:

\[
\text{Var} \sum_{v \in \text{Nout}(u)} \tilde{\pi}_{\ell}(v, t) \leq \text{out}(u) \cdot \sum_{v \in \text{Nout}(u)} \text{Var} \tilde{\pi}_{\ell}(v, t).
\]

Therefore, we can rewrite \(\text{Var}[E[\tilde{\pi}_{\ell+1}(u, t) \mid \{\tilde{\pi}_{\ell}\}]]\) as below:

\[
\text{Var}[E[\tilde{\pi}_{\ell+1}(u, t) \mid \{\tilde{\pi}_{\ell}\}]] \leq \frac{(1-\alpha)^2}{\text{out}(u)} \cdot \sum_{v \in \text{Nout}(u)} \text{Var} \tilde{\pi}_{\ell}(v, t). \tag{13}
\]

Applying equation (12) and equation (13), we can derive that

\[
\text{Var} [\tilde{\pi}_{\ell+1}(u, t)] \leq \frac{\alpha^2}{\lambda(u)} \cdot \pi_{t+1}(u, t) + \frac{(1-\alpha)^2}{\text{out}(u)} \cdot \sum_{v \in \text{Nout}(u)} \text{Var} \tilde{\pi}_{\ell}(v, t).
\]

\[
\text{Var} [\tilde{\pi}_{\ell+1}(u, t)] \leq \frac{\alpha^2}{\lambda(u)} \cdot \pi_{t+1}(u, t) + \frac{(1-\alpha)^2}{\text{out}(u)} \cdot \pi_{t+1}(u, t) = \frac{\alpha^2}{\lambda(u)} \cdot \pi_{t+1}(u, t).
\]

For relative error, we set \(\lambda(u) = 1\) that \(\text{Var} [\tilde{\pi}_{\ell+1}(u, t)] \leq \theta \cdot \pi_{t+1}(u, t)\).

For additive error, we set \(\lambda(u) = \sqrt{\text{out}(u)}\), and it follows that

\[
\text{Var} [\tilde{\pi}_{\ell+1}(u, t)] \leq \theta \cdot \pi_{t+1}(u, t) = \theta \cdot \sum_{v \in \text{Nout}(u)} \frac{(1-\alpha)\pi_{t}(v, t)}{\text{out}(u)}.
\]

Note that \(1-\alpha)\pi_{t}(v, t) \leq \frac{\alpha^2}{\lambda(u)} \cdot \text{out}(u)\) in the randomized scenario. So,

\[
\text{Var} [\tilde{\pi}_{\ell+1}(u, t)] \leq \theta \cdot \sum_{v \in \text{Nout}(u)} \frac{\alpha^2}{\lambda(u)} \cdot \text{out}(u) = \alpha^2 \cdot \theta. \tag{14}
\]

\(\square\)
A.3 Proof of Lemma 4.5

Proof. Let \( C_{\ell t} (u, v) \) denote the cost of one push operation through edge \((u, v)\) from level \( \ell \) to \( \ell + 1 \). If \( \frac{1 - \theta \hat{\pi}_t (u, v)}{d_{out}(u)} \geq \frac{\alpha \theta}{\lambda(u)} \), the push operation will be guaranteed once. Otherwise, the push happens with probability \( \frac{\lambda(u) - (1 - \theta) \hat{\pi}_t (u, v)}{\alpha \theta d_{out}(u)} \). Hence, we can derive the expectation of \( C_{\ell t+1} (u, v) \) conditioned on the value of all estimators \( \{ \hat{x}_t \} \) at \( \ell \)-th level \( \{ \hat{x}_t \} \)

\[
E[C_{\ell t+1}(u, v) \mid \{ \hat{x}_t \}] = \begin{cases} 
1, & \text{if } \frac{(1 - \theta) \pi_t (u, v)}{\lambda(u) - (1 - \theta) \pi_t (u, v)} \geq \frac{\alpha \theta}{\lambda(u)}, \\
1 \frac{\lambda(u) - (1 - \theta) \pi_t (u, v)}{\alpha \theta d_{out}(u)}, & \text{otherwise}.
\end{cases}
\]

Note that if \( \frac{(1 - \theta) \pi_t (u, v)}{\lambda(u) - (1 - \theta) \pi_t (u, v)} \geq \frac{\lambda(u) - (1 - \theta) \pi_t (u, v)}{\alpha \theta d_{out}(u)} \), the conditional expectation \( E[C_{\ell t}(u, v) \mid \{ \hat{x}_t \}] \) satisfies that \( E[C_{\ell t}(u, v) \mid \{ \hat{x}_t \}] = 1 \leq \frac{\lambda(u) - (1 - \theta) \pi_t (u, v)}{\lambda(u) - (1 - \theta) \pi_t (u, v)} \). Thus, we can derive that \( E[C_{\ell t}(u, v) \mid \{ \hat{x}_t \}] \leq \frac{\lambda(u) - (1 - \theta) \pi_t (u, v)}{\lambda(u) - (1 - \theta) \pi_t (u, v)} \) always holds. Applying the unbiasedness of \( \hat{x}_t (u, v) \) according to Lemma 4.3, we have

\[
E[C_{\ell t+1}(u, v) \mid \{ \hat{x}_t \}] = \frac{(1 - \theta) \pi_t (u, v)}{\lambda(u) - (1 - \theta) \pi_t (u, v)} \frac{\lambda(u) - (1 - \theta) \pi_t (u, v)}{\alpha \theta d_{out}(u)}.
\]

Recall that \( C_{\ell t} \) denotes the total cost in the whole process and \( C_{\ell t} = \sum_{i=1}^{\ell} \sum_{u \in V} \sum_{v \in N_{out}(v)} C_{i}(u, v) \). The expectation of \( C_{\ell t} \) can be derived that

\[
E[C_{\ell t}] = \frac{1}{\alpha \theta} \sum_{u \in V} \sum_{v \in N_{out}(v)} \lambda(u) - (1 - \theta) \pi_t (u, v).
\]

According to the property of \( \ell \)-hop PPR that \( \sum_{i=0}^{\infty} \pi_i (u, v) = \pi(u, v) \),

\[
E[C_{\ell t}] \leq \frac{1}{\alpha \theta} \sum_{u \in V} \pi(u, v).
\]

which proves the lemma. \( \square \)

A.4 Proof of Theorem 4.1

Proof. We first show that by truncating at the \( L = \log_{1 - \theta} \) hop, we only introduce an additive error of \( \theta \). More precisely, note that \( \sum_{i=1}^{L} (1 - \theta)^i \leq (1 - \theta)^{L+1} \leq \theta \). By setting \( \theta \) that is significantly smaller than the relative error threshold \( \delta \) or the additive error bound \( \varepsilon \), we can accumulate the \( \theta \) additive error without increasing the asymptotic query time.

According to Lemma 4.4, we have \( \text{Var} \{ \hat{\pi}_t (s, t) \} \leq \theta \pi_t (s, t) \). By Chebyshev inequality, we have

\[
\Pr \left( |\hat{\pi}_t (s, t) - \pi_t (s, t)| \geq \sqrt{30} \theta \pi_t (s, t) \right) \leq 1/3.
\]

We claim that this variance implies an \( \varepsilon \)-relative error for all \( \pi_t (s, t) \geq 30 \varepsilon^2 \). For a proof, note that \( \theta \leq \varepsilon \pi_t (s, t)/3 \) and consequently \( \sqrt{30} \theta \pi_t (s, t) \leq \varepsilon \pi_t (s, t) \). It follows that \( \Pr \left( |\hat{\pi}_t (s, t) - \pi_t (s, t)| \geq \varepsilon \pi_t (s, t) \right) \leq 1/3 \) for all \( \pi_t (s, t) \geq 30 \varepsilon^2 \).

By setting \( \theta = \frac{\varepsilon^2}{3} \), we obtain a constant relative error guarantee for all \( \pi_t (s, t) \geq \delta \) and consequently a constant relative error for \( \pi_t (s, t) \geq \delta \). To obtain a high probability result, we can apply the Median-of-Mean trick [12], which takes the median of \( O(\log n) \) independent copies of \( \hat{\pi}_t (s, t) \) as the final estimator to \( \pi_t (s, t) \). This trick brought the failure probability from \( 1/3 \) to \( 1/n^2 \) by increasing the running time by a factor of \( O(\log n) \).

Applying the union bound to \( n \) source nodes \( s \in V \) and \( \ell = 0, \ldots, \), the failure probability becomes \( 1/n \). Finally, by setting \( \lambda = 1 \) in Lemma 4.5, we can rewrite the time cost as below.

\[
E[C_{\ell t}] \leq \frac{1}{\alpha \theta} \sum_{u \in V} \pi(u, t) = \frac{1}{\alpha \theta} \sum_{u \in V} \pi(u, t) = \frac{n \pi(t)}{\alpha \theta},
\]

where \( \pi(t) \) represents \( t \)'s PageRank and \( n \pi(t) = \sum_{u \in V} \pi(u, t) \) according to PPR’s definition. By setting \( \theta = \frac{\varepsilon^2}{3} \) and running \( O(\log n) \) independent copies of Algorithm 2, the time complexity can be bounded by \( \tilde{O} \left( \frac{n \pi(t) \log n}{\alpha \varepsilon^2} \right) = \tilde{O} \left( \frac{n \pi(t)}{\varepsilon} \right) \). If we choose the target node \( t \) uniformly at random from set \( V \), then \( E[\pi(t)] = \frac{1}{n} \), and the running time becomes \( \tilde{O} \left( \frac{1}{\varepsilon} \right) \).

A.5 Proof of Theorem 4.2

Proof. Applying Lemma 4.4, we have \( \text{Var} \{ \hat{\pi}_t (s, t) \} \leq \alpha \theta^2 \). Consequently, we have \( \text{Var} \{ \hat{\pi}_t (s, t) \} = \text{Var} \{ \sum_{t=0}^{L} \hat{\pi}_t (s, t) \} \leq \alpha \theta^2 \). By Chebyshev’s inequality, we have \( \Pr \left( |\hat{\pi}_t (s, t) - \pi(s, t)| \geq \sqrt{30} \pi(s, t) \right) \leq 1/3 \). By setting \( \theta = \epsilon / \sqrt{30} \), it follows that \( \hat{\pi}_t (s, t) \) is an \( \epsilon \) additive error for all \( \pi(s, t) \). Similar to the proof of Theorem 4.1, we can use the median of \( O(\log n) \) independent copies of \( \hat{\pi}_t (s, t) \) as the estimator to reduce the failure probability from \( 1/3 \) to \( 1/n \) for all source nodes \( s \in V \).

For the time cost, Lemma 4.5 implies that

\[
E[C_{\ell t}] \leq \frac{1}{\alpha \theta} \sum_{u \in V} \pi(u, t) = \frac{1}{\alpha \theta} \sum_{u \in V} \pi(u, t) = \frac{n \pi(t)}{\alpha \theta},
\]

Recall that we set \( \theta = \epsilon / \sqrt{30} \) and run \( O(\log n) \) independent copies of Algorithm 2, it follows that the running time can be bounded by \( \tilde{O} \left( \frac{1}{\epsilon} \sum_{u \in V} \sqrt{d_{out}(u)} \cdot \pi(u, t) \right) \). If \( t \) is chosen uniformly at random, we have \( \sum_{t \in V} \pi(u, t) = 1 \). Ignoring the \( \tilde{O} \) notation, we have

\[
E[C_{\ell t}] \leq \frac{1}{\epsilon} \sum_{u \in V} \sqrt{d_{out}(u)} \cdot \pi(u, t) = \frac{1}{\epsilon} \sum_{u \in V} \sqrt{d_{out}(u)} \cdot \pi(u, t) = \frac{1}{\epsilon} \sum_{u \in V} \sqrt{d_{out}(u)}.
\]

By the AM-GM inequality, we have \( \frac{1}{\epsilon} \sum_{u \in V} \sqrt{d_{out}(u)} \leq \sqrt{\sum_{u \in V} d_{out}(u)} \leq \sqrt{\lambda} \). Hence, \( E[C_{\ell t}] \leq \frac{1}{\epsilon} \sum_{u \in V} \sqrt{d_{out}(u)} \leq \sqrt{\lambda} \), and the theorem follows. \( \square \)