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
Gaoling School 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.

ACM Reference Format:
Hanzhi Wang, Zhewei Wei, Junhao Gan, Sibo Wang, and Zengfeng Huang. 2020. Personalized PageRank to a Target Node, Revisited. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. 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 α-discounted* random walk from node s terminates at t. Here an α-discounted random walk represents a random traversal that, at each step, either terminates at the current node with probability α, or moves to a random out-neighbor with probability $1 - \alpha$. For a given source node s, the PPR value of each node t sum up to $\sum_{t \in V} \pi(s,t) = 1$, 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 $\sum_{s \in V} \pi(s,t) = n \cdot \pi(t)$, where $\pi(t)$ is the PageRank of t [46]. 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			
		Random target node		Worst case	
	Monte-Carlo [14]	Backward Search [38]	Ours	Backward Search [38]	Ours
Relative error	$\tilde{O}\left(\frac{1}{\delta}\right)$	$O\left(\frac{\bar{d}}{\delta}\right)$	$\tilde{O}\left(\frac{1}{\delta}\right)$	$O\left(\sum_{u \in V} \frac{d_{out}(u) \cdot \pi(u,t)}{\delta}\right)$	$\tilde{O}\left(\frac{n\pi(t)}{\delta}\right)$
Additive error	$\tilde{O}\left(\frac{1}{\varepsilon^2}\right)$	$O\left(\frac{\bar{d}}{\varepsilon}\right)$	$\tilde{O}\left(\frac{\sqrt{\bar{d}}}{\varepsilon}\right)$	$O\left(\sum_{u \in V} \frac{d_{out}(u) \cdot \pi(u,t)}{\varepsilon}\right)$	$\tilde{O}\left(\sum_{u \in V} \frac{\sqrt{d_{out}(u)} \cdot \pi(u,t)}{\varepsilon}\right)$

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 $\pi(s,t) > \phi \cdot n\pi(t)$ with a given node t and a parameter ϕ. 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 [25, 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 α-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 α-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 $\pi(s,t) \geq \delta$ 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 $\pi(s,t) \geq \delta$, which implies that there is a lower bound of $\Omega(1/\delta)$ 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 $\pi(s,t) \geq \delta$ in $\tilde{O}(\bar{d}/\delta)$ time, where $\bar{d}$ is the average degree of the graph. Thus, there is an $O(\bar{d})$ gap between the upper bound and lower bound for the single-target PPR problem. For dense graphs such as complete graphs, the $O(\bar{d})$ 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 defintion 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 ε, 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

$$|\hat{\pi}(s,t) - \pi(s,t)| \le \varepsilon \tag{1}$$

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 δ, 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 $\pi(s,t) > \delta$,

$$|\hat{\pi}(s,t) - \pi(s,t)| \le \frac{1}{10} \cdot \pi(s,t) \tag{2}$$

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 $\pi(s,t) \ge \delta$ 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 $\pi(s,t) \ge \delta$.
- Given a random target node t, RBS answers a single-target PPR query with an additive error ε with constant probability using $\tilde{O}\left(\frac{\sqrt{\bar{d}}}{\varepsilon}\right)$ time. This query time complexity improves previous bound for single-targe PPR query with additive error by a factor of $\sqrt{\bar{d}}$. 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 α-discounted random walk terminates at the current node with α 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.

$$\vec{\pi_s} = (1-\alpha)\vec{\pi_s} \cdot \mathbf{P} + \alpha \cdot \vec{e_s}, \tag{3}$$

where $\vec{\pi_s}$ denotes the PPR vector with respect to a given source node s, $\vec{e_s}$ denotes the one-hot vector with $\vec{e_s}(s) = 1$, and $\mathbf{P}$ 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.
$\pi_\ell(s,t)$, $\hat{\pi}_\ell(s,t)$	the true and estimated ℓ-hop PPR values of node s to t.
α	the teleport probability that a random walk terminates at each step
ε	the additive error parameter
δ	the relative error threshold
$\bar{d}$	the average degree, $\bar{d} = \frac{m}{n}$
$\tilde{O}$	the Big-Oh notation ignoring the log factors

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

Reversing this process, we can also compute single-target PPR values with the given target node t. The iteration formula should be adjusted correspondingly:

$$\vec{\pi_t} = (1-\alpha)\vec{\pi_t} \cdot \mathbf{P}^\top + \alpha \cdot \vec{e_t}. \tag{5}$$

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 ε 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 residue $r^b(s,t)$ to denote the probability mass to be distributed at node s, and reserve $\pi^b(s,t)$ denotes the probability mass that will stay at s permanently. For initialization, Backward Search sets $r^b(u,t) = \pi^b(u,t) = 0$ for $\forall u \in V$, except for the residue $r^b(t,t) = 1$. In each push operation, it picks the node v with the largest residue $r^b(v,t)$, and transfer a fraction of α to $\pi^b(v,t)$, the reserve of v. Then the algorithm transfers the other $(1-\alpha)$ faction to the in-neighbors of v. For each in-neighbor u of v, the residue $r^b(u,t), u \in N_{in}(v)$ is incremented by $\frac{(1-\alpha)r^b(v,t)}{d_{out}(u)}$. 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 ε. Finally, Backward Search uses the reserve $\pi^b(s,t)$ 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 $\forall s,t \in V$, $\pi(s,t)$ satisfies that*

$$\pi(s,t) = \sum_{u \in N_{in}(t)} \frac{1-\alpha}{d_{out}(u)} \cdot \pi(s,u) + \alpha \cdot I\{s = t\}. \tag{6}$$

Algorithm 1: Backward Search [38]

```
    Input: Graph $G = (V, E)$, target node $t$, teleport probability $\alpha$,
           additive error parameter $\varepsilon$
    Output: Reserve $\pi^b(s,t)$ for all $s \in V$
1   for each $u \in V$ do
2       $r^b(u,t), \pi^b(u,t) \leftarrow 0$;
3   $r^b(t,t) \leftarrow 1$;
4   while The largest $r^b(v,t) > \varepsilon$ do
5       $\pi^b(v,t) \leftarrow \pi^b(v,t) + \alpha \cdot r^b(v,t)$;
6       for each $u \in \mathcal{N}^{in}(v)$ do
7           $r^b(u,t) \leftarrow r^b(u,t) + (1-\alpha) \cdot \frac{r^b(v,t)}{d_{out}(u)}$
8       $r^b(v,t) \leftarrow 0$;
9   return $\pi^b(s,t)$ as the estimator for $\pi(s,t), s \in V$;
```

Utilizing this property, it is shown in [38] that the residues and reserves of Backward Search satisfies the following invariant:

$$\pi(s,t) = \pi^b(s,t) + \sum_{u \in V} r^b(u,t) \cdot \pi(s,u). \tag{7}$$

Note that when the Backward Search algorithm terminates, all residues $r^b(u,t) \le \varepsilon$. It follows that $\pi^b(s,t) \le \pi(s,t) \le \pi^b(s,t) + \varepsilon \sum_{u \in V} \pi(s,u) = \pi^b(s,t) + \varepsilon$, where $\sum_{u \in V} \pi(s,u) = 1$. Therefore, Backward Search ensures an additive error of ε. It is shown in [38] that the running time of Backward Search is bounded by $O\left(\sum_{u \in V} \frac{d_{in}(u) \cdot \pi(u,t)}{\varepsilon}\right)$. We claim that the running time of Backward Search can also be bounded by $O\left(\sum_{u \in V} \frac{d_{out}(u) \cdot \pi(u,t)}{\varepsilon}\right)$. 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 $O(\frac{\bar{d}}{\varepsilon})$, where $\bar{d}$ is the average degree of the graph. For relative error, we can set $\delta = \Theta(\varepsilon)$ and obtain a worst-case complexity of $O\left(\sum_{u \in V} \frac{d_{out}(u) \cdot \pi(u,t)}{\delta}\right)$ and an average complexity of $O(\frac{\bar{d}}{\delta})$, 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 ε is $\tilde{O}(\frac{1}{\varepsilon^2})$, while the number of random walks required to ensure constant relative error for all PPR larger than δ is $\tilde{O}(\frac{1}{\delta})$. This simple method is optimal for single-source PPR queries with relative error, as there are at most $O(\frac{1}{\delta})$ nodes t with PPR $\pi(s,t) \ge \delta$. 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 $O(\frac{1}{\delta})$ 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 $\pi^f(s,u)$ 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 $\forall u \in V$, except for the residue $r^f(s,s) = 1$. In each push operation, it picks the node u with the largest residue/degree ratio $r^f(s,u)/d_{out}(u)$, and transfer a fraction of α to $\pi^f(s,u)$, the reserve of u. Then the algorithm transfers the other $(1-\alpha)$ faction to the out-neighbors of u. For each out-neighbor v of u, the residue $r^f(s,v)$ is incremented by $\frac{(1-\alpha)r^f(s,u)}{d_{out}(u)}$. After all in-neighbors are processed, the algorithm sets the residue of u to be 0. The process ends when the maximum residue/degree ratio descends below the error parameter ε. 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 $O(1/\varepsilon)$ time. However, the major problem with Forward Search is that it can only ensure an additive error of $\varepsilon d_{out}(u)$ for each PPR value $\pi(s,u)$ on undirected graphs. Compared to the ε 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, 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 *ℓ-hop Personalized PageRank $\pi_\ell(s,t)$*, which is defined as the probability that an α-discounted random walk from s terminates at t at exactly ℓ steps. For different ℓ, 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 θ to the final estimator and $\theta = \tilde{O}(\varepsilon)$. 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 $(1-\alpha)$ faction 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 v and thus leads to the $\bar{d}$ overhead. We avoid this complexity by pushing the probability mass to a small random subset of v's in-neighbors. The probability for including an in-neighbor u depends

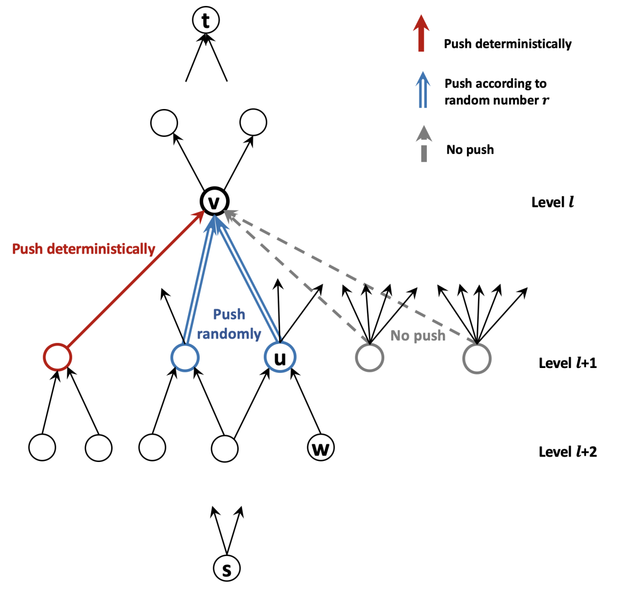

Figure 1: Sketch of Randomized Backward Search

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 $d_{out}(u_1) \leq \ldots, \leq d_{out}(u_d)$. 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 $(u, v, d_{out}(u))$ for each edge $(u, v) \in E$, and use counting sort to sort $(u, v, d_{out}(u))$ tuples in the ascending order of $d_{out}(u)$. Since each $d_{out}(u)$ is bounded by n, and there are m tuples, the cost of counting sort is bounded by $O(m + n)$. Finally, for each $u, v, d_{out}(u)$, 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 α. The algorithm takes in two additional parameters: error parameter θ and sampling function $\lambda(u)$. We can manipulate these two parameters to obtain the additive or relative error guarantees. Recall that $\tilde{O}$ denotes the Big-Oh notation ignoring the log factors. For single-target PPR query with additive error, we set θ to be $\tilde{O}(\varepsilon)$, the maximum additive error allowed, and $\lambda(u) = \sqrt{d_{out}(u)}$. For relative error, we set $\theta = \tilde{O}(\delta)$, 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). We then initialize the estimators $\hat{\pi}_\ell(s, t) = 0$ for $\forall s \in V$ and $\forall \ell \in [0, L]$ 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

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  $\hat{\pi}_\ell(s, t) \leftarrow 0$ for $\ell = 0, \ldots, L, s \in V$;
3  $\hat{\pi}_0(t, t) \leftarrow \alpha$;
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 $d_{out}(u) \leq \frac{\lambda(u)\cdot(1-\alpha)\hat{\pi}_\ell(v,t)}{\alpha\theta}$ do
7              $\hat{\pi}_{\ell+1}(u, t) \leftarrow \hat{\pi}_{\ell+1}(u, t) + \frac{(1-\alpha)\hat{\pi}_\ell(v,t)}{d_{out}(u)}$;
8          $r \leftarrow rand(0, 1)$;
9          for each $u \in N_{in}(v)$ and
           $\frac{\lambda(u)\cdot(1-\alpha)\hat{\pi}_\ell(v,t)}{\alpha\theta} < d_{out}(u) \leq \frac{\lambda(u)\cdot(1-\alpha)\hat{\pi}_\ell(v,t)}{r\alpha\theta}$ do
10             $\hat{\pi}_{\ell+1}(u, t) \leftarrow \hat{\pi}_{\ell+1}(u, t) + \frac{\alpha\theta}{\lambda(u)}$;
11 return all non-zero $\hat{\pi}(s, t) = \sum_{\ell=0}^{L} \hat{\pi}_\ell(s, t)$ for each $s \in V$;
```

ℓ, 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 $d_{out}(u) \leq \frac{\lambda(u)\cdot(1-\alpha)\hat{\pi}_\ell(v,t)}{\alpha\theta}$, we deterministically push a probability mass of $\frac{(1-\alpha)\hat{\pi}_\ell(v,t)}{d_{out}(u)}$ 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 $d_{out}(u) > \frac{\lambda(u)\cdot(1-\alpha)\hat{\pi}_\ell(v,t)}{\alpha\theta}$. For in-neighbors with higher out-degrees, we generate a random number r from $(0, 1)$, and push a probability mass of $\frac{\alpha\theta}{\lambda(u)}$ to $\hat{\pi}_{\ell+1}(u, t)$ for each in-neighbor u with $d_{out}(u) \leq \frac{\lambda(u)\cdot(1-\alpha)\hat{\pi}_\ell(v,t)}{r\alpha\theta}$ (lines 8 -10). Similarly, we can sequentially scan the in-adjacency list of v until we encounter the first in-neighbor u with $d_{out}(u) > \frac{\lambda(u)\cdot(1-\alpha)\hat{\pi}_\ell(v,t)}{r\alpha\theta}$. Finally, after all L hops are processed, we return $\hat{\pi}(s, t) = \sum_{\ell=0}^{L} \hat{\pi}_\ell(s, t)$ as the estimator for each $\pi(s, t), s \in V$. The sketch map of the above process is shown in Figure 1.

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 $\sqrt{d_{out}(u)}$ and $\theta = \tilde{O}(\varepsilon)$ to be the error bound. For relative error, we set $\lambda(u) = 1$ and $\theta = \tilde{O}(\delta)$, the threshold for constant relative error guarantee. Recall that $\tilde{O}$ 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 $\theta = \tilde{O}(\delta)$, Algorithm 2 answers the single-target PPR queries with a relative error threshold δ with high probability. The expected worst-case time cost is bounded by $\tilde{O}\left(\frac{n\pi(t)}{\delta}\right)$. If the target node t is chosen uniformly at random from V, the time cost becomes $\tilde{O}\left(\frac{1}{\delta}\right)$.*

Theorem 4.2. *By setting $\lambda(u) = \sqrt{d_{out}(u)}$ and $\theta = \tilde{O}(\varepsilon)$, Algorithm 2 answers the single-target PPR queries with an additive error parameter ε with high probability. The expected worst-case time cost is bounded by*

$$\mathrm{E}[Cost] = \tilde{O}\left(\frac{1}{\varepsilon}\sum_{u\in V}\sqrt{d_{out}(u)}\cdot\pi(u,t)\right).$$

If the target node t is chosen uniformly at random from V, the time cost becomes $\tilde{O}\left(\frac{\sqrt{\bar{d}}}{\varepsilon}\right)$, where $\bar{d}$ is the average degree of the graph.

To prove Theorem 4.1 and 4.2, we need several technical lemmas. In particular, we first prove that Algorithm 2 provides an unbiased estimator for the ℓ-hop PPR values $\pi_\ell(s,t)$.

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

Next, we bound the variance of the ℓ-hop estimators.

Lemma 4.4. *For any $s \in V$ and $\ell \in \{0,1,2,...,L\}$, the variance of each estimator $\hat{\pi}_\ell(s,t)$ obtained by Algorithm 2 satisfies that:*
1) If we set $\lambda(u) = 1$, then $\mathrm{Var}\left[\hat{\pi}_\ell(s,t)\right] \le \theta\pi_\ell(s,t)$.
2) If we set $\lambda(u) = \sqrt{d_{out}(u)}$, then $\mathrm{Var}\left[\hat{\pi}_\ell(s,t)\right] \le \alpha\theta^2$.

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*

$$\mathrm{E}\left[C_{total}\right] \le \frac{1}{\alpha\theta}\sum_{u\in V}\lambda(u)\cdot\pi(u,t).$$

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

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

On the other hand, by running the RBS algorithm with relative error threshold $\delta = c\phi n\pi(t)$, we can return the c-approximate heavy hitter for node t with high probability. By Theorem 4.1, the running time of RBS algorithm is bounded by $\tilde{O}\left(\frac{n\pi(t)}{\delta}\right) = \tilde{O}\left(\frac{1}{\phi}\right)$. Note that this complexity is optimal up to log factors, as there may be $O\left(\frac{1}{\phi}\right)$ 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 α-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 ℓ-hop PPR: $s(u,v) = \frac{1}{(1-\sqrt{c})^2}\sum_{\ell=0}^{\infty}\sum_{w\in V}\pi_\ell(u,w)\pi_\ell(v,w)\eta(w)$, where $\pi_\ell(u,w)$ is the ℓ-hop PPR with decay factor $\alpha = 1-\sqrt{c}$, and $\eta(w)$ is a value called the last meeting probability. SLING [53] proposes to use Backward Search to precompute an approximation of $\pi_\ell(v,w)$ with additive error ε for each $w,v \in V, \ell = 0,\ldots,\infty$, while PRSim [60] only precomputes an approximation of $\pi_\ell(v,w)$ for node w with large PageRanks. Recall that the running time of the Backward Search algorithm on a random node t is $O\left(\frac{\bar{d}}{\varepsilon}\right)$. It follows that the total precomputation cost for SLING or PRSim is bounded by $O\left(\frac{m}{\varepsilon}\right)$. 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 $\tilde{O}\left(\frac{\sqrt{\bar{d}}}{\varepsilon}\right)$. And thus the total precomputation time is improved to $\tilde{O}\left(\frac{n\sqrt{\bar{d}}}{\varepsilon}\right)$.

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 $\tilde{O}(mn)$ 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 $\varepsilon d_{out}(t)$ 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 $O\left(\frac{m}{\varepsilon}\right)$ for computing an approximate PPR matrix with additive error ε. By replacing the Backward Search algorithm with the new RBS algorithm with additive error, we can improve the complexity to $\tilde{O}\left(\frac{n\sqrt{\bar{d}}}{\varepsilon}\right)$, 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

Table 3: Data Sets.

Data Set	Type	n	m
ca-GrQc (GQ)	undirected	5,242	28,968
AS-2000(AS)	undirected	6,474	25,144
DBLP-Author (DB)	undirected	5,425,963	17,298,032
IndoChina (IC)	directed	7,414,768	191,606,827
Orkut-Links (OL)	undirected	3,072,441	234,369,798
It-2004 (IT)	directed	41,290,682	1,135,718,909
Twitter (TW)	directed	41,652,230	1,468,364,884

from [2, 3]. All experiments are conducted on a machine with an Intel(R) Xeon(R) E7-4809 @2.10GHz CPU and 196GB memory.

6.1 Single-Target Query

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

$$MaxAdditiveErr = \max_{v_i \in V} |\pi(v_i, t) - \hat{\pi}(v_i, t)| \tag{8}$$

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 δ. Hence, we consider the *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. *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 *MaxAdditiveErr* and *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 $\lambda(u) = \sqrt{d_{out}(u)}$ and $\theta = \varepsilon$ for additive error, $\lambda(u) = 1$ and $\theta = \delta$ for relative error. For Backward Search (BS), we set $\varepsilon = \delta$ for relative error. We vary the additive error parameter ε and relative threshold δ from 0.1 to 10^{-6} in both experiments. Following previous work [56], we set the decay factor α to be 0.2.

Figure 2 shows the tradeoffs between the *MaxAdditiveErr* and the query time for the additive error experiments. Figure 3 presents the tradeoffs between *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 ϕ is a parameter. Following [58], we fix $\phi = 10^{-5}$ in our experiments. Since the number of true heavy hitters is oblivious to the algorithms, we use the *F1 score* instead of precision to evaluate the approximate algorithms, where the *F1 score* is defined as $F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$. We set $\lambda(u) = 1$ and $\theta = \delta$ for RBS, and $\varepsilon = \delta$ for BS, and set δ 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: $s(u, v) = \frac{1}{(1-\sqrt{c})^2} \sum_{\ell=0}^{\infty} \sum_{w \in V} \pi_\ell(u, w) \pi_\ell(v, w) \eta(w)$. The state-of-the-art single-source SimRank methods, SLING [53] and PRSim [60], pre-compute the ℓ-hop PPR values $\pi_\ell(v, w)$ 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 *MaxAdditiveErr@50*, the maximum additive error of single-source top-50 SimRank values for a given query node. PRSim has one error parameter ε. 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 *MaxAdditiveErr@50* for each approximate method. Figure 5 illustrates the tradeoffs between the preprocessing time with *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 ℓ-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 FS and BS 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 $\lambda(u) = \sqrt{d_{out}(u)}$ and $\theta = \varepsilon$ for RBS, and vary the additive error parameter ε in RBS from 0.1 to 10^{-6}. Similarly, we vary the parameter ε of FS and BS from 0.1 to 10^{-6}. Figure 6 shows the tradeoffs between *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 only guarantees an additive error of $\varepsilon d_{out}(t)$ while BS guarantees an additive error of ε. 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.

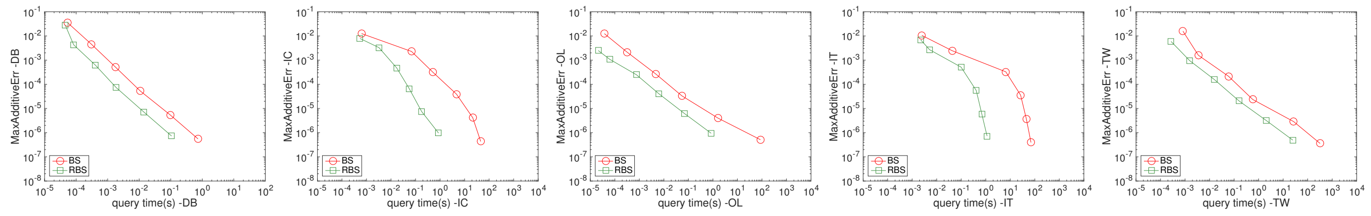

Figure 2: Tradeoffs between *MaxAdditiveErr* and query time.

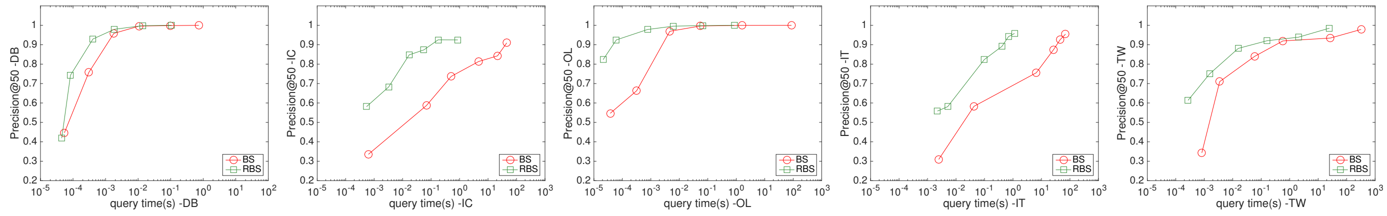

Figure 3: Tradeoffs between *Precision@50* and query time.

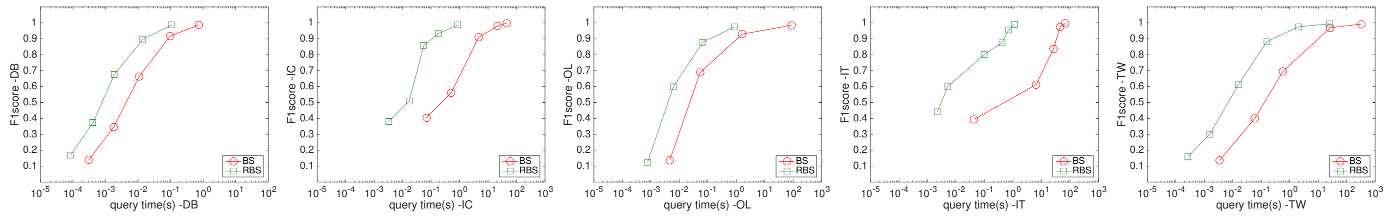

Figure 4: Heavy hitters: Tradeoffs between *F1 score* and query time.

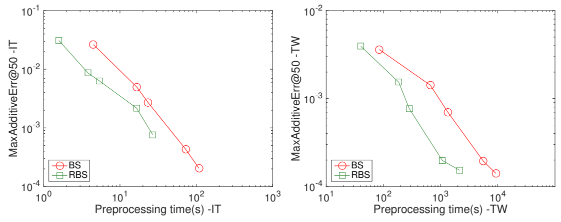

Figure 5: Single-source SimRank: Tradeoffs between *MaxAdditiveErr@50* and preprocessing time.

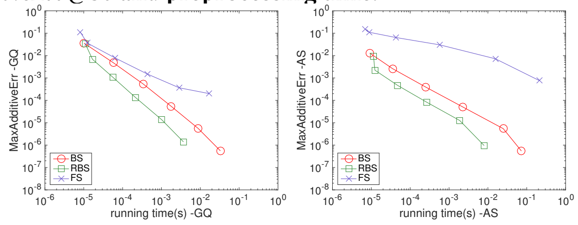

Figure 6: Approximate PPR matrix: tradeoffs between *MaxAdditiveErr* and running time.

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 [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. BJJWZYJH012019100020098, and by the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China under Grant 18XNLG21. Junhao Gan is supported by Australian Research Council (ARC) DECRA DE190101118. 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

[1] http://arxiv.org/abs/2006.11876.
[2] http://snap.stanford.edu/data.
[3] http://law.di.unimi.it/datasets.php.
[4] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcroft, Kamal Jain, Vahab Mirrokni, and Shanghua Teng. Robust pagerank and locally computable spam detection features. In *Proceedings of the 4th international workshop on Adversarial information retrieval on the web*, pages 69–76, 2008.
[5] Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. Local computation of pagerank contributions. In *WAW*, pages 150–165, 2007.
[6] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Local graph partitioning using pagerank vectors. In *FOCS*, pages 475–486, 2006.
[7] Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, pages 635–644, 2011.
[8] Bahman Bahmani, Kaushik Chakrabarti, and Dong Xin. Fast personalized pagerank on mapreduce. In *SIGMOD*, pages 973–984, 2011.
[9] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *VLDB*, 4(3):173–184, 2010.

[10] Marco Bressan, Enoch Peserico, and Luca Pretto. Sublinear algorithms for local graph centrality estimation. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 709–718. IEEE, 2018.

[11] Soumen Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *WWW*, pages 571–580, 2007.

[12] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *ICALP*, pages 693–703. Springer, 2002.

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

[14] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2(3):333–358, 2005.

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

[16] Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Takeshi Mishima, and Makoto Onizuka. Efficient ad-hoc search for personalized pagerank. In *SIGMOD*, pages 445–456, 2013.

[17] Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Takeshi Mishima, and Makoto Onizuka. Fast and exact top-k algorithm for pagerank. In *AAAI*, 2013.

[18] Yasuhiro Fujiwara, Makoto Nakatsuji, Takeshi Yamamuro, Hiroaki Shiokawa, and Makoto Onizuka. Efficient personalized pagerank with accuracy assurance. In *KDD*, pages 15–23, 2012.

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

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

[21] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. Wtf: The who to follow service at twitter. In *WWW*, pages 505–514, 2013.

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

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

[24] Minhao Jiang, Ada Wai-Chee Fu, and Raymond Chi-Wing Wong. Reads: a random walk approach for efficient and accurate dynamic simrank. *PPVLDB*, 10(9):937–948, 2017.

[25] Ruoming Jin, Victor E Lee, and Hui Hong. Axiomatic ranking of network role similarity. In *KDD*, pages 922–930, 2011.

[26] Jinhong Jung, Namyong Park, Sael Lee, and U Kang. Bepi: Fast and memory-efficient method for billion-scale random walk with restart. In *SIGMOD*, pages 789–804, 2017.

[27] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.

[28] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Personalized embedding propagation: Combining neural networks on graphs with personalized pagerank. *CoRR*, abs/1810.05997, 2018.

[29] Johannes Klicpera, Stefan WeiÃ§enberger, and Stephan GÃijnnemann. Diffusion improves graph learning, 2019.

[30] Mitsuru Kusumoto, Takanori Maehara, and Ken-ichi Kawarabayashi. Scalable similarity search for simrank. In *SIGMOD*, pages 325–336, 2014.

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

[32] Lina Li, Cuiping 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.

[33] Zhenguo Li, Yixiang Fang, Qin Liu, Jiefeng Cheng, Reynold Cheng, and John Lui. Walking in the cloud: Parallel simrank at scale. *PVLDB*, 9(1):24–35, 2015.

[34] David Liben-Nowell and Jon M. Kleinberg. The link prediction problem for social networks. In *CIKM*, pages 556–559, 2003.

[35] Yu Liu, Bolong Zheng, Xiaodong He, Zhewei Wei, Xiaokui Xiao, Kai Zheng, and Jiaheng Lu. Probesim: scalable single-source and top-k simrank computations on dynamic graphs. *PVLDB*, 11(1):14–26, 2017.

[36] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. Bidirectional pagerank estimation: From average-case to worst-case. In *WAW*, pages 164–176, 2015.

[37] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. Personalized pagerank estimation and search: A bidirectional approach. In *WSDM*, pages 163–172, 2016.

[38] Peter Lofgren and Ashish Goel. Personalized pagerank to a target node. *arXiv preprint arXiv:1304.4658*, 2013.

[39] Peter A. Lofgren, Siddhartha Banerjee, Ashish Goel, and C. Seshadhri. Fast-ppr: Scaling personalized pagerank estimation for large graphs. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1436–1445, New York, NY, USA, 2014. ACM.

[40] Peter A Lofgren, Siddhartha Banerjee, Ashish Goel, and C Seshadhri. Fast-ppr: Scaling personalized pagerank estimation for large graphs. In *KDD*, pages 1436–1445, 2014.

[41] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390(6):1150–1170, 2011.

[42] Takanori Maehara, Takuya Akiba, Yoichi Iwata, and Ken-ichi Kawarabayashi. Computing personalized pagerank quickly by exploiting graph structures. *PVLDB*, 7(12):1023–1034, 2014.

[43] Takanori Maehara, Mitsuru Kusumoto, and Ken-ichi Kawarabayashi. Efficient simrank computation via linearization. *CoRR*, abs/1411.7228, 2014.

[44] Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. Efficient pagerank tracking in evolving networks. In *KDD*, pages 875–884, 2015.

[45] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *SIGKDD*, pages 1105–1114. ACM, 2016.

[46] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[47] Hannu Reittu, Ilkka Norros, Tomi Räty, Marianna Bolla, and Fülöp Bazsó. Regular decomposition of large graphs: Foundation of a sampling approach to stochastic block model fitting. *Data Science and Engineering*, 4(1):44–60, 2019.

[48] CH Ren, Luyi Mo, CM Kao, CK Cheng, and DWL Cheung. Clude: An efficient algorithm for lu decomposition over a sequence of evolving graphs. In *EDBT*, 2014.

[49] Atish Das Sarma, Anisur Rahaman Molla, Gopal Pandurangan, and Eli Upfal. Fast distributed pagerank computation. *Theoretical Computer Science*, 561:113–121, 2015.

[50] Yingxia Shao, Bin Cui, Lei Chen, Mingming Liu, and Xing Xie. An efficient similarity search framework for simrank over large dynamic graphs. *PVLDB*, 8(8):838–849, 2015.

[51] Kijung Shin, Jinhong Jung, Lee Sael, and U. Kang. BEAR: block elimination approach for random walk with restart on large graphs. In *SIGMOD*, pages 1571–1585, 2015.

[52] Nikita Spirin and Jiawei Han. Survey on web spam detection: principles and algorithms. *SIGKDD Explorations*, 13(2):50–64, 2011.

[53] Boyu Tian and Xiaokui Xiao. SLING: A near-optimal index structure for simrank. In *SIGMOD*, pages 1859–1874, 2016.

[54] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *WWW*, pages 539–548. International World Wide Web Conferences Steering Committee, 2018.

[55] Petar VeliÄŋkoviÄĞ, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro LiÃš, and Yoshua Bengio. Graph attention networks, 2017.

[56] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. Hubppr: Effective indexing for approximate personalized pagerank. *PVLDB*, 10(3):205–216, 2016.

[57] Sibo Wang, Youze Tang, Xiaokui Xiao, Yang Yin, and Zengxiang Li. Hubppr: Effective indexing for approximate personalized pagerank. In *PVLDB*, 2016.

[58] Sibo Wang and Yufei Tao. Efficient algorithms for finding approximate heavy hitters in personalized pageranks. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1113–1127, 2018.

[59] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. FORA: simple and effective approximate single-source personalized pagerank. In *KDD*, pages 505–514, 2017.

[60] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Yu Liu, Xiaoyong Du, and Ji-Rong Wen. Prsim: Sublinear time simrank computation on large power-law graphs. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1042–1059, 2019.

[61] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. Topppr: top-k personalized pagerank queries with precision guarantees on large graphs. In *SIGMOD*, pages 441–456. ACM, 2018.

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

[63] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *CoRR*, abs/1806.03536, 2018.

[64] Yuan Yin and Zhewei Wei. Scalable graph embeddings via sparse transpose proximities. *CoRR*, abs/1905.07245, 2019.

[65] Weiren Yu and Xuemin Lin. IRWR: incremental random walk with restart. In *SIGIR*, pages 1017–1020, 2013.

[66] Weiren Yu and Julie A. McCann. Efficient partial-pairs simrank search on large networks. *Proceedings of the Vldb Endowment*, 8(5):569–580.

[67] Weiren Yu and Julie A. McCann. Random walk with restart over dynamic graphs. In *ICDM*, pages 589–598, 2016.

[68] Hongyang Zhang, Peter Lofgren, and Ashish Goel. Approximate personalized pagerank on dynamic graphs. In *KDD*, pages 1315–1324, 2016.

[69] Fanwei Zhu, Yuan Fang, Kevin Chen-Chuan Chang, and Jing Ying. Incremental and accuracy-aware personalized pagerank through scheduled approximation. *PVLDB*, 6(6):481–492, 2013.

A APPENDIX

A.1 Proof of Lemma 4.3

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

$$\begin{aligned} \mathrm{E}\left[X_{\ell+1}(u,v) \mid \{\hat{\pi}_\ell\}\right] &= \begin{cases} \frac{(1-\alpha)\hat{\pi}_\ell(v,t)}{d_{out}(u)}, & if\, \frac{(1-\alpha)\hat{\pi}_\ell(v,t)}{d_{out}(u)} \geq \frac{\alpha\theta}{\lambda(u)} \\ \frac{\alpha\theta}{\lambda(u)} \cdot \frac{\lambda(u)(1-\alpha)\hat{\pi}_\ell(v,t)}{\alpha\theta\cdot d_{out}(u)}, & otherwise \end{cases} \\ &= \frac{(1-\alpha)\hat{\pi}_\ell(v,t)}{d_{out}(u)}. \end{aligned} \tag{9}$$

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

$$\begin{aligned} \mathrm{E}\left[\hat{\pi}_{\ell+1}(u,t) \mid \{\hat{\pi}_\ell\}\right] &= \mathrm{E}\left[\sum_{v\in N_{out}(u)} X_{\ell+1}(u,v) \mid \{\hat{\pi}_\ell\}\right] \\ &= \sum_{v\in N_{out}(u)} \mathrm{E}\left[X_{\ell+1}(u,v) \mid \{\hat{\pi}_\ell\}\right]. \end{aligned}$$

Based on equation (9), we have

$$\mathrm{E}\left[\hat{\pi}_{\ell+1}(u,t) \mid \{\hat{\pi}_\ell\}\right] = \sum_{v\in N_{out}(u)} \left(\frac{(1-\alpha)\hat{\pi}_\ell(v,t)}{d_{out}(u)}\right). \tag{10}$$

Because $\mathrm{E}\left[\hat{\pi}_{\ell+1}(u, t)\right] = \mathrm{E}\left[\mathrm{E}\left[\hat{\pi}_{\ell+1}(u, t) \mid \{\hat{\pi}_\ell\}\right]\right]$, it follows that

$$\mathrm{E}\left[\hat{\pi}_{\ell+1}(u,t)\right] = \sum_{v\in N_{out}(u)} \left(\frac{(1-\alpha)\mathrm{E}\left[\hat{\pi}_\ell(v,t)\right]}{d_{out}(u)}\right).$$

$\mathrm{E}\left[\hat{\pi}_i(x, t)\right] = \pi_i(x, t)$ holds for $\forall x \in V$ and $i = 0$ in the initial state, because $\mathrm{E}\left[\hat{\pi}_0(t, t)\right] = \pi_0(t, t) = \alpha$ and $\mathrm{E}\left[\hat{\pi}_0(u, t)\right] = \pi_0(u, t) = 0$ $(u \neq t)$. Assume $\mathrm{E}\left[\hat{\pi}_i(x, t)\right] = \pi_i(x, t)$ holds for $\forall x \in V$ and $i \leq \ell$. We can derive that

$$\begin{aligned} \mathrm{E}\left[\hat{\pi}_{\ell+1}(u,t)\right] &= \sum_{v\in N_{out}(u)} \left(\frac{(1-\alpha)\mathrm{E}\left[\hat{\pi}_\ell(v,t)\right]}{d_{out}(u)}\right) \\ &= \sum_{v\in N_{out}(u)} \left(\frac{(1-\alpha)\pi_\ell(v,t)}{d_{out}(u)}\right) = \pi_{\ell+1}(u,t), \end{aligned}$$

which testifies the unbiasedness. □

A.2 Proof of Lemma 4.4

Proof. During each push operation, the randomness comes from the second scenario that $\frac{(1-\alpha)\hat{\pi}_\ell(v,t)}{d_{out}(u)} < \frac{\alpha\theta}{\lambda(u)}$. Focus on this situation,

$$\begin{aligned} &\mathrm{Var}\left[X_{\ell+1}(u,v) \mid \{\hat{\pi}_\ell\}\right] \leq \mathrm{E}\left[X^2_{\ell+1}(u,v) \mid \{\hat{\pi}_\ell\}\right] \\ &= \left(\frac{\alpha\theta}{\lambda(u)}\right)^2 \cdot \frac{\lambda(u)\cdot(1-\alpha)\hat{\pi}_\ell(v,t)}{\alpha\theta\cdot d_{out}(u)} = \frac{\alpha\theta}{\lambda(u)} \cdot \frac{(1-\alpha)\hat{\pi}_\ell(v,t)}{d_{out}(u)}. \end{aligned}$$

Note that for each $v \in N_{out}(u)$, $X_{\ell+1}(u, v)$ is independent with each other because of the independent generation for the random number r in Algorithm 2. Applying $\hat{\pi}_{\ell+1}(u, t) = \sum_{v\in N_{out}(u)} X_{\ell+1}(u, v)$, the conditional variance of $\hat{\pi}_{\ell+1}(u, t)$ is followed that

$$\begin{aligned} \mathrm{Var}\left[\hat{\pi}_{\ell+1}(u,t) \mid \{\hat{\pi}_\ell\}\right] &= \sum_{v\in N_{out}(u)} \mathrm{Var}[X_{\ell+1}(u,v) \mid \{\hat{\pi}_\ell\}] \\ &\leq \frac{\alpha\theta}{\lambda(u)} \cdot \sum_{v\in N_{out}(u)} \frac{(1-\alpha)\hat{\pi}_\ell(v,t)}{d_{out}(u)}. \end{aligned} \tag{11}$$

By the total variance law, $\mathrm{Var}\left[\hat{\pi}_{\ell+1}(u, t)\right] = \mathrm{E}\left[\mathrm{Var}\left[\hat{\pi}_{\ell+1}(u, t) \mid \{\hat{\pi}_\ell\}\right]\right] + \mathrm{Var}\left[\mathrm{E}\left[\hat{\pi}_{\ell+1}(u, t) \mid \{\hat{\pi}_\ell\}\right]\right]$. Based on equation (11) and the unbiasedness of $\hat{\pi}_\ell(v, t)$ proven in Lemma 4.3, we have

$$\begin{aligned} &\mathrm{E}\left[\mathrm{Var}\left[\hat{\pi}_{\ell+1}(u,t) \mid \{\hat{\pi}_\ell\}\right]\right] \leq \frac{\alpha\theta}{\lambda(u)} \cdot \sum_{v\in N_{out}(u)} \frac{(1-\alpha)\mathrm{E}\left[\hat{\pi}_\ell(v,t)\right]}{d_{out}(u)} \\ &= \frac{\alpha\theta}{\lambda(u)} \cdot \sum_{v\in N_{out}(u)} \frac{(1-\alpha)\pi_\ell(v,t)}{d_{out}(u)} = \frac{\alpha\theta}{\lambda(u)} \cdot \pi_{\ell+1}(u,t). \end{aligned} \tag{12}$$

Meanwhile, applying equation (10), we can derive

$$\begin{aligned} \mathrm{Var}\left[\mathrm{E}\left[\hat{\pi}_{\ell+1}(u,t) \mid \{\hat{\pi}_\ell\}\right]\right] &= \mathrm{Var}\left[\sum_{v\in N_{out}(u)} \left(\frac{(1-\alpha)\hat{\pi}_\ell(v,t)}{d_{out}(u)}\right)\right] \\ &= \frac{(1-\alpha)^2}{d^2_{out}(u)} \cdot \mathrm{Var}\left[\sum_{v\in N_{out}(u)} \hat{\pi}_\ell(v,t)\right]. \end{aligned}$$

The convexity of variance implies that:

$$\mathrm{Var}\left[\sum_{v\in N_{out}(u)} \hat{\pi}_\ell(v,t)\right] \leq d_{out}(u) \cdot \sum_{v\in N_{out}(u)} \mathrm{Var}\left[\hat{\pi}_\ell(v,t)\right].$$

Therefore, we can rewrite $\mathrm{Var}\left[\mathrm{E}\left[\hat{\pi}_{\ell+1}(u, t) \mid \{\hat{\pi}_\ell\}\right]\right]$ as below:

$$\mathrm{Var}\left[\mathrm{E}\left[\hat{\pi}_{\ell+1}(u,t) \mid \{\hat{\pi}_\ell\}\right]\right] \leq \frac{(1-\alpha)^2}{d_{out}(u)} \cdot \sum_{v\in N_{out}(u)} \mathrm{Var}\left[\hat{\pi}_\ell(v,t)\right]. \tag{13}$$

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

$$\mathrm{Var}\left[\hat{\pi}_{\ell+1}(u,t)\right] \leq \frac{\alpha\theta}{\lambda(u)} \cdot \pi_{\ell+1}(u,t) + \frac{(1-\alpha)^2}{d_{out}(u)} \cdot \sum_{v\in N_{out}(u)} \mathrm{Var}\left[\hat{\pi}_\ell(v,t)\right].$$

$\mathrm{Var}[\hat{\pi}_i(x, t)] \leq \frac{\theta}{\lambda(u)} \cdot \pi_i(x, t)$ holds for $\forall x \in V$ when $i = 0$, because $\mathrm{Var}[\hat{\pi}_0(x, t)] = 0$. Assume $\mathrm{Var}[\hat{\pi}_i(x, t)] \leq \frac{\theta}{\lambda(u)} \cdot \pi_i(x, t)$ holds for $\forall x \in V$ and $i < \ell$. Using mathematical induction, we can derive that

$$\begin{aligned} &\mathrm{Var}\left[\hat{\pi}_{\ell+1}(u,t)\right] \leq \frac{\alpha\theta}{\lambda(u)} \cdot \pi_{\ell+1}(u,t) + \frac{(1-\alpha)^2\theta}{\lambda(u)\cdot d_{out}(u)} \cdot \sum_{v\in N_{out}(u)} \pi_\ell(v,t) \\ &= \frac{\alpha\theta}{\lambda(u)} \cdot \pi_{\ell+1}(u,t) + \frac{(1-\alpha)\theta}{\lambda(u)} \cdot \pi_{\ell+1}(u,t) = \frac{\theta}{\lambda(u)} \cdot \pi_{\ell+1}(u,t). \end{aligned}$$

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

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

$$\mathrm{Var}\left[\hat{\pi}_{\ell+1}(u,t)\right] \leq \frac{\theta}{\lambda(u)} \cdot \pi_{\ell+1}(u,t) = \frac{\theta}{\lambda(u)} \cdot \sum_{v\in N_{out}(u)} \frac{(1-\alpha)\pi_\ell(v,t)}{d_{out}(u)}.$$

Note that $\frac{(1-\alpha)\pi_\ell(v,t)}{d_{out}(u)} < \frac{\alpha\theta}{\lambda(u)}$ in the randomized scenario. So,

$$\mathrm{Var}\left[\hat{\pi}_{\ell+1}(u,t)\right] \leq \frac{\theta}{\lambda(u)} \cdot \sum_{v\in N_{out}(u)} \frac{\alpha\theta}{\lambda(u)} = \frac{\alpha\theta^2}{\lambda^2(u)} \cdot d_{out}(u) = \alpha\theta^2.$$

□

A.3 Proof of Lemma 4.5

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

$$\mathrm{E}\left[C_{\ell+1}(u,v) \mid \{\hat{\pi}_\ell\}\right] = \begin{cases} 1, & if \quad \frac{(1-\alpha)\hat{\pi}_\ell(v,t)}{d_{out}(u)} \geq \frac{\alpha\theta}{\lambda(u)}, \\ 1 \cdot \frac{\lambda(u)\cdot(1-\alpha)\hat{\pi}_\ell(v,t)}{\alpha\theta\cdot d_{out}(u)}, & otherwise. \end{cases}$$

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

$$\mathrm{E}\left[C_{\ell+1}(u,v)\right] = \mathrm{E}\left[\mathrm{E}\left[C_{\ell+1}(u,v) \mid \{\hat{\pi}_\ell\}\right]\right] \leq \frac{\lambda(u)\cdot(1-\alpha)\pi_\ell(v,t)}{\alpha\theta\cdot d_{out}(u)}.$$

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

$$\mathrm{E}\left[C_{total}\right] = \sum_{i=1}^{L}\sum_{u\in V}\sum_{v\in N_{out}(u)} \mathrm{E}\left[C_i(u,v)\right]$$

$$\leq \sum_{i=1}^{\infty}\sum_{u\in V}\sum_{v\in N_{out}(u)} \frac{\lambda(u)\cdot(1-\alpha)\pi_{i-1}(v,t)}{\alpha\theta\cdot d_{out}(u)} = \frac{1}{\alpha\theta}\cdot\sum_{u\in V}\lambda(u)\cdot\sum_{i=1}^{\infty}\pi_i(u,t).$$

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

$$\mathrm{E}\left[C_{total}\right] \leq \frac{1}{\alpha\theta}\sum_{u\in V}\lambda(u)\cdot\pi(u,t),$$

which proves the lemma. □

A.4 Proof of Theorem 4.1

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

According to Lemma 4.4, we have $\mathrm{Var}\left[\hat{\pi}_\ell(s,t)\right] \leq \theta\pi_\ell(s,t)$. By Chebyshev inequality, we have

$$\Pr\left[\left|\hat{\pi}_\ell(s,t) - \pi_\ell(s,t)\right| \geq \sqrt{3\theta\pi_\ell(s,t)}\right] \leq 1/3.$$

We claim that this variance implies an ε_r-relative error for all $\pi_\ell(s,t) \geq 3\theta/\varepsilon_r^2$. For a proof, note that $\theta \leq \varepsilon_r^2\pi_\ell(s,t)/3$ and consequently $\sqrt{3\theta\pi_\ell(s,t)} \leq \sqrt{\varepsilon_r^2\pi_\ell(s,t)^2} = \varepsilon_r\pi_\ell(s,t)$. It follows that $\Pr\left[\left|\hat{\pi}_\ell(s,t) - \pi_\ell(s,t)\right| \geq \varepsilon_r\pi_\ell(s,t)\right] \leq 1/3$ for all $\pi_\ell(s,t) \geq 3\theta/\varepsilon_r^2$. By setting $\theta = \frac{\varepsilon_r^2\delta}{3L}$, we obtain a constant relative error guarantee for all $\pi_\ell(s,t) \geq \delta/L$, and consequently a constant relative error for $\pi(s,t) \geq \delta$. 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 $\hat{\pi}_\ell(s,t)$ as the final estimator to $\pi_\ell(s,t)$. 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 $\ell = 0,\ldots,L$, the failure probability becomes $1/n$. Finally, by setting $\lambda(u) = 1$ in Lemma 4.5, we can rewrite the time cost as below.

$$\mathrm{E}\left[C_{total}\right] \leq \frac{1}{\alpha\theta}\sum_{u\in V}\lambda(u)\cdot\pi(u,t) = \frac{1}{\alpha\theta}\sum_{u\in V}\pi(u,t) = \frac{n\pi(t)}{\alpha\theta},$$

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

□

A.5 Proof of Theorem 4.2

Proof. Applying Lemma 4.4, we have $\mathrm{Var}\left[\hat{\pi}_\ell(s,t)\right] \leq \alpha\theta^2$. Consequently, we have $\mathrm{Var}\left[\hat{\pi}(s,t)\right] = \mathrm{Var}\left[\sum_{\ell=0}^{L}\hat{\pi}_\ell(s,t)\right] \leq \alpha L\theta^2$. By Chebyshev's inequality, we have $\Pr\left[\left|\hat{\pi}(s,t) - \pi(s,t)\right| \geq \sqrt{3L\alpha}\theta\right] \leq 1/3$. By setting $\theta = \varepsilon/\sqrt{3L\alpha}$, it follows that $\hat{\pi}(s,t)$ is an ε 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 $\hat{\pi}(s,t)$ 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

$$\mathrm{E}\left[C_{total}\right] \leq \frac{1}{\alpha\theta}\sum_{u\in V}\lambda(u)\cdot\pi(u,t) = \frac{1}{\alpha\theta}\sum_{u\in V}\sqrt{d_{out}(u)}\cdot\pi(u,t).$$

Recall that we set $\theta = \varepsilon/\sqrt{3L\alpha}$ and run $O(\log n)$ independent copies of Algorithm 2, it follows that the running time can be bounded by $\tilde{O}\left(\frac{1}{\varepsilon}\sum_{u\in V}\sqrt{d_{out}(u)}\cdot\pi(u,t)\right)$. If t is chosen uniformly at random, we have $\sum_{t\in V}\pi(u,t) = 1$. Ignoring the $\tilde{O}$ notation, we have

$$\mathrm{E}\left[C_{total}\right] \leq \frac{1}{\varepsilon}\cdot\frac{1}{n}\cdot\sum_{t\in V}\sum_{u\in V}\sqrt{d_{out}(u)}\cdot\pi(u,t)$$

$$= \frac{1}{\varepsilon}\cdot\frac{1}{n}\cdot\sum_{u\in V}\sqrt{d_{out}(u)}\sum_{t\in V}\pi(u,t) = \frac{1}{\varepsilon}\cdot\frac{1}{n}\cdot\sum_{u\in V}\sqrt{d_{out}(u)}.$$

By the AM-GM inequality, we have $\frac{1}{n}\cdot\sum_{u\in V}\sqrt{d_{out}(u)} \leq \sqrt{\frac{\sum_{u\in V}d_{out}(u)}{n}} = \sqrt{\bar{d}}$, Hence, $\mathrm{E}\left[C_{total}\right] \leq \frac{1}{\varepsilon}\cdot\frac{1}{n}\cdot\sum_{u\in V}\sqrt{d_{out}(u)} \leq \frac{\sqrt{\bar{d}}}{\varepsilon}$, and the theorem follows. □