PRSim: Sublinear Time SimRank Computation on Large Power-Law Graphs

Zhewei Wei*
zhewei@ruc.edu.cn
School of Information, DEKE MOE, Renmin University of China

Xiaodong He
hexiaodong_1993@ruc.edu.cn
4Paradigm Inc.
Beijing, China

Xiaokui Xiao
xkxiao@nus.edu.sg
School of Computing, National University of Singapore

Sibo Wang
swang@se.cuhk.edu.hk
The Chinese University of Hong Kong

Yu Liu
dokiliu@pku.edu.cn
Peking University

Xiaoyong Du
Ji-Rong Wen†
{duyong,jrwen}@ruc.edu.cn
Renmin University of China

ABSTRACT

SimRank is a classic measure of the similarities of nodes in a graph. Given a node u in graph $G = (V, E)$, a *single-source SimRank query* returns the SimRank similarities $s(u, v)$ between node u and each node $v \in V$. This type of queries has numerous applications in web search and social networks analysis, such as link prediction, web mining, and spam detection. Existing methods for single-source SimRank queries, however, incur query cost at least linear to the number of nodes n, which renders them inapplicable for real-time and interactive analysis.

This paper proposes PRSim, an algorithm that exploits the structure of graphs to efficiently answer single-source SimRank queries. PRSim uses an index of size $O(m)$, where m is the number of edges in the graph, and guarantees a query time that depends on the *reverse PageRank* distribution of the input graph. In particular, we prove that PRSim runs in sub-linear time if the degree distribution of the input graph follows the power-law distribution, a property possessed by many real-world graphs. Based on the theoretical analysis, we show that the empirical query time of all existing SimRank algorithms also depends on the reverse PageRank distribution of the graph. Finally, we present the first experimental study that evaluates the absolute errors of various SimRank algorithms on large graphs, and we show that PRSim outperforms the state of the art in terms of query time, accuracy, index size, and scalability.

CCS CONCEPTS

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

KEYWORDS

SimRank; Power-Law Graphs; Personalized PageRank

ACM Reference Format:
Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Yu Liu, Xiaoyong Du, and Ji-Rong Wen. 2019. PRSim: Sublinear Time SimRank Computation on Large Power-Law Graphs. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3299869.3319873

1 INTRODUCTION

Measuring similarities and proximities of nodes in the graph is a classic task in graph analytics. Several link-based similarity measures have been proposed, including Personalized PageRank [29], Simfusion [36], P-rank [47] and Panther [45]. Among them, *SimRank* [15], proposed by Jeh and Widom, is regarded as one of the most influential similarity measures, and has been adopted in numerous applications such as web mining [17], social network analysis [23], and spam detection [31]. Given a graph $G = (V, E)$, the SimRank similarity of nodes u and v, denoted as $s(u, v)$, is defined as

$$s(u,v) = \begin{cases} 1, & \text{if } u = v \\ \dfrac{c}{|\mathcal{I}(u)| \cdot |\mathcal{I}(v)|} \displaystyle\sum_{u' \in \mathcal{I}(u)} \sum_{v' \in \mathcal{I}(v)} s(u', v'), & \text{otherwise} \end{cases} \quad (1)$$

*Work partly done at Beijing Key Laboratory of Big Data Management and Analysis Method, Renmin University of China.
†Ji-Rong Wen is the corresponding author.

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3319873

where $\mathcal{I}(u)$ denotes the set of in-neighbors of u, and $c \in (0, 1)$ is a decay factor typically set to 0.6 or 0.8 [15, 26]. This formulation is based on two intuitive statements: (1) two objects are similar if they are referenced by similar objects, and (2) an object is most similar to itself. Due to its recursive nature, SimRank computation is a non-trivial problem and has been extensively studied for more than a decade. Existing work mostly considers three types of SimRank queries: (1) *Single-pair* queries, which ask for the SimRank similarity between two given nodes u and v; (2) *All-pair* queries, which ask for the SimRank similarity between any pair of nodes u and v; (3) *Single-source* queries, which ask for the SimRank similarity between every node and u. All-pair queries require storing $O(n^2)$ node pairs, and thus is infeasible for large graphs. Meanwhile, single-source queries has become the focus of recent research [12, 16, 18, 20, 22, 22, 25, 28, 30, 32, 41], due to its connections to recommendation applications. In this paper, we aim to answer *approximate* single-source SimRank queries, defined as follows:

Definition 1.1 (Approximate Single-Source Queries). Given a node u in a directed graph G and an absolute error threshold ε, an approximate single-source SimRank query returns an estimated value $\hat{s}(u, v)$ for each node v in G, such that

$$|\hat{s}(u, v) - s(u, v)| \leq \varepsilon$$

holds for any v with at least $1 - \delta$ probability. □

Power-law graphs. It was experimentally observed that most real-world networks are scale-free and follow power-law degree distribution. In particular, let $P_o(k)$ and $P_i(k)$ denote the fraction of nodes in the graph having out-degree and in-degree at least k, respectively. Then, on a *power-law graph*, $P_o(k)$ and $P_i(k)$ satisfy that $P_o(k) \sim k^{-\gamma}$ and $P_i(k) \sim k^{-\gamma'}$ [7], where γ and γ' are the (cumulative) power-law exponents that usually take values from 1 to 2. Recent work has demonstrated that by exploiting this fact, we can improve the asymptotic bounds for various graph algorithms such as triangle counting [8], transitive closure [8], perfect matching [8], PageRank computation [27, 35] and maximum independent set [24].

Motivations. Since many graph algorithms can benefit from the structure of real-world graphs, a natural question is: Can we do the same for SimRank algorithms? On one hand, we are interested in designing a more efficient SimRank algorithm by exploiting the structure of the graphs, since existing work for SimRank computation [12, 16, 18, 20, 22, 22, 25, 28, 30, 32, 41] has missed this opportunity for optimization. On the other hand, we are also interested in analyzing how the graph structure affects the performance of existing SimRank algorithms. More precisely, it has been observed in previous work [46] that the performance of existing SimRank algorithms may vary dramatically on graphs with similar numbers of nodes and edges. A typical example is the *Twitter (TW)* and *IT-2004 (IT)* data sets, both of which have around 40 million nodes and 1 billion edges. However, as shown in [46] and in our experiments, the query times of most SimRank algorithms are significantly smaller on *IT-2004* than on *Twitter*. Based on this phenomenon, [46] suggests that *Twitter (TW)* is "locally dense" and *IT-2004 (IT)* is "locally sparse". However, it is still desirable to obtain a quantifiable measure that describes the hardness of each graph in terms of SimRank computation. Finally, since obtaining ground truth for single-source SimRank queries requires n^2 space, which is infeasible for large graphs, most existing work only evaluate the accuracy of the algorithms on small graphs. The only exception is recent work [25], which evaluates precision for approximate top-k queries on graphs with billion edges using the idea of *pooling*. However, there is no prior experimental study that evaluates absolute error for single-source queries on large graphs.

Our contributions. This paper studies the approximate single-source SimRank queries, and makes the following contributions.

(1) We propose PRSim, an algorithm that leverages the graph structure to efficiently answer approximate single-source SimRank queries. The query time complexity of PRSim is related to the *reverse PageRank* of the input graph G, which is defined as the PageRank of the graph G' constructed by reversing the direction of each edge in G. Let $\pi(w)$ denote reverse PageRank of node w, and $\sum_{w \in V} \pi(w)^2$ denote the second moment of the reverse PageRanks. The average expected query cost for PRSim on worst-case graphs is bounded by $O\left(\frac{n \log \frac{n}{\delta}}{\varepsilon^2} \cdot \sum_{w \in V} \pi(w)^2\right)$. By the fact that $\sum_{w \in V} \pi(w)^2 \leq (\sum_{w \in V} \pi(w))^2 = 1$, PRSim provides at least the same complexity as the random walk based algorithms (ProbeSim, TSF, and READS) do on worst-case graphs. Furthermore, PRSim uses an index of size $O(m)$, which significantly improves the scalability of the algorithm. See Table 1 for the theoretical comparison between our algorithm and the state of the art.

On the other hand, we show that on power-law graphs, the second moment $\sum_{w \in V} \pi(w)^2$ is an asymptotic variable that is close to 0, which means PRSim actually achieves sublinear query cost on real-world graphs. More precisely, Let γ denote the cumulative power-law exponent of the out-degree distribution. We show that the average expected query cost for PRSim on power-law graphs is bounded by:

$$\mathrm{E}[Cost] = \begin{cases} O(\frac{1}{\varepsilon^2} \log \frac{n}{\delta}), & \text{for } \gamma > 2; \\ O(\frac{1}{\varepsilon^2} \log \frac{n}{\delta} \cdot \log n), & \text{for } \gamma = 2; \\ O\left(\min\left\{\frac{n^{\frac{1}{\gamma}}}{\varepsilon^{2-\frac{1}{\gamma}}}, \frac{n^{\frac{2}{\gamma}-1}}{\varepsilon^2}\right\}\right), & \text{for } 1 < \gamma < 2, \end{cases} \tag{2}$$

Table 1: Comparison of single-source SimRank algorithms with ε additive error and $1-\delta$ success probability.

Algorithm	Query Time	Query Time (Power-Law Graphs)	Space Overhead	Preprocessing Time
PRSim	$O\left(\frac{n\log\frac{n}{\delta}}{\varepsilon^2}\cdot\sum_{w\in V}\pi(w)^2\right)$	$O\left(\log\frac{n}{\delta}/\varepsilon^2\right)$ for $\gamma>2$ $O\left(\log\frac{n}{\delta}\cdot\log n/\varepsilon^2\right)$ for $\gamma=2$ $O\left(\min\left\{n^{\frac{1}{\gamma}}/\varepsilon^{2-\frac{1}{\gamma}}, n^{\frac{2}{\gamma}-1}/\varepsilon^2\right\}\right)$ for $1<\gamma<2$	$O(\min\{n/\varepsilon, m\})$	$O(m/\varepsilon)$
TSF [30]	$O\left(n\log\frac{n}{\delta}/\varepsilon^2\right)$		$O\left(n\log\frac{n}{\delta}/\varepsilon^2\right)$	$O\left(n\log\frac{n}{\delta}/\varepsilon^2\right)$
READS [16]	$O\left(n\log\frac{n}{\delta}/\varepsilon^2\right)$		$O\left(n\log\frac{n}{\delta}/\varepsilon^2\right)$	$O\left(n\log\frac{n}{\delta}/\varepsilon^2\right)$
ProbeSim [25]	$O\left(n\log\frac{n}{\delta}/\varepsilon^2\right)$		0	0
SLING [32]	$O(n/\varepsilon)$		$O(n/\varepsilon)$	$O\left(m/\varepsilon+n\log\frac{n}{\delta}/\varepsilon^2\right)$

for $\frac{1}{n^{\Omega(1)}}<\varepsilon<1$ and $\delta>\frac{1}{n^{\Omega(1)}}$. To understand this complexity, we first note that when $\gamma\geq 2$, our bounds depend only on $\log n$, which is significantly better than the corresponding bound of any previous SimRank algorithms. For $1<\gamma<2$, since $\varepsilon>\frac{1}{n}$, we have $\frac{n^{\frac{1}{\gamma}}}{\varepsilon^{2-\frac{1}{\gamma}}}\leq\frac{n}{\varepsilon}$. This implies that PRSim also outperforms SLING on power-law graphs. To the best of our knowledge, this is the first sublinear algorithm for single-source SimRank queries on power-law graphs.

(2) To achieve the desired query cost in Table 1, we design several novel techniques for computing SimRank and *Personalized PageRank (PPR)* . First, we propose an algorithm that estimates the *last meeting probabilities* [32] (see Section for definition) for ALL nodes in $O(\log\frac{n}{\delta}/\varepsilon^2)$ time. This improves the $O(n\log\frac{n}{\delta}/\varepsilon^2)$ bounds in [32] by an order of $O(n)$ and is the key to achieve sub-linearity. Second, we propose an index scheme which performs the *backward search* [27] algorithm only on a number j_0 of *hub nodes*. The parameter j_0 enables us to manipulate the tradeoffs between index size and query time, which improves the scalability of our algorithm. Finally, we design *Variance Bounded Backward Walk*, an algorithm that estimates the Personalized PageRank values to a given target node w with additive error ε in $O(n\pi(w)\log\frac{n}{\delta}/\varepsilon^2)$ time, where $\pi(w)$ is the reverse PageRank of node w. Since the average value of $\pi(w)$ is $1/n$, this significantly improves the $O(n\log\frac{n}{\delta}/\varepsilon^2)$ time complexity of the *Randomized Probe* algorithm [25], and is the key to the relation between the time complexity and the reverse PageRank distribution. We also note that the Variance Bounded Backward Walk algorithm actually improves the time complexity of state-of-the-art PPR algorithms to target nodes for dense graphs [33], and may be of independent interest.

(3) Based on the time complexity of PRSim, we conduct experiments to confirm that the hardness of SimRank queries is indeed reversely related to the out-degree power-law exponent γ of the graph. This observation provides a quantifiable measure for the concept of locally dense and locally sparse networks introduced in [46]. In particular, the out-degree distribution of *IT-2004* is significantly more skewed than that of *Twitter* (see Figure 1), which explains the performance discrepancy of existing SimRank algorithms on these two datasets. We also conduct a large set of experiments that evaluate PRSim against the state of the art on benchmark data sets. In particular, our experiments include the first empirical study on the tradeoffs between absolute error and query cost for single-source SimRank algorithms on graphs with billions of edges. Our empirical study shows that PRSim outperforms the state of the art in terms of query time, accuracy, index size, and scalability.

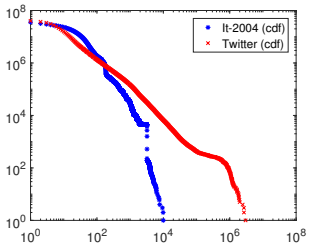


Figure 1: Out-degree distributions of IT and TW.

2 PRELIMINARIES

Table 2 shows the notations that are frequently used in the remainder of the paper.

$\sqrt{c}$-walk and Reverse PageRank. We unify the definition of SimRank and reverse PageRank under the notation of $\sqrt{c}$-walk. Let $G=(V,E)$ be a directed graph with n nodes and m edges. Given a source node $u\in V$ and a decay factor c, a *reverse $\sqrt{c}$-discounted random walk (or $\sqrt{c}$-walk in short)* from u is a traversal of G that starts from u and, at each step, either (i) terminates at the current node with $1-\sqrt{c}$ probability, or (ii) proceeds to a randomly selected in-neighbor of the current node with $\sqrt{c}$ probability. We define the *reverse PageRank* $\pi(w)$ of a node w to be the probability that an $\sqrt{c}$-walk from a uniformly chosen source node terminates at w. It is easy to see that the reverse PageRank of a node w in the original graph G equals to the PageRank of w in the

Table 2: Table of notations.

Notation	Description
n, m	the numbers of nodes and edges in G
$\mathcal{I}(v), O(v)$	the set of in-neighbors and out-neighbors of a node v
$d_{out}(v)$, $d_{in}(v)$	the out-degree and in-degree of node v
$s(u, v)$	the SimRank similarity of nodes u and v
$\hat{s}(u, v)$	an estimation of $s(u, v)$
c	the decay factor of SimRank
ε	the maximum absolute error allowed in SimRank computation
$\pi(w)$	the reverse PageRank of node w
$\pi(u, w)$, $\pi_\ell(u, w)$	the RPPR and ℓ-hop RPPR values of w with respect to u
$\hat{\pi}(u, w)$, $\hat{\pi}_\ell(u, w)$	estimators of $\pi(u, w)$ and $\pi_\ell(u, w)$
$r_\ell(v, w)$, $\psi_\ell(v, w)$	the residue and reserve of v at level ℓ from w in the backward search

reverse graph G' constructed by reversing the direction of each edge in G.

Given a source node u and a target node w, we further define the *reverse Personalized PageRank (RPPR)* $\pi(u, w)$ of w with respect to u to be the probability that an $\sqrt{c}$-walk from u terminates at w. Again, the reverse Personalized PageRank on the original graph G equals to the Personalized PageRank on the reverse graph G'. Since the RPPR values from a given source node u form a probability distribution, we have $\sum_{w \in V} \pi(u, w) = 1$. Meanwhile, since the reverse PageRank $\pi(w)$ is equal to the probability that an $\sqrt{c}$-walk from a random source node terminates at w, we have $\sum_{u \in V} \pi(u, w) = n\pi(w)$.

ℓ-Hop RPPR. In this paper, we will mainly use a variant of Personalized PageRank called *ℓ-hop Reverse Personalized PageRank (ℓ-hop RPPR)*. Given a source node u, the ℓ-hop RPPR $\pi_\ell(u, w)$ of node w respected to u is the probability that a reverse $\sqrt{c}$-walk from u terminates at node w with exactly ℓ steps. By the definition of ℓ-hop RPPR, we have

$$\pi_{\ell+1}(y, w) = \sum_{x \in \mathcal{I}(y)} \frac{\sqrt{c}}{d_{in}(y)} \pi_\ell(x, w). \tag{3}$$

On the other hand, it is easy to see that RPPR $\pi(u, w)$ can be expressed as the sum of ℓ-hop RPPR, that is, $\sum_{\ell=0}^{\infty} \pi_\ell(u, w) = \pi(u, w)$. Thus, we have $\sum_{\ell=0}^{\infty} \sum_{w \in V} \pi_\ell(u, w) = 1$, and

$$\sum_{\ell=0}^{\infty} \sum_{u \in V} \pi_\ell(u, w) = n\pi(w). \tag{4}$$

SimRank, $\sqrt{c}$-walk, and hitting probability. It is shown in [32] that the SimRank similarity $s(u, v)$ between two different nodes u and v can also be formulated using $\sqrt{c}$-walks. Given two distinct nodes u and v, we start a $\sqrt{c}$-walk from each node. If the two $\sqrt{c}$-walks visit the same node after exactly i steps, we say the two $\sqrt{c}$-walks meet at step i. [32] shows that $s(u, v)$ is equal to the probability that the two $\sqrt{c}$-walks meet.

Moreover, [32] proposes SLING, an algorithm that uses the following formula to estimate SimRank values:

$$s(u, v) = \sum_{\ell=0}^{\infty} \sum_{w \in V} h_\ell(u, w) h_\ell(v, w) \eta(w). \tag{5}$$

Here $h_\ell(u, w)$ denote the *hitting probability* that an $\sqrt{c}$-walk from node u *visits* w in its ℓ-step, and $\eta(w)$ is a parameter that characterizes the last-meeting probability:

Definition 2.1 (Last-meeting probability). The last-meeting probability $\eta(w)$ for node w is the probability that two $\sqrt{c}$-walk from w do not meet at i step for any $i \geq 1$.

SLING precomputes $h_\ell(u, w)$ and $\eta(w)$ with an additive error up to ε, and stores them in the index. Given a query node u, it retrieves all levels ℓ and nodes w such that $h_\ell(u, w) > \varepsilon$. For each (ℓ, w) pair, SLING retrieves all nodes v with $h_\ell(v, w) > \varepsilon$ and $\eta(w)$, and estimates $s(u, v)$ with Equation (5).

There are two major issues with SLING. First, storing all $h_\ell(u, w)$ with additive error up to ε takes $O(n/\varepsilon)$ space, which can be significantly larger than the graph size for reasonable choices of ε. Second, approximating $\eta(w)$ for each $w \in V$ requires sampling a large number of random walks from each node in the graph, which makes the preprocessing time infeasible on very large graphs. Our algorithm overcomes these two drawbacks by (1) providing an index size that is at most the size of the graph, and (2) designing an algorithm that estimates $\eta(w)$ on-the-fly, using only $O(\log n/\varepsilon^2)$ time.

3 PRSIM ALGORITHM

In this section, we present PRSim, an index-based algorithm that exploits the graph structure to efficiently answer approximate single-source SimRank queries. We first provide the estimating formula that relates SimRank and ℓ-hop RPPR.

3.1 SimRank and ℓ-hop RPPR

The relation between SimRank and reverse Personalized PageRank can be directly derived from equation (5). Observe the fact that ℓ-hop RPPR $\pi_\ell(u, w)$ equals to the hitting probability $h_\ell(u, w)$ multiplied the the termination probability $\alpha = 1 - \sqrt{c}$, and we have

$$s(u, v) = \frac{1}{(1 - \sqrt{c})^2} \sum_{\ell=0}^{\infty} \sum_{w \in V} \pi_\ell(u, w) \pi_\ell(v, w) \eta(w). \tag{6}$$

There are two reasons for using ℓ-hop RPPR over hitting probability. Firstly, we have $\sum_{\ell=0}^{\infty}\sum_{w\in V}\pi_\ell(u,w) = 1$. As we will show later, this is critical for estimating $\eta(w)$ in $O(\log\frac{n}{\delta}/\varepsilon^2)$ time. Secondly, we have $\sum_{\ell=0}^{\infty}\sum_{u\in V}\pi_\ell(u,w) = n\pi(w)$. This property relates SimRank with the reverse PageRank, and thus is essential for achieving sublinear query time.

Recall that given a source node u, our goal is to estimate SimRank values $s(u,v)$ with additive error ε for any node $v \in V$. By Equation (6), we can decompose the query process into three subroutines: 1) Given a source node u, compute the ℓ-hop RPPR values $\pi_\ell(u,w)$ for any nodes $w \in V$; 2) Compute last meeting probabilities $\eta(w)$ for each $w \in V$; 3) For any node $v \in V$, compute ℓ-hop RPPR values $\pi_\ell(v,w)$ to any target node w. For the first task, we can employ a simple Monte Carlo algorithm which generates a number $n_r = O(\log\frac{n}{\delta}/\varepsilon^2)$ of $\sqrt{c}$-walks from u and uses the proportion of $\sqrt{c}$-walks that terminate at w with exact ℓ steps to approximate $\pi_\ell(u,w)$. This algorithm runs in $O(\log\frac{n}{\delta}/\varepsilon^2)$ time, so we will focus on the remaining two tasks.

3.2 Computing Last Meeting Probability

The first challenge is how to estimate $\eta(w)$ for each $w \in V$ efficiently. SLING [32] generates $n_r = \Theta\left(\frac{\log\frac{n}{\delta}}{\varepsilon^2}\right)$ pair of $\sqrt{c}$-walks for each $w \in V$, and obtains an approximation to $\eta(w)$ with error ε for each $w \in V$. However, this solution leads to a preprocessing time of $O\left(\frac{n\log\frac{n}{\delta}}{\varepsilon^2}\right)$, and thus, is not feasible if we need small error ε on large graphs.

Our first key insight is that, instead of estimating the ℓ-hop PPR $\pi_\ell(u,w)$ and last meeting probability $\eta(w)$ separately, we can estimate their product $\eta(w)\pi_\ell(u,w)$ in the query phase, using only $n_r = \Theta\left(\frac{\log\frac{n}{\delta}}{\varepsilon^2}\right)$ samples. More precisely, we observe that $\eta(w)\pi_\ell(u,w)$ is the probability that an $\sqrt{c}$-walk from u terminates at w with ℓ steps, and then, two independent $\sqrt{c}$-walks from w do not meet. Therefore, we can generate an $\sqrt{c}$-walk $\mathcal{W}(u)$ from u, and then two $\sqrt{c}$-walks $\mathcal{W}_1(w)$ and $\mathcal{W}_2(w)$ from the node w where $\mathcal{W}(u)$ terminates. If $\mathcal{W}_1(w)$ and $\mathcal{W}_2(w)$ do not meet, we set the estimator $\widehat{\eta\pi}_\ell(u,w) = 1$. This way we obtain an unbiased estimator for each $\eta(w)\pi_\ell(u,w)$, $w \in V$ and $\ell = 0,\ldots,\infty$. We also note that the summation $\sum_{w\in V}\sum_{\ell=0}^{\infty}\eta(w)\pi_\ell(u,w) \le \sum_{w\in V}\sum_{\ell=0}^{\infty}\pi_\ell(u,w) = 1$, which means we can use Chernoff bound A.1 to estimates $\eta(w)\pi_\ell(u,w)$ with additive error ε for any $w \in V, \ell \ge 0$ with only $n_r = \Theta\left(\frac{\log\frac{n}{\delta}}{\varepsilon^2}\right)$ samples.

3.3 Precomputing RPPR to Hub Nodes

Given a target node w, computing ℓ-hop RPPR $\pi_\ell(v,w)$ for any node $v \in V$ is time-consuming, especially when w is a hub node with many out-neighbors. Therefore, we will use index to help reduce the cost. SLING [32] proposes the following approach: for each (source) node v, we precompute $\pi_\ell(v,w)$ for any $w \in V$ and put $\pi_\ell(v,w)$ into an inverted list, so we can efficiently track $\pi_\ell(v,w), v \in V$ for a given target node w. This approach, however, essentially builds an index for every target node $w \in V$ and results in an index of size $O\left(\frac{n}{\varepsilon}\right)$, which is usually significantly larger than the graph size m for reasonably small ε.

Algorithm 1: Preprocessing Algorithm

Input: Graph G, decay factor c, error parameter ε
Output: Lists $L_\ell(w)$ consisting of tuples $(v,\psi_\ell(v,w))$ for each w with top-j_0 reverse PageRank values and $\ell = 0,\ldots,\infty$

Construct a tuple $(x,y,d_{in}(y))$ for each edge $(x,y) \in E$;
Use counting sort to sort the $(x,y,d_{in}(y))$ tuples according the ascending order of $d_{in}(y)$.;
for *each* $(x,y,d_{in}(y))$ **do**
  Append y to the end of x's out-adjacency list;
Calculate reverse PageRank $\pi(w)$ for $w \in V$;
for *each node w with top-j_0 reverse PageRank values* **do**
  $r_\ell(v,w), \psi_\ell(v,w) \leftarrow 0$ for $\ell = 0,\ldots,\infty, v \in V$;
  $r_0(w,w) \leftarrow 1$, $c_1 \leftarrow \frac{12}{(1-\sqrt{c})^2}$, $r_{max} \leftarrow \frac{\varepsilon}{c_1}$;
  for ℓ *from* 0 *to* ∞ **do**
    for *each* $v \in V$ *with* $r_\ell(v,w) > r_{max}$ **do**
      for *each* $z \in O(v)$ **do**
        $r_{\ell+1}(z,w) \leftarrow r_{\ell+1}(z,w) + \sqrt{c}\cdot\frac{r_\ell(v,w)}{d_{in}(z)}$
      $\psi_\ell(v,w) \leftarrow \psi_\ell(v,w) + (1-\sqrt{c})\cdot r_\ell(v,w)$;
      $r_\ell(v,w) \leftarrow 0$;
    for *each* v *with reserve* $\psi_\ell(v,w) > r_{max}$ **do**
      Append tuple $(v,\psi_\ell(v,w))$ to $L_\ell(w)$;

To reduce the index size, we propose to build index only for *hub nodes*. In particular, we identify j_0 nodes with the largest reverse PageRanks as hub nodes, where j_0 is a user-specified parameter. We then perform the *backward search* [27] algorithm on each hub node w to precompute $\pi_\ell(v,w)$ for any $v \in V$ and any $\ell > 0$. The definition of hub nodes is based on two intuitions. First, recall that the reverse PageRank of node w is the probability that an $\sqrt{c}$-walk from a random node u terminates at w. Therefore, a hub node w is more likely to be visited in a single-source SimRank query on u. Second, since $\sum_{\ell=0}^{\infty}\sum_{v\in V}\pi_\ell(v,w) = n\pi(w)$, a hub node will also have more (ℓ,w)-tuples with $\pi_\ell(v,w) > \varepsilon$, which makes it more difficult to compute $\pi_\ell(v,w)$ on the fly. Therefore, pre-computing $\pi_\ell(v,w)$ for nodes w with highest reverse PageRank reduces the query cost most efficiently. We also note that we can choose the value of j_0 to balance the query time, index size and preprocessing time. For ease of presentation, we select j_0 such that the index size is bounded by $O(m)$ in this section.

Algorithm 1 illustrates the pseudocode for the preprocessing algorithm. For reasons we shall see later, for each node u with out-neighbor set $O(x) = \{y_1, \ldots, y_d\}$, we store the adjacency list of x in a way such that $d_{in}(y_1) \leq \ldots \leq d_{in}(y_d)$. To sort the adjacency list of each node in total $O(m)$ time, we first construct a tuple $(x, y, d_{in}(y))$ for each edge $(x, y) \in E$. Then we employ the *counting sort* algorithm to sort the m tuples $(x, y, d_{in}(y))$ according to the ascending order of $d_{in}(y)$. Since $d_{in}(y)$ is an integer in range $[0, n]$, the counting sort algorithm runs in time $O(m + n)$. We then scan the m sorted tuples and, for each tuple $(x, y, d_{in}(y))$, we append y to the end of x's out-adjacency list. This algorithm sorts the out-adjacency list of each node in $O(m + n)$ time. (Lines 1-4). We then calculate the reverse PageRanks for each node $w \in V$, and retrieve the j_0 nodes with the largest reverse PageRank as the hub nodes (line 5). For each hub node w, we use backward search [27] to compute an estimator $\psi_\ell(v, w)$ for the l-hop RPPR $\pi_\ell(v, w)$, for each $\ell = 0, \ldots, \infty$ and $v \in V$. More precisely, we first set *residue* $r_\ell(v, w)$ and a *reserve* $\psi_\ell(v, w) = 0$ to each node v and $\ell = 0, \ldots, \infty$. Then, we set $r_0(w, w) = 1$ and the residue threshold $r_{max} = \frac{(1-\sqrt{c})^2\varepsilon}{12}$ (Lines 6-8). Note that we choose the constant $(1 - \sqrt{c})^2$ to compensate the denominator $(1 - \sqrt{c})^2$ in equation (6), and the constant 12 so that we can sum various errors up to at most ε. Starting from level 0, we traverse from w, following the out-going edges of each node (Line 9). On visiting a node v at level ℓ, we check if v's residue $r_\ell(v, t)$ is larger than the threshold r_{max}. If so, for each out-neighbor z of v, we increase the residue $r_{\ell+1}(z, w)$ of z at level $\ell+1$ by $\sqrt{c} \cdot \frac{r_\ell(v,w)}{d_{in}(z)}$ (Lines 10-12). Next, we increase $\psi_\ell(v, w)$, v's backward reserve at level ℓ by $\sqrt{c}r_\ell(v, w)$ (line 13). After that, we reset v's backward residue $r_\ell(v, w)$ to 0 (line 14). After all nodes v with residue $r_\ell(v, w) > r_{max}$ are processed, we append tuples $(v, \psi_\ell(v, w))$ to a list $L_\ell(w)$ for each v with reserve $\psi_\ell(v, w) > r_{max}$ (line 15-17). Note that for each a node w and a level ℓ with at least one $\psi_\ell(v, w) > r_{max}$, we store all tuples $(v, \psi_\ell(v, w))$ with $\psi_\ell(v, w) > \varepsilon$ in a list $L_\ell(w)$, so we can quickly retrieve them given w and ℓ in the query phase. The following lemma can be directly derived from [27]

Lemma 3.1 ([27]). *For any hub node w, any $v \in V$ and $\ell \geq 0$, Algorithm1 ensures $|\psi_\ell(v, w) - \pi_\ell(v, w)| < r_{max} = \frac{(1-\sqrt{c})^2\varepsilon}{12}$.*

We have the following lemma that bounds the space usage and running time of Algorithm 1 on worst-case graphs.

Lemma 3.2. *The size of the index generated by Algorithm 1 is bounded by $O\left(\frac{n}{\varepsilon}\sum_{j=1}^{j_0}\pi(w_j)\right)$. The preprocessing time is bounded by $O\left(\frac{m}{\varepsilon}\right)$.*

We set j_0 so that $O\left(\frac{n}{\varepsilon}\sum_{j=1}^{j_0}\pi(w_j)\right) = O(m)$ in the theoretical analysis of PRSim, for ease of presentation. Note that if the largest reverse PageRank $\pi(w_1)$ satisfies $\pi(w_1) > \varepsilon m/n$, we need to set $j_0 = 0$, in which case PRSim becomes an index-free algorithm. However, in practice, we can manipulate j_0 to get a tradeoff between the index size and query cost.

3.4 Sampling RPPR to Non-Hub Nodes

The third key component of our method is a sampling-based algorithm that efficiently computes ℓ-Hop PPR values to non-hub target nodes (i.e., nodes with small reverse PPR values and thus are not in the index). Given a node w, the goal is to provide an unbiased estimator $\hat{\pi}_\ell(v, w)$ for $\pi_\ell(v, w)$ for each $v \in V$ and any $\ell \geq 0$. Once we obtain such a sampler, we can estimate each $\pi_\ell(v, w)$ with additive error ε using $\log \frac{n}{\delta}/\varepsilon^2$ samples. [25] provides such a sampler by employing a *Randomized Probe* algorithm, which runs in $O(n)$ time for a single sample. This time complexity, however, is unacceptable if we want sub-linear query time.

In this section, we propose an algorithm that achieves the following goals: 1) Given a node w, the algorithm provides an unbiased estimator $\hat{\pi}_\ell(v, w)$ for $\pi_\ell(v, w)$, for each $v \in V$ and any $\ell \geq 0$; 2) the algorithm runs in $O(n\pi(w))$ expected time. Note that $n\pi(w) = \sum_{i=0}^{\infty}\sum_{v\in V}\pi_i(v, w)$ is the expected output size and consequently the minimum cost for generating unbiased estimators $\hat{\pi}_i(v, w)$ for $i = 0, \ldots, \infty$, $v \in V$. (3) The variance of $\hat{\pi}_i(v, w)$ is bounded, so we can use Chebyshev's inequality to bound the error, and the Median Trick to boost the success probability.

Algorithm 2: Backward Walk

Input: Directed graph $G = (V, E)$; node $w \in V$; level ℓ
Output: $\hat{\pi}_\ell(v, w)$ for each $v \in V$

1 $\hat{\pi}_\ell(v, w) \leftarrow 0$ for $\ell = 0, \ldots, \infty$, $x \in V$;
2 $\hat{\pi}_0(w, w) \leftarrow 1 - \sqrt{c}$;
3 **for** $i = 0$ *to* $\ell - 1$ **do**
4 **for** *each* $x \in V$ *with non-zero* $\hat{\pi}_i(x, w)$ **do**
5 $r \leftarrow rand(0, 1)$;
6 **for** *each* $y \in O(x)$ *and* $d_{in}(y) \leq \frac{\sqrt{c}}{r}$ **do**
7 $\hat{\pi}_{i+1}(y, w) \leftarrow \hat{\pi}_{i+1}(y, w) + \hat{\pi}_i(x, w)$;
8 **return** all non-zero $\hat{\pi}_\ell(v, w)$;

Simple Backward Walk with Unbounded Variance. For ease of exposition, we first present a simple *Backward Walk* that achieves the first two goals. The pseudocode is illustrated by Algorithm 2. Given a node w and a level ℓ, this algorithm also gives an unbiased estimator $\hat{\pi}_\ell(v, w)$ for each $v \in V$. We first initialize $\hat{\pi}_0(w, w) = 1 - \sqrt{c}$ and $\hat{\pi}_\ell(x, w) = 0$ for other ℓ or $x \in V$ (Lines 1-2). Then, we iterate i from 0 to $\ell-1$ (Line 3). At level i, for each $x \in V$ with non-zero $\hat{\pi}_i(x, w)$, we generate a random number r from $(0, 1)$ (Line 4-5), and scan the out-neighbors of x until we encounter the first node y with $d_{in}(y) > \frac{\sqrt{c}}{r}$. Recall that in the preprocessing phase,

we sort the out adjacency list of x so that nodes in $O(x)$ are ordered according to their in-degrees (see Algorithm 1). Therefore, we only have to visit the nodes with $d_{in}(y) \le \frac{\sqrt{c}}{r}$, which is a subset of $O(x)$. For each out-neighbor y of x with $d_{in}(y) \le \frac{\sqrt{c}}{r}$, we add $\hat{\pi}_i(x,w)$ to $\hat{\pi}_{i+1}(y,w)$ (Lines 6-7). Finally, after level $\ell-1$ is processed, we return each non-zero $\hat{\pi}_\ell(v,w)$ as the estimator for $\pi_\ell(v,w)$ (Line 8).

We can use a simple induction to prove the unbiasedness of Algorithm 2. For the base case, we have $\mathrm{E}[\hat{\pi}_0(w,w)] = 1-\sqrt{c} = \pi_0(w,w)$. Assume that $\mathrm{E}[\hat{\pi}_i(x,w)] = \pi_i(x,w)$ for any $x \in V$. For a node y at level $i+1$, each $\hat{\pi}_i(x,w), x \in \mathcal{I}(y)$ is added to $\hat{\pi}_{i+1}(y,w)$ with probability $\frac{\sqrt{c}}{d_{in}(y)}$, and thus $\mathrm{E}[\hat{\pi}_{i+1}(y,w)] = \sum_{x\in\mathcal{I}(y)} \frac{\sqrt{c}}{d_{in}(y)} \mathrm{E}[\hat{\pi}_i(x,w)]$. Therefore, we have $\mathrm{E}[\hat{\pi}_{i+1}(y,w)] = \sum_{x\in\mathcal{I}(y)} \frac{\sqrt{c}}{d_{in}(y)} \pi_i(x,w) = \pi_{i+1}(y,w)$. To analyze the running time, note that the cost for computing $\hat{\pi}_i(x,w)$ is bounded by the number of times that $\hat{\pi}_i(x,w)$ is incremented. Since each increment adds at least $(1-\sqrt{c})$ to $\hat{\pi}_i(x,w)$, this cost is bounded by $\frac{\hat{\pi}_i(x,w)}{1-\sqrt{c}}$. Summing over $i = 0, \ldots, \infty$ and $x \in V$, and using equation (4), the total cost is at most $O(n\pi(w))$.

Unfortunately, the estimator $\hat{\pi}_\ell(v,w)$ returned by Algorithm 2 can be unbounded, since we may sum up all estimators from level i to form an estimator of level $i+1$. To make thing worse, it is even unclear if $\hat{\pi}_\ell(v,w)$ has bounded variance. This means that $\hat{\pi}_\ell(v,w)$ may not be sub-gaussian or sub-exponential, and thus we are unable to apply concentration inequality to bound the error.

Algorithm 3: Variance Bounded Backward Walk

Input: Directed graph $G = (V, E)$; node $w \in V$; target level ℓ
Output: $\hat{\pi}_\ell(v,w)$ for each $v \in V$

1 $\hat{\pi}_\ell(v,w) \leftarrow 0$ for $\ell = 0, \ldots, \infty, x \in V$;
2 $\hat{\pi}_0(w,w) \leftarrow 1-\sqrt{c}$;
3 **for** $i = 0$ *to* $\ell-1$ **do**
4 **for** *each* $x \in V$ *with non-zero* $\hat{\pi}_i(x,w)$ **do**
5 **if** $r_0 \leftarrow rand() < \sqrt{c}$ **then**
6 **for** *each* $y \in O(x)$ *and* $d_{in}(y) \le \frac{\hat{\pi}_i(x,w)}{1-\sqrt{c}}$ **do**
7 $\hat{\pi}_{i+1}(y,w) \leftarrow \hat{\pi}_{i+1}(y,w) + \frac{\hat{\pi}_i(x,w)}{d_{in}(y)}$;
8 $r \leftarrow rand(0,1)$;
9 **for** *each* $y \in O(x)$ *and* $\frac{\hat{\pi}_i(x,w)}{1-\sqrt{c}} < d_{in}(y) \le \frac{\hat{\pi}_i(x,w)}{r(1-\sqrt{c})}$ **do**
10 $\hat{\pi}_{i+1}(y,w) \leftarrow \hat{\pi}_{i+1}(y,w) + 1 - \sqrt{c}$;
11 **return** all non-zero $\hat{\pi}_\ell(v,w)$;

Variance Bounded Backward Walk. To overcome the drawback of simple Backward Walk, we propose the *Variance Bounded Backward Walk* algorithm, which achieves bounded variance without sacrificing the $O(n\pi(w))$ query bound or the unbiasedness guarantee. Algorithm 3 illustrates the pseudocode of the Variance Bounded Backward Walk algorithm. We set $\hat{\pi}_0(w,w) = 1-\sqrt{c}$ and $\hat{\pi}_\ell(x,w) = 0$ for other ℓ or $x \in V$ (Lines 1-2). Then we iterate i from 0 to $\ell-1$ (Line 3). At level i, for each $x \in V$ with non-zero $\hat{\pi}_i(x,w)$, we first generate a random number r_0 so that we can stop the process at x with probability $1-\sqrt{c}$ (Lines 4-5). With probability $\sqrt{c}$, we first scan through the out-neighbors of x until we encounter the first node y with $d_{in}(y) > \frac{\hat{\pi}_i(x,w)}{1-\sqrt{c}}$. For each out-neighbor y with $d_{in}(y) \le \frac{\hat{\pi}_i(x,w)}{1-\sqrt{c}}$ we increase $\hat{\pi}_i(y,w)$ by $\frac{\hat{\pi}_i(x,w)}{d_{in}(y)}$ (Lines 6-7). Then, we choose a random number r from $(0,1)$ (Line 8), and continue to scan the out-neighbors of x until we encounter the first node y with $d_{in}(y) > \frac{\hat{\pi}_i(x,w)}{r(1-\sqrt{c})}$. Again, we only visit a subset of $O(x)$, as the nodes in $O(x)$ are ordered according to their in-degrees. For each out-neighbor y of x with $d_{in}(y) \le \frac{\hat{\pi}_i(x,w)}{r(1-\sqrt{c})}$, we increment $\hat{\pi}_{i+1}(y,w)$ by $1-\sqrt{c}$ (Lines 9-10). After ℓ levels are processed, we return all non-zero $\hat{\pi}_\ell(v,w)$ as estimators for $\pi_\ell(v,w)$ (Line 11).

Analysis. We prove three properties of the Variance Bounded Backward Walk algorithm. First, the algorithm gives an unbiased estimator $\hat{\pi}_\ell(v,w)$ for $\pi_i(v,w)$ for each $v \in V$ and $i \le \ell$. In particular, we have the following lemma.

Lemma 3.3. *Consider a node v on a target level ℓ, and let $\hat{\pi}_\ell(v,w)$ be an estimator provided by Algorithm 3. We have* $\mathrm{E}[\hat{\pi}_\ell(v,w)] = \pi_\ell(v,w)$.

Next, we show that the running time of Algorithm 3 on node w is proportional to its reverse PageRank $\pi(w)$. In particular, we have the following lemma.

Lemma 3.4. *The complexity of Algorithm 3 on node w, regardless of the target level ℓ, is bounded by $O(n\pi(w))$.*

Note that $n\pi(w) = \sum_{i=0}^{\infty} \sum_{v\in V} \pi_i(v,w)$, which implies that the minimum number of operations to return a unbiased estimator $\hat{\pi}_i(v,w)$ for each $\pi_i(v,w)$ is $\Omega(n\pi(w))$. This essentially means that Algorithm 3 achieves optimal sampling complexity for this task.

Finally, we note that although the estimator $\hat{\pi}_\ell(v,w)$ is unbiased, it may be unbounded on certain graphs. To see this, consider a graph that has $n+2$ nodes $w, v, x_1, \ldots, x_n$. For each $i = 1, \ldots, n$, there is an edge from w to x_i and an edge from x_i to v. Suppose we run Algorithm 2 on node w with target level $\ell = 2$. The algorithm first sets $\hat{\pi}_0(w,w) = 1-\sqrt{c}$. For each $i = 1, \ldots, n$, the algorithm sets $\hat{\pi}_1(x_i,w) = 1-\sqrt{c}$ with probability $\sqrt{c}$. This means there are approximately $\sqrt{c}$ fraction of x_i's with $\hat{\pi}_1(x_i,w) = 1-\sqrt{c}$. Finally, for each $i = 1, \ldots, n$ and $\hat{\pi}_1(x_i,w) = 1-\sqrt{c}$, the algorithm increments $\hat{\pi}_2(v,w)$ by $1-\sqrt{c}$ with probability $\frac{1}{n}$. This implies that in

the worst-case, all $\hat{\pi}_1(x_i, w) = 1 - \sqrt{c}$ for $i = 1, \ldots, n$, and $\hat{\pi}_2(v, w)$ can be as large as $(1 - \sqrt{c})n$.

Fortunately, we can bound the variance of Algorithm 3, which enables us to use the Median Trick to boost accuracy. The following lemma states that the variance of $\hat{\pi}_\ell(v, w)$ is bounded by $\pi_\ell(v, w)$, the actual value of the ℓ-hop RPPR.

Lemma 3.5. *For any level $\ell \geq 0$ and node $v \in V$, we have* $\mathrm{Var}\left[\hat{\pi}_\ell(v, w)\right] \leq \mathrm{E}\left[\hat{\pi}_\ell(v, w)^2\right] \leq \pi_\ell(v, w)$.

3.5 Putting Things Together

Based on the definition of hub nodes, we divide the SimRank value $s(u, v)$ of nodes u and v into two terms $s(u, v) = s_I(u, v) + s_B(u, v)$, where

$$s_I(u, v) = \frac{1}{(1 - \sqrt{c})^2} \sum_{\ell=0}^{\infty} \sum_{j=1}^{j_0} \pi_\ell(u, w_j)\pi_\ell(v, w_j)\eta(w_j), \quad (7)$$

and

$$s_B(u, v) = \frac{1}{(1 - \sqrt{c})^2} \sum_{\ell=0}^{\infty} \sum_{j=j_0+1}^{n} \pi_\ell(u, w_j)\pi_\ell(v, w)\eta(w_j). \quad (8)$$

PRSim algorithm uses pre-computed index to generate an estimator $\hat{s}_I(u, v)$ for $s_I(u, v)$, and uses backward walks to generate an estimator $\hat{s}_B(u, v)$ for $s_B(u, v)$.

Algorithm 4 shows the pseudo-code of the query algorithm for PRSim. Given a source node u on a directed graph $G = (V, E)$, a decay factor c and an error parameter ε, the algorithm returns an estimator $\hat{s}(u, v)$ for each $v \in V$. We set the constant $c_1 = \frac{12}{(1-\sqrt{c})^2}$, the number of samples in a round to $d_r = \frac{c_1}{\varepsilon^2}$, the number of rounds to $f_r = 3 \log \frac{n}{\delta}$, and the total sample number to $n_r = d_r f_r = \Theta\left(\frac{\log \frac{n}{\delta}}{\varepsilon^2}\right)$ (Line 1). Note that for the constant c_1, we choose $(1 - \sqrt{c})^2$ to compensate the denominator $(1 - \sqrt{c})^2$ in equation (6), and 12 so that we can sum various errors up to at most ε. We choose the value of d_r according to Chernoff bound A.1, and the value of f_r according to the Median Trick A.3. Then we initialize estimators $\hat{s}(u, v)$ $\hat{s}_I(u, v)$, $\hat{s}_B(u, v)$ and $\hat{s}^i_B(u, v)$ to be 0 for $v \in V$ and $i = 1, \ldots, f_r$ (Line 2). We also set $\widehat{\eta\pi}_\ell(u, w)$, the estimator for $\eta(w) \cdot \pi_\ell(u, w)$, to be 0 for $w \in V$ and $\ell = 0, \ldots, \infty$ (Line 3). Note that in order to achieve sublinear query time, we can use hash maps to store only the non-zero entries in $\hat{s}$, $\hat{s}_B$ $\hat{s}_I$, $\hat{s}^i_B$ and $\widehat{\eta\pi}$.

For each i from 1 to f_r and j from 1 to d_r, we sample an $\sqrt{c}$-walk $\mathcal{W}(u)$ from u (Lines 4-6). If $\mathcal{W}(u)$ terminates at node w in ℓ steps, we further sample a pair of $\sqrt{c}$-walks $\mathcal{W}_1(w)$ and $\mathcal{W}_2(w)$ from w (Line 8). Recall that the probability that the two $\sqrt{c}$-walks do not meet is exactly $\eta(w)$. If this event happens, we increase the estimator $\widehat{\eta\pi}_\ell(u, w)$ by $\frac{1}{n_r}$ (Lines 9-10). If w is not stored in the index, we estimate $\pi_\ell(v, w)$ for each $v \in V$ with Algorithm 3, and update the i-th estimator $\hat{s}^i_B(u, v)$ by $\frac{\hat{\pi}_\ell(v,w)}{(1-\sqrt{c})^2 d_r}$ for each $v \in V$ (Lines 11-13). After $n_r = d_r \cdot f_r$ samples are processed, we return $\hat{s}_B(u, v) = \mathrm{Median}_{1 \leq i \leq f_r} \hat{s}^i_B(u, v)$ as an estimator for $s_B(u, v)$ (Lines 14-15). Again, to ensure sublinear query time, we only compute median for a node v if there is at least one non-zero $\hat{s}^i_B(u, v)$ for some $1 \leq i \leq f_r$. Finally, for each (w, ℓ)-tuple with $\widehat{\eta\pi}_\ell(u, w) > \frac{\varepsilon}{c_1}$ and w in the index, we retrieve $\hat{\pi}_\ell(v, w)$ for each $v \in V$ from the index, and update $\hat{s}_I(u, v)$ by $\frac{\widehat{\eta\pi}_\ell(v,w)}{(1-\sqrt{c})^2}$ (Lines 16-18). We return all non-zero $\hat{s}(u, v) = \hat{s}_I(u, v) + \hat{s}_B(u, v)$ as the estimator for $s(u, v)$, for $v \in V$ (Line 19).

Algorithm 4: Query Algorithm

Input: Directed graph $G = (V, E)$; node u; decay factor c; error parameter ε; Failure probability δ
Output: $\hat{s}(u, v)$ for each $v \in V$

1 $c_1 \leftarrow \frac{12}{(1-\sqrt{c})^2}$, $d_r \leftarrow \frac{c_1}{\varepsilon^2}$, $f_r \leftarrow 3 \log \frac{n}{\delta}$, $n_r \leftarrow d_r \cdot f_r$;
2 $\hat{s}(u, v), \hat{s}_I(u, v), \hat{s}_B(u, v), \hat{s}^i_B(u, v) \leftarrow 0$ for each $v \in V$, $i = 1, \ldots, f_r$;
3 $\widehat{\eta\pi}_\ell(u, w) \leftarrow 0$ for $w \in V$, $\ell = 0, \ldots, \infty$;
4 **for** $i = 1$ *to* f_r **do**
5 **for** $j = 1$ *to* d_r **do**
6 Sample an $\sqrt{c}$-walk $\mathcal{W}(u)$ from u ;
7 **if** $\mathcal{W}(u)$ *terminates at node* w *with* ℓ *steps* **then**
8 Sample two independent $\sqrt{c}$-walks $\mathcal{W}_1(w)$ and $\mathcal{W}_2(w)$ from w;
9 **if** $\mathcal{W}_1(w)$ *and* $\mathcal{W}_2(w)$ *do not meet* **then**
10 $\widehat{\eta\pi}_\ell(u, w) \leftarrow \widehat{\eta\pi}_\ell(u, w) + \frac{1}{n_r}$;
11 **if** $w \notin$ *Index* **then**
12 Estimate $\hat{\pi}_\ell(v, w)$ for $v \in V$ with Algorithm 3;
13 $\hat{s}^i_B(u, v) \leftarrow \hat{s}^i_B(u, v) + \frac{\hat{\pi}_\ell(v,w)}{(1-\sqrt{c})^2 d_r}$;
14 **for** *each* v *with nonzero* $\hat{s}^i_B(u, v)$ *for some* $1 \leq i \leq f_r$ **do**
15 $\hat{s}_B(u, v) \leftarrow \mathrm{Median}_{1 \leq i \leq f_r} \hat{s}^i_B(u, v)$;
16 **for** *each* (w, ℓ) *with* $\widehat{\eta\pi}_\ell(u, w) > \frac{\varepsilon}{c_1}$ *and* $w \in$ *Index* **do**
17 **for** *each* $(v, \psi_\ell(v, w))$ *tuple in* $L_\ell(w)$ *in Index* **do**
18 $\hat{s}_I(u, v) \leftarrow \hat{s}_I(u, v) + \frac{\widehat{\eta\pi}_\ell(u,w)\psi_\ell(v,w)}{(1-\sqrt{c})^2}$;
19 **return** all non-zero $\hat{s}(u, v) \leftarrow \hat{s}_B(u, v) + \hat{s}_I(u, v)$;

Error Analysis. We now analyze the overall error bounds of the PRSim algorithm. Recall that given a source node u and a target node v, $s(u, v) = s_I(u, v) + s_B(u, v)$ where $s_I(u, v)$ and $s_B(u, v)$ are defined by equations (7) and (8), respectively. Algorithm 4 uses index to generate an estimator $\hat{s}_I(u, v)$ for each $s_I(u, v)$, $v \in V$, and uses backward walks to generate an estimator $\hat{s}_B(u, v)$ for each $s_B(u, v)$, $v \in V$. We have the following two lemmas that bound the errors of the two approximations.

Lemma 3.6. *Given a source node u, for any $v \in V$, Algorithm 4 provides an estimator $\hat{s}_I(u,v)$ for $s_I(u,v)$ such that:*

$$\Pr\left[|\hat{s}_I(u,v) - s_I(u,v)| > \frac{\varepsilon}{2}\right] \le \frac{\delta}{2n}. \tag{9}$$

Lemma 3.7. *Given a source node u, for any $v \in V$, Algorithm 4 provides an estimator $\hat{s}_B(u,v)$ for $s_B(u,v)$ such that:*

$$\Pr\left[|\hat{s}_B(u,v) - s_B(u,v)| > \frac{\varepsilon}{2}\right] \le \frac{\delta}{2n}. \tag{10}$$

Combining Lemmas 3.6 and 3.7 follows that

$$\Pr\left[|\hat{s}(u,v) - s(u,v)| > \varepsilon\right] \le \frac{\delta}{2n} + \frac{\delta}{2n} = \frac{\delta}{n}.$$

Applying union bound on n nodes follows Theorem 3.8.

Theorem 3.8. *PRSim answers single-source SimRank queries with additive error ε with probability at least $1-\delta$.*

Query Time Analysis for Worst-Case Graphs. We first analyze the query time of the PRSim algorithm on worst-case graphs. Given a node $u \in V$, let $C(u)$ denote the query cost of PRSim on u, and $C = \frac{1}{n}\sum_{u\in V} C(u)$ denote the average query cost. We divide $C(u)$ into three terms: $C(u) = C_F(u) + C_I(u) + C_B(u)$, where $C_F(u)$ denote the cost for computing $\widehat{\eta\pi}_\ell(u,w)$ from source node u, $C_I(u)$ denote the query cost for retrieving reserves $\psi_\ell(v,w)$ from the index, and $C_B(u)$ denote the query cost for estimating $\hat{\pi}_\ell(v,w)$ with backward walks. Let $C_F = \frac{1}{n}\sum_{u\in V} C_F(u)$, $C_I = \frac{1}{n}\sum_{u\in V} C_I(u)$ and $C_B = \frac{1}{n}\sum_{u\in V} C_B(u)$ denote the average query cost of $C_F(u)$, $C_I(u)$ and $C_B(u)$, respectively. We can express the expected average query cost of Algorithm 4 as $\mathrm{E}[C] = \mathrm{E}[C_F] + \mathrm{E}[C_I] + \mathrm{E}[C_B]$.

For $\mathrm{E}[C_F]$, recall that we generate a number $n_r = \Theta\left(\frac{\log\frac{n}{\delta}}{\varepsilon^2}\right)$ of $\sqrt{c}$-walks to estimate $\widehat{\eta\pi}_\ell(u,w)$. Since each $\sqrt{c}$-walk takes constant time, we have $C_F(u) = O\left(\frac{\log\frac{n}{\delta}}{\varepsilon^2}\right)$, and $\mathrm{E}[C_F] = O\left(\frac{\log\frac{n}{\delta}}{\varepsilon^2}\right)$. We have the following lemmas for $\mathrm{E}[C_I]$ and $\mathrm{E}[C_B]$.

Lemma 3.9. *Let $c_1 = \frac{12}{(1-\sqrt{c})^2}$ and C_I denote the average cost for querying the index. We have*

$$\mathrm{E}[C_I] = O\left(\min\left\{\frac{n}{\varepsilon}\sum_{j=1}^{\frac{c_1}{\varepsilon}}\pi(w_j), \frac{n}{\varepsilon^2}\sum_{j=1}^{j_0}\pi(w_j)^2\right\}\right).$$

Lemma 3.10. *Let C_B denote the average cost for performing Variance Bounded Backward Walks. We have $\mathrm{E}[C_B] = O\left(\frac{n\log\frac{n}{\delta}}{\varepsilon^2}\sum_{j=j_0+1}^{n}\pi(w_j)^2\right)$.*

By Lemma 3.9, we have $\mathrm{E}[C_I] \le O\left(\frac{n\log\frac{n}{\delta}}{\varepsilon^2}\sum_{j=j_0+1}^{n}\pi(w_j)^2\right)$. Combining with Lemma 3.10 follows Theorem 3.11.

Theorem 3.11. *Suppose the query node u is uniformly chosen from V. The expected query cost of PRSim on worst-case graphs is bounded by*

$$\mathrm{E}[C] = O\left(\frac{n\log\frac{n}{\delta}}{\varepsilon^2}\cdot\sum_{w\in V}\pi(w)^2\right). \tag{11}$$

Query Time Analysis for Power-Law Graphs. Recall that on a power-law graph, the fractions $P_o(k)$ and $P_i(k)$ of nodes with out- and in-degree at least k satisfy that $P_o(k) \sim k^{-\gamma}$ and $P_i(k) \sim k^{-\gamma'}$ [7], where γ and γ' are the cumulative power-law exponents that usually take values from 1 to 3. It is shown in [5, 27, 35] that the PageRank of a power-law graph also follows power-law with same exponent γ' as the *in-degree distribution*. Thus, the reverse PageRank follows the same power-law distribution as the out-degree distribution. In particular, let $P_\pi(x)$ denote the portion of nodes with reverse PageRank value at least x, then $P_\pi(x) \sim x^{-\gamma}$.

Now consider the following alternating statement of the above power-law distribution: let $w_1, \ldots, w_n$ denote the nodes in the graph sorted in descending order of their reverse PageRank values, that is, $\pi(w_1) \ge \pi(w_2) \ge \ldots \ge \pi(w_n)$. We have that the j-th largest reverse PageRank value $\pi(w_j)$ is proportional to $j^{-\beta}$. Here β is the power-law exponent that takes value from $(0,1)$. This assumption has been widely adopted in the literature of PageRank computations [5, 27, 35]. To understand the relation between two exponents γ and β, note that there are j nodes with reverse PageRank value at least $x = \frac{\kappa j^{-\beta}}{n^{1-\beta}}$, and thus we have $j \sim \left(\frac{j^{-\beta}}{n^{1-\beta}}\right)^{-\gamma} \sim j^{\beta\cdot\gamma}$. It follows that $\beta = \frac{1}{\gamma}$. Therefore, for power-law graphs, we have

$$\pi(w_j) = \kappa\cdot j^{-\beta}/n^{1-\beta} = \kappa\cdot j^{-\frac{1}{\gamma}}/n^{1-\frac{1}{\gamma}}, \tag{12}$$

where κ is a normalization constant such that $\kappa\sum_{j=1}^{n}\frac{j^{-\frac{1}{\gamma}}}{n^{1-\frac{1}{\gamma}}} = 1$. Combing equation (12) and Lemma 3.2, the index size is bounded by $O\left(\frac{n}{\varepsilon}\sum_{j=1}^{j_0}\frac{j^{-\frac{1}{\gamma}}}{n^{1-\frac{1}{\gamma}}}\right) = O\left(\frac{n}{\varepsilon}\cdot\frac{j^{1-\frac{1}{\gamma}}}{n^{1-\frac{1}{\gamma}}}\right) = O\left(\frac{n^{\frac{1}{\gamma}}j_0^{1-\frac{1}{\gamma}}}{\varepsilon}\right)$. Here we use the property of Riemann zeta function (see Lemma A.4). By setting $j_0 = n(\varepsilon\bar{d})^{\frac{\gamma}{\gamma-1}}$, we have index size is bounded by $O\left(\frac{n^{\frac{1}{\gamma}}n^{1-\frac{1}{\gamma}}\varepsilon\bar{d}}{\varepsilon}\right) = O(m)$. Plugging $\pi(w_j) = \kappa\cdot\frac{j^{-\frac{1}{\gamma}}}{n^{1-\frac{1}{\gamma}}}$ and $j_0 = n(\varepsilon\bar{d})^{\frac{\gamma}{\gamma-1}}$ into Lemma 3.10 and Lemma 3.9, and we have the following theorem.

Theorem 3.12. *Assume that the out-degree distribution of the graph follows power-law distribution with exponent $\gamma \ge 1$, and let $\varepsilon \ge \log^{\frac{\gamma-1}{2-\gamma}} n/(n^{\frac{\gamma-1}{\gamma}}\bar{d}^{2-\gamma})$, $\delta > 1/n^{\Omega(1)}$. Suppose the query node u is uniformly chosen from V. By setting $j_0 =$*

$n(\varepsilon\bar{d})^{\frac{\gamma}{\gamma-1}}$, *the expected cost of Algorithm 4 is bounded by*

$$\mathrm{E}[C] = \begin{cases} O(\frac{1}{\varepsilon^2}\log\frac{n}{\delta}), & \text{for } \gamma > 1/2; \\ O(\frac{1}{\varepsilon^2}\log\frac{n}{\delta}\log n), & \text{for } \gamma = 1/2; \\ O\left(\min\left\{\frac{n^{\frac{1}{\gamma}}}{\varepsilon^{2-\frac{1}{\gamma}}}, \frac{n^{\frac{2}{\gamma}-1}}{\varepsilon^2}\right\}\right), & \text{for } 1 < \gamma < 2. \end{cases} \quad (13)$$

The size of the index generated by Algorithm 1 is bounded by $O(m)$. *The preprocessing time is bounded by* $O\left(\frac{m}{\varepsilon}\right)$.

Dynamic Graphs. Our algorithm is able to support dynamic graphs where edges may be inserted or deleted. Recall that PRSim generates the index by performing the backward search algorithm. It is shown in [44] that the results of the backward search to a randomly selected target node w can be maintained with cost $O(k + \frac{\bar{d}}{\varepsilon})$, where k is the total number of insertions/deletions. Since our index stores the results of the backward search for j_0 target nodes, it can process k insertions/deletions in $O(kj_0 + \frac{m}{\varepsilon})$ time. Therefore, the per-update-cost for processing k updates is bounded by $O(j_0 + \frac{m}{\varepsilon k})$. However, a thorough investigation of this issue is beyond the scope of our paper.

4 RELATED WORK

In what follows, we briefly review some of the state-of-the-art solutions for SimRank computation. We exclude SLING [32], which we have discussed in Section 2.

Monte Carlo and READS. Based on the $\sqrt{c}$-walk interpretation, we can use the following Monte Carlo algorithm [12, 32] to estimate the SimRank value $s(u, v)$: we generate n_r pairs of $\sqrt{c}$-walks from u and v, and use the percentage of $\sqrt{c}$-walks that meet as an estimation of $s(u, v)$. Using concentration inequality, one can show that by setting $n_r = \Theta\left(\frac{\log\frac{n}{\delta}}{\varepsilon^2}\right)$, the Monte Carlo algorithm estimates $s(u, v)$ with an additive error ε with probability at least $1 - \delta$. For a single-source query on node u, we can generate n_r walks from each node $v \in V$ and estimate $s(u, v)$ with additive error ε. The query cost is $O\left(\frac{n\log\frac{n}{\delta}}{\varepsilon^2}\right)$, which is inefficient on large graphs.

A recent work proposes the READS algorithm [16] based on the Monte Carlo approach. READS pre-computes the $\sqrt{c}$-walks from each node, and compresses the $\sqrt{c}$-walks by merging them into trees. Given a query node u, READS retrieves the $\sqrt{c}$-walks starting from u, finds all $\sqrt{c}$-walks that meet with u's $\sqrt{c}$-walks, and then updates the SimRank estimator for each v related to these $\sqrt{c}$-walks. Several optimization techniques were adopted to improve the query efficiency of READS. The major issue of READS is that it requires generating and storing a large number of $\sqrt{c}$-walks from each node in the preprocessing phase. The query cost also remains $O(n\log\frac{n}{\delta}/\varepsilon^2)$, which is the same as that of the classic Monte Carlo algorithm.

ProbeSim. ProbeSim [25] is an index-free algorithm that computes single-source and top-k SimRank queries on large graphs. Given a query node u, the ProbeSim algorithm samples a $\sqrt{c}$-walk $\mathcal{W}(u)$ from u. For a node w visited by $\mathcal{W}(u)$ at the ℓ-th step, the algorithm performs a *Probe* procedure that computes the probability of an $\sqrt{c}$-walk from each node v visiting w at the ℓ-th step. To rule out the probability that a pair of ℓ-walks may meet multiple times, the Probe algorithm avoids the nodes previously visited by $\mathcal{W}(u)$. It is shown in [25] that the ProbeSim algorithm gives an unbiased estimator for the SimRank values $s(u, v), v \in V$. Therefore, by repeating the sampling procedure $O(\log\frac{n}{\delta}/\varepsilon^2)$ times, ProbeSim answers single-source SimRank queries with probability at least $1 - \delta$.

There are two subtle problems with ProbeSim. First, to avoid multiple meeting nodes, the Probe from node w has to avoid the nodes on $\mathcal{W}(u)$, which means it is impossible to pre-compute the Probe results to speed up the query time. Second, as we will show later, the probability that a node w in the graph is visited by the $\sqrt{c}$-walk from u is proportional to $\pi(w)$, the reverse PageRank of w. On the other hand, the complexity of the Probe algorithm on w is also proportional to $\pi(w)$. This essentially means it is likely that a *hub node* with high reverse PageRank value is visited by the $\sqrt{c}$-walk from u, and it will incur significant cost in the Probe phase. Finally, the algorithm also requires $O(n\log\frac{n}{\delta}/\varepsilon^2)$ query cost to answer a single-source query.

TSF. TSF [30] is a two-stage random-walk sampling algorithm for single-source and top-k SimRank queries on dynamic graphs. Given a parameter R_g, TSF starts by building R_g *one-way graphs* as an index structure. Each one-way graph is constructed by uniformly sampling *one* in-neighbor from each vertex's in-coming edges. The one-way graphs are then used to simulate random walks during query processing. To achieve high efficiency, TSF allows two $\sqrt{c}$-walks to meet multiple times, and thus overestimate the actual SimRank values. Furthermore, TSF assumes that every random walk would not contain any cycle, which does not hold in practice.

Other Related Work. *Power method* [15] is the classic algorithm that computes all-pair SimRank similarities for a given graph. Let S be the SimRank matrix such that $S_{ij} = s(i, j)$, and A be the transition matrix of G. Power method recursively computes the SimRank Matrix S using the following formula [18]

$$S = (cA^\top SA) \vee I, \quad (14)$$

where $\vee$ is the element-wise maximum operator. Several follow-up works [26, 40, 43] improve the efficiency or effectiveness of the power method in terms of either efficiency or accuracy. However, these methods still incur $O(n^2)$ space overheads, as there are $O(n^2)$ pairs of nodes in the graph. A recent work [34] reduces the cost to $O(NNZ)$, where NNZ

is the number of node pairs with large SimRank similarities. However, as shown in [34], there are still a constant fraction of $O(n^2)$ node pairs with large SimRank similarities, so the worst case complexity remains $O(n^2)$.

Motivated by difficulty in dealing with the element-wise maximum operator $\vee$ in Equation 14, some existing work [13, 14, 18, 21, 38, 39, 41] consider the following alternative formula for SimRank:

$$S = cA^\top SA + (1-c) \cdot I. \quad (15)$$

However, it is shown that the similarities calculated by this formula are different from SimRank [18].

For single-source queries, Fogaras and Rácz [12] propose a Monte Carlo algorithm that uses random walks to approximate SimRank values. Maehara et al. [28] propose an index structure for top-k SimRank queries, but it relies on heuristic assumptions about G, and hence, does not provide any worst-case error guarantee. Li et al. [22] propose a distributed version of the Monte Carlo approach in [12], but it achieves scalability at the cost of significant computation resources. Finally, there is existing work on variants of SimRank [4, 11, 42, 48] and on various graph applications [6, 19, 37], but the proposed solutions are inapplicable for top-k and single-source SimRank queries.

5 EXPERIMENTS

This section experimentally evaluates the proposed solutions against the state of the art. All experiments are conducted on a machine with a Xeon(R) CPU E7-4809@2.10GHz CPU and 196GB memory.

5.1 Experimental Settings

Methods. We compare PRSim against five SimRank algorithms: READS [16], SLING [32], TSF [30], ProbeSim [25] and TopSim [20]. As mentioned in Section 4, READS, SLING and TSF are the state-of-the-art index-based methods, and ProbeSim and TopSim are the state-of-the-art index-free methods.

Ground Truth for single-pair queries. Given a pair of nodes u and v, we use the Monte Carlo algorithm to estimate $s(u,v)$ with high precisions, and then use the result as the ground truth for $s(u,v)$. In particular, we set the parameters of the Monte Carlo algorithm such that it incurs an error less than 0.00001 with confidence over 99.999%.

Pooling. We extend the *pooling* idea [25] to evaluate the effectiveness of the single-source algorithms on large graphs. Given a source node u, we run each single-source algorithm, order the nodes according to their estimated SimRank values, and retrieve the top-k nodes. We merge the top-k nodes returned by each algorithm, remove the duplicates, and put them into a *pool*. As such, if we were to evaluate ℓ algorithms, then the pool size is between k and ℓk. For each node v in the pool, we obtain the ground truth of $s(u,v)$ using the Monte Carlo algorithm, and retrieve $V_k = \{v_1, \ldots, v_k\}$, namely, the k nodes with the highest SimRank values from the pool.

Table 3: Data Sets.

Data Set	Type	n	m
DBLP-Author (DB)	undirected	5,425,963	17,298,033
LiveJournal (LJ)	directed	4,847,571	68,993,773
It-2004 (IT)	directed	41,291,594	1,150,725,436
Twitter (TW)	directed	41,652,230	1,468,365,182
UK-Union (UK)	directed	133,633,040	5,507,679,822

Metrics. To evaluate the absolute error of single-source SimRank algorithms, we calculate the average absolute errors for approximating $s(u,v_i)$ for each v_i in the pool. More precisely, for each $v_i \in V_k$ returned by the pool, let $\hat{s}(u,v_i)$ be the estimator for $s(u,v_i)$ returned by the algorithm to be evaluated. We set

$$AvgError@k = \frac{1}{k} \sum_{1 \le i \le k} |\hat{s}(u,v_i) - s(u,v_i)|.$$

To evaluate the algorithms' abilities to return the top-k results, we use $V_k = \{v_1, \ldots, v_k\}$ as the ground truth for the top-k nodes. Note that these nodes are the best possible results that can be returned by any of the algorithms to be evaluated. Let $V'_k = \{v'_1, \ldots, v'_k\}$ denote the top-k node set returned by the algorithm to be evaluated. Note that *Precision@k* evaluates how many correct (or best possible) nodes are included in V'_k.

5.2 Experiments on Real-World Graphs

We evaluate the tradeoffs between accuracy and complexity for each algorithm on real world graphs. We use 5 data sets, as shown in Table 3. All data sets are obtained from public sources [1, 2].

Parameters. SLING [32] has a parameter ε_a, the upper bound on the absolute error. We vary ε_a in $\{0.5, 0.1, 0.05, 0.01, 0.005\}$, where $\varepsilon_a = 0.05$ is the default value in [32]. TSF has two parameters R_g and R_q, where R_g is the number of one-way graphs stored in the index, and R_q is the number of times each one-way graph is reused in the query stage. We vary (R_g, R_q) in $\{(10,2), (100,20), (200,30), (300,40), (600,80)\}$, where [30] sets $(R_g, R_q) = (300, 40)$ by default. TopSim has four internal parameters T, h, η and H, where T is the depth of the random walks, $1/h$ is the minimal degree threshold used to identify a high degree node, η is the similarity threshold for trimming a random walk, and H is the number of random walks to be expanded at each level. We fix H and η to their default values 100 and 0.001, and vary $(T, 1/h)$ in $\{(1,10), (3,100), (3,1000), (3,10000), (4,10000)\}$. Note that [20] sets $(T, 1/h) = (3, 100)$ by default. The READS paper [16] proposed three algorithms: READS, READS-D, and

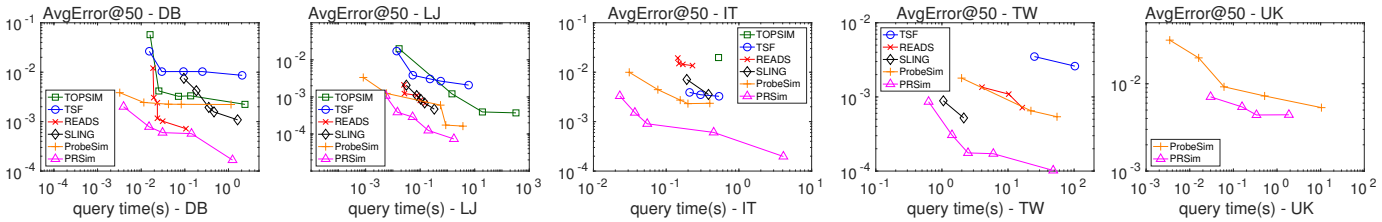


Figure 2: AvgError@50 v.s. *Query time*

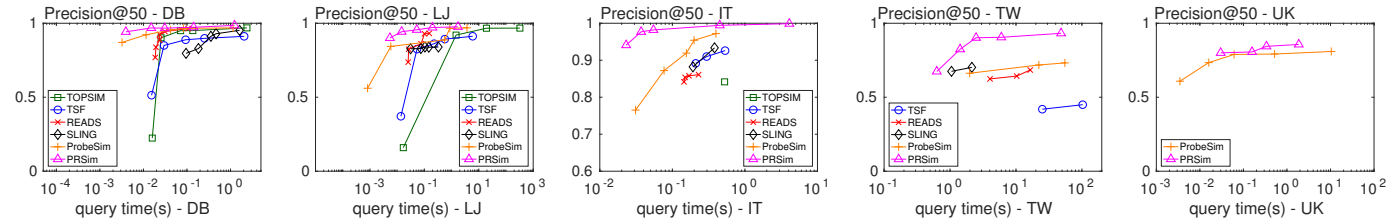


Figure 3: Precision@50 v.s. *Query time*

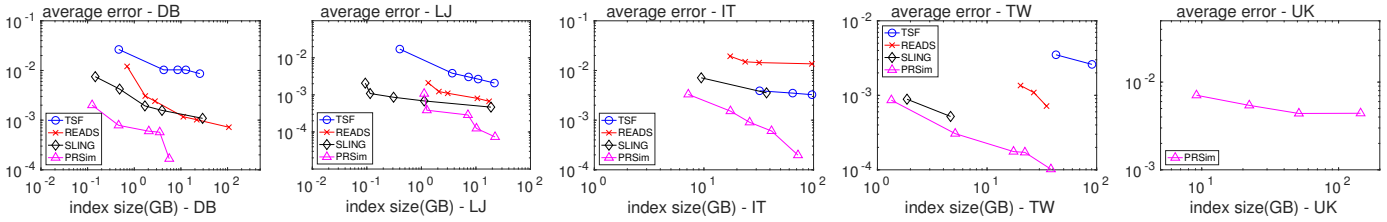


Figure 4: AvgError@50 v.s. *Index size*

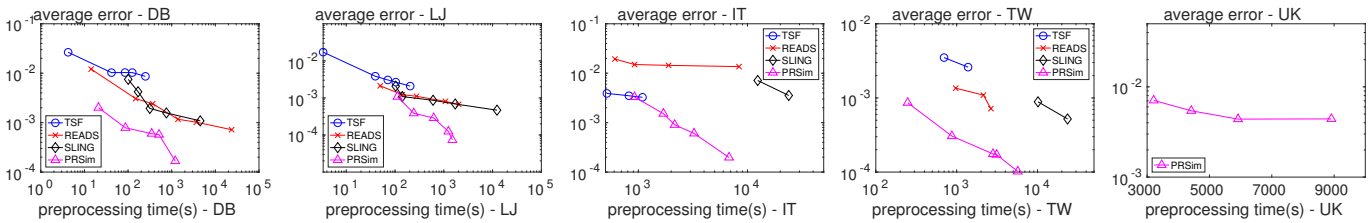
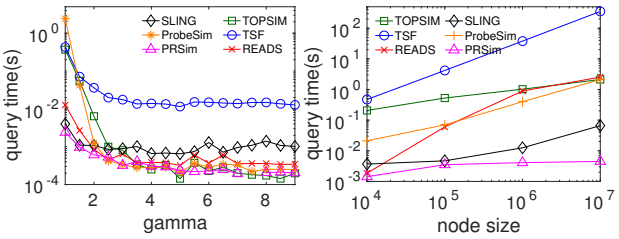


Figure 5: AvgError@50 v.s. *Preprocessing time*

READS-Rq. We only include the static version of READS in our experiments, as it is the fastest among the three [16]. READS has two parameters r and t, where r is the number of $\sqrt{c}$-walks generated for each node in the preprocessing stage and t is the maximum depth of the $\sqrt{c}$-walks. We vary (r, t) in $\{(10, 2), (50, 5), (100, 10), (500, 10), (1000, 20)\}$, where (r, t) $= (100, 10)$ is the default setting in [16]. For ProbeSim [25], we vary the error parameter ε_a in $\{0.5, 0.1, 0.05, 0.01, 0.005\}$, where $\varepsilon_a = 0.1$ is the default setting in [25]. For PRSim, we vary ε in $\{0.5, 0.1, 0.05, 0.01, 0.005\}$. We also set j_0 to $\sqrt{n}$ so that the index size of PRSim increases with $1/\varepsilon$. We fix the failure probability $\delta = 0.0001$ unless otherwise specified. We set the decay factor c of SimRank to 0.6, following previous work [26, 28, 39, 41, 42].

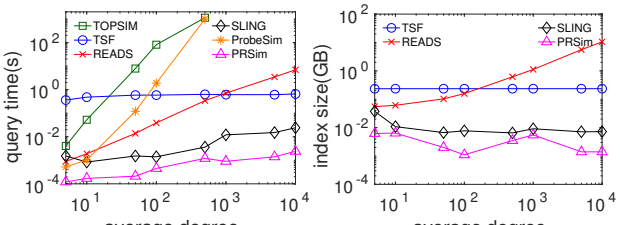
Experimental results. On each data set, we issue 100 single-source queries and 100 top-50 queries for each algorithm and each parameter set, and record the averages of the query time, index sizes, preprocessing time, *AvgError@50* and *Precision@50*. For each algorithm and each dataset, we omit a parameter set if it runs out of 196GB memory or takes over 10 hours to finish queries or preprocessing on that data set.

Figures 2, 3 show the tradeoffs between *AvgError@50* and the query time and the tradeoffs between *Precision@50* and the query time. The overall observation is that PRSim outperforms all competitors by achieving lower errors and higher precisions with less query time on all datasets. Most notably, on the TW dataset, PRSim achieves a *Precision@50* of 92% using a query time of 5 seconds, while the closest competitor, ProbeSim, achieves a precision around 75% using over



(a) *Query time* v.s. γ. (b) *Query time* v.s. n.

Figure 6: Results on power-law graphs.



(a) *Query time* v.s. $\bar{d}$. (b) *Index Size* v.s. $\bar{d}$.

Figure 7: Results on non-power-law graphs.

50 seconds. Furthermore, on the 5-billion-edge UK data set, PRSim is the only two index-based algorithms that are able to finish preprocessing and queries, which demonstrates the scalability of our algorithms. We also note that the query time of SLING and READS are not sensitive to the choices of parameters. This is as expected, since the majority of their query cost is spent on reading the index, which is a cache-friendly task. After observing the skewed trend of READS on DB in Figure 2, we decide to evaluate an extra parameter set $(r, t) = (5000, 20)$ to see if READS can outperform PRSim in terms of query-time-error tradeoff, given significantly more indexing space. The result shows that PRSim still achieves better accuracy with less query time.

Figure 4, and 5 show the tradeoffs between*AvgError@50* and the index size and the tradeoffs between *AvgError@50* and the preprocessing time, respectively. Again, our algorithm manages to outperform all index-based algorithms (SLING, TSF, READS) by achieving a lower error with less index size and preprocessing time. In particular, on the DB dataset, our algorithm is able to achieve an average error of 10^{-3} using an index of size $200MB$, while the closest competitor READS needs $100GB$.

5.3 Experiments on Synthetic Data Sets

We now evaluate PRSim and the competitors with fixed parameters on synthetic datasets with varying network structure and sizes. We set $\varepsilon_a = 0.25$ for SLING, $R_g = 300$ and $R_q = 40$ for TSF, $T = 3$, $1/h = 100$, $\eta = 0.001$, and $H = 100$ for TopSim, $\varepsilon_a = 0.25$ for ProbeSim, $r = 100$ and $t = 10$ for READS, and $\varepsilon = 0.25$ for PRSim. We fix the failure probability $\delta = 0.001$ unless otherwise specified. On each data set, we issue 100 single-source queries with each algorithm to be evaluated, and report the corresponding measures.

Hardness of SimRank computation and degree distributions. We first investigate the relation between the hardness of SimRank computation and degree distributions. We generate a set of undirected power-law graphs with various power-law exponents using the hyperbolic graph generator [3]. In particular, we fix the number of nodes n to be $100,000$ and the average degree $\bar{d}$ to be 10, and vary the degree power-law exponent γ from 1 to 9. Figure 6(a) reports the average query time of each algorithm. Recall that the theoretical analysis of PRSim suggests that its query time increases with $1/\gamma$. Figure 6(a) concurs with this analysis. In fact, we observe that the query time of all algorithms follows a similar distribution as the function $y = 1/\gamma$ on the log-log plot: the query time decreases as we increase γ from 1 to 4, and becomes stable after $\gamma > 4$. Based on this observation and on the theoretical analysis for PRSim, we make the following conjuncture:

Conjuncture 1. *The hardness of SimRank computation is correlated to the reciprocal of the power-law exponent γ of the out-degree distribution.*

Scalability analysis. To evaluate the scalability of our algorithm, we generate synthetic power-law graphs by fixing the exponent $\gamma = 3$ and average degree $\bar{d} = 10$, and vary the graph size n from 10^4 to 10^7. Figure 6(b) shows the running time of PRSim on these graphs. The results show that the running time of PRSim forms a concave curve in a log-log plot, which proves the sub-linearity of PRSim.

Experiments on non-power-law Graphs. We generate random graphs using the Erős and Rényi (ER) model, where we assign an edge to each node pair with a user-specified probability p. We fix the number of nodes to $n = 10,000$ and set the value of p so that the average degree $\bar{d}$ of each graph varies from 5 to $10,000$. Figure 7 shows the query time of each algorithm on these synthetic graphs. We observe that the query performance of ProbeSim degrades dramatically as we increase $\bar{d}$. On the other hand, PRSim is able to answer queries on very dense graphs efficiently. We attribute this quality to the fact that the Randomized Probe algorithm in ProbeSim always goes through all out-neighbors of a target node, while our Variance Bounded Backward Walk algorithm only needs to visit a fraction of the out-neighbors.

6 CONCLUSIONS

This paper presents PRSim, an algorithm for single-source SimRank queries. PRSim connects the time complexity of SimRank computation with the distribution of the reverse PageRank, and achieves sublinear query time on power-law graphs with small index size. Our experiments show that the algorithm significantly outperforms the existing methods in terms of query time, accuracy, index size and scalability.

7 ACKNOWLEDGEMENTS

This research was supported in part by National Natural Science Foundation of China (No. 61832017 and No. 61732014), by MOE, Singapore under grant MOE2015-T2-2-069, and by NUS, Singapore under an SUG. Sibo Wang was supported by CUHK Direct Grant No. 4055114. He was also supported by the CUHK University Startup Grant No. 4930911 and No. 5501570.

REFERENCES

[1] http://snap.stanford.edu/data/index.html.
[2] http://law.di.unimi.it/datasets.php.
[3] Rodrigo Aldecoa, Chiara Orsini, and Dmitri Krioukov. Hyperbolic graph generator. *Computer Physics Communications*, 196:492–496, 2015.
[4] Ioannis Antonellis, Hector Garcia Molina, and Chi Chao Chang. Simrank++: query rewriting through link analysis of the click graph. *PVLDB*, 1(1):408–421, 2008.
[5] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *VLDB*, 4(3):173–184, 2010.
[6] Mansurul Bhuiyan and Mohammad Al Hasan. Representing graphs as bag of vertices and partitions for graph classification. *Data Science and Engineering*, 3(2):150–165, 2018.
[7] Béla Bollobás, Christian Borgs, Jennifer T. Chayes, and Oliver Riordan. Directed scale-free graphs. In *SODA*, pages 132–139, 2003.
[8] Pawel Brach, Marek Cygan, Jakub Lkacki, and Piotr Sankowski. Algorithmic complexity of power law networks. In *SODA*, pages 1306–1325. Society for Industrial and Applied Mathematics, 2016.
[9] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *ICALP*, pages 693–703. Springer, 2002.
[10] Fan R. K. Chung and Lincoln Lu. Concentration inequalities and martingale inequalities: A survey. *Internet Mathematics*, 3(1):79–127, 2006.
[11] Dániel Fogaras and Balázs Rácz. Scaling link-based similarity search. In *WWW*, pages 641–650, 2005.
[12] 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.
[13] Yuichiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, and Makoto Onizuka. Efficient search algorithm for simrank. In *ICDE*, pages 589–600, 2013.
[14] Guoming He, Haijun Feng, Cuiping Li, and Hong Chen. Parallel simrank computation on large graphs with iterative aggregation. In *KDD*, pages 543–552, 2010.
[15] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *SIGKDD*, pages 538–543, 2002.
[16] 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.
[17] Ruoming Jin, Victor E Lee, and Hui Hong. Axiomatic ranking of network role similarity. In *KDD*, pages 922–930, 2011.
[18] Mitsuru Kusumoto, Takanori Maehara, and Ken-ichi Kawarabayashi. Scalable similarity search for simrank. In *SIGMOD*, pages 325–336, 2014.
[19] JooYoung Lee and Rustam Tukhvatov. Evaluations of similarity measures on vk for link prediction. *Data Science and Engineering*, 3(3):277–289, 2018.
[20] Pei Lee, Laks V. S. Lakshmanan, and Jeffrey Xu Yu. On top-k structural similarity search. In *ICDE*, pages 774–785, 2012.
[21] Cuiping Li, Jiawei Han, Guoming He, Xin Jin, Yizhou Sun, Yintao Yu, and Tianyi Wu. Fast computation of simrank for static and dynamic information networks. In *EDBT*, pages 465–476, 2010.
[22] 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.
[23] David Liben-Nowell and Jon M. Kleinberg. The link-prediction problem for social networks. *JASIST*, 58(7):1019–1031, 2007.
[24] Yu Liu, Jiaheng Lu, Hua Yang, Xiaokui Xiao, and Zhewei Wei. Towards maximum independent sets on massive graphs. *VLDB*, 8(13):2122–2133, 2015.
[25] 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.
[26] Dmitry Lizorkin, Pavel Velikhov, Maxim N. Grinev, and Denis Turdakov. Accuracy estimate and optimization techniques for simrank computation. *VLDB J.*, 19(1):45–66, 2010.
[27] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. Personalized pagerank estimation and search: A bidirectional approach. In *WSDM*, pages 163–172, 2016.
[28] Takanori Maehara, Mitsuru Kusumoto, and Ken-ichi Kawarabayashi. Efficient simrank computation via linearization. *CoRR*, abs/1411.7228, 2014.
[29] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.
[30] 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.
[31] Nikita Spirin and Jiawei Han. Survey on web spam detection: principles and algorithms. *SIGKDD Explorations*, 13(2):50–64, 2011.
[32] Boyu Tian and Xiaokui Xiao. SLING: A near-optimal index structure for simrank. In *SIGMOD*, pages 1859–1874, 2016.
[33] Sibo Wang and Yufei Tao. Efficient algorithms for finding approximate heavy hitters in personalized pageranks. In *SIGMOD*, pages 1113–1127. ACM, 2018.
[34] Yue Wang, Xiang Lian, and Lei Chen. Efficient simrank tracking in dynamic graphs. In *ICDE*, pages 545–556. IEEE, 2018.
[35] 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.
[36] Wensi Xi, Edward A Fox, Weiguo Fan, Benyu Zhang, Zheng Chen, Jun Yan, and Dong Zhuang. Simfusion: measuring similarity using unified relationship matrix. In *SIGIR*, pages 130–137. ACM, 2005.
[37] Qi Ye, Changlei Zhu, Gang Li, Zhimin Liu, and Feng Wang. Using node identifiers and community prior for graph-based classification. *Data Science and Engineering*, 3(1):68–83, 2018.
[38] Weiren Yu, Xuemin Lin, and Wenjie Zhang. Fast incremental simrank on link-evolving graphs. In *ICDE*, pages 304–315, 2014.
[39] Weiren Yu, Xuemin Lin, Wenjie Zhang, Lijun Chang, and Jian Pei. More is simpler: Effectively and efficiently assessing node-pair similarities based on hyperlinks. *PVLDB*, 7(1):13–24, 2013.
[40] Weiren Yu and Julie McCann. Gauging correct relative rankings for similarity search. In *CIKM*, pages 1791–1794, 2015.
[41] Weiren Yu and Julie A. McCann. Efficient partial-pairs simrank search for large networks. *PVLDB*, 8(5):569–580, 2015.
[42] Weiren Yu and Julie Ann McCann. High quality graph-based similarity search. In *SIGIR*, pages 83–92, 2015.
[43] Weiren Yu, Wenjie Zhang, Xuemin Lin, Qing Zhang, and Jiajin Le. A space and time efficient algorithm for simrank computation. *World Wide Web*, 15(3):327–353, 2012.

[44] Hongyang Zhang, Peter Lofgren, and Ashish Goel. Approximate personalized pagerank on dynamic graphs. In *KDD*, pages 1315–1324, 2016.
[45] Jing Zhang, Jie Tang, Cong Ma, Hanghang Tong, Yu Jing, and Juanzi Li. Panther: Fast top-k similarity search on large networks. In *SIGKDD*, pages 1445–1454. ACM, 2015.
[46] Zhipeng Zhang, Yingxia Shao, Bin Cui, and Ce Zhang. An experimental evaluation of simrank-based similarity search algorithms. *PVLDB*, 10(5):601–612, 2017.
[47] Peixiang Zhao, Jiawei Han, and Yizhou Sun. P-rank: a comprehensive structural similarity measure over information networks. In *CIKM*, pages 553–562. ACM, 2009.
[48] Peixiang Zhao, Jiawei Han, and Yizhou Sun. P-rank: a comprehensive structural similarity measure over information networks. In *CIKM*, pages 553–562, 2009.

A INEQUALITIES

A.1 Chernoff Bound

Lemma A.1 (Chernoff Bound [10]). *For a set $\{x_i\}$ ($i \in [1, n_r]$) of i.i.d. random variables with mean μ and $x_i \in [0,1]$,* $\Pr\left[\left|\frac{1}{n_r}\sum_{i=1}^{n_x} x_i - \mu\right| \geq \varepsilon\right] \leq \exp\left(-\frac{n_r \cdot \varepsilon^2}{\frac{2}{3}\varepsilon + 2\mu}\right)$.

A.2 Chebyshev's Inequality

Lemma A.2 (Chebyshev's inequality). *Let X be a random variable, then* $\Pr[|X - E[X]| \geq \varepsilon] \leq \frac{\mathrm{Var}[X]}{\varepsilon^2}$.

A.3 Median Trick

Lemma A.3 ([9]). *Let $X_1, \ldots, X_k$ be $k \geq 3\log\frac{1}{\delta}$ i.i.d. random variables, such that* $\Pr[|X_i - E[X_i]| \geq \varepsilon] \leq \frac{1}{3}$. *Let $X = Median_{1\leq i\leq k} X_i$, then* $\Pr[|X - E[X]| \geq \varepsilon] \leq \delta$.

A.4 Partial sum of Riemann zeta function

Lemma A.4. *The partial sum of Riemann zeta function satisfies the following property:*

$$\sum_{k=i+1}^{j} k^{-\alpha} = \begin{cases} O(j^{1-\alpha}), & \text{for } \alpha < 1; \\ O(\log j - \log i), & \text{for } \alpha = 1; \\ O\left(i^{1-\alpha}\right), & \text{for } \alpha > 1. \end{cases} \quad (16)$$

B PROOFS

B.1 Proof of Lemma 3.2

Proof. Let $w_1, \ldots, w_n$ be the nodes of the graph sorted in descending order of the reverse PageRank value $\pi(w_j)$. Let $size(w_j)$ denote index size for node w_j. Then, $size = \sum_{j=1}^{j_0} size(w_j)$ is the total size of the index. For each w_j, recall that Algorithm 1 uses backward search to find node x and level ℓ with ℓ-hop RPPR $\pi_\ell(x, w) \geq \varepsilon$, and record the tuple $(x, \ell, \pi_\ell(x, w))$. Hence, the space usage $size(w_j)$ is bounded by the total number of pairs (x, ℓ) with ℓ-hop RPPR $\pi_\ell(x, w) \geq \varepsilon$, i.e., $size(w_j) \leq \sum_{\ell=0}^{\infty}\sum_{x\in V} I(\pi_\ell(x, w) \geq \varepsilon)$, where $I(\pi_\ell(x, w) \geq \varepsilon)$ is an indicating function such that $I(\pi_\ell(x, w) \geq \varepsilon) = 1$ if $\pi_\ell(x, w) \geq \varepsilon$ and $I(\pi_\ell(x, w) \geq \varepsilon) = 0$ otherwise. We observe that $I(\pi_\ell(x, w) \geq \varepsilon) \leq \frac{\pi_\ell(x,w)}{\varepsilon}$, and thus $size(w_j) \leq \sum_{\ell=0}^{\infty}\sum_{x\in V}\frac{\pi_\ell(x,w)}{\varepsilon} = \frac{n\pi(w_j)}{\varepsilon}$. □

B.2 Proof of Lemma 3.3

Notations. We begin by defining two types of random variables. Consider a node y at level $i+1$ and a node $x \in \mathcal{I}(y)$. For ease of presentation, we let A denote the set of $x \in \mathcal{I}(y)$ such that $\hat{\pi}(x, w) > d_{in}(y)(1 - \sqrt{c})$ and B denote the set of $x \in \mathcal{I}(y)$ such that $\hat{\pi}(x, w) \leq d_{in}(y)(1 - \sqrt{c})$. We use $R_i(x)$ to denote the random variable indicating that the random number $r_0 < \sqrt{c}$. For each $x \in B$, we define random variable $Z_i(x, y) = 1$ if random number $r \leq \frac{\hat{\pi}_i(x,w)}{d_{in}(y)(1-\sqrt{c})}$, and $Z_i(x, y) = 0$ otherwise. Recall that for a node $x \in A$, we increment $\hat{\pi}_{i+1}(y, w)$ by $\frac{\hat{\pi}_i(x,w)}{d_{in}(y)}$ if and only if $R_i(x) = 1$; for a node $x \in B$, we increment $\hat{\pi}_{i+1}(y, w)$ by $1 - \sqrt{c}$ if and only if $R_i(x) = 1$ and $Z_i(x, y) = 1$. We can express $\hat{\pi}_{i+1}(y, w)$ as

$$\hat{\pi}_{i+1}(y, w) = \sum_{x\in A} R_i(x)\frac{\hat{\pi}_i(x, w)}{d_{in}(y)} + \sum_{x\in B} R_i(x)Z_i(x, y)(1 - \sqrt{c}). \quad (17)$$

Proof of Lemma 3.3. We prove the lemma by induction. For the base case, we have $\hat{\pi}_0(w, w) = 1 - \sqrt{c} = \pi_0(w, w)$. Assume that $\mathrm{E}[\hat{\pi}_i(x, w)] = \pi_i(x, w)$ for any $x \in V$. For an node $y \in V$, we will show that $\mathrm{E}[\hat{\pi}_{i+1}(y, w)] = \pi_{i+1}(y, w)$. Conditioning on $\hat{\pi}_i(x, w)$ in equation (17) follows that

$$E\left[\hat{\pi}_{i+1}(y, w) \mid \hat{\pi}_i(x, w), x \in V\right] = \sum_{x\in A} E[R_i(x)]\frac{\hat{\pi}_i(x, w)}{d_{in}(y)} + \sum_{x\in B} E[R_i(x)Z_i(x, y)](1 - \sqrt{c}).$$

We have $E[R_i(x)] = \Pr[r_0 \leq \sqrt{c}] = \sqrt{c}$ and

$$E[Z_i(x, y)] = \Pr[r < \frac{\hat{\pi}_i(x, w)}{d_{in}(y)(1 - \sqrt{c})}] = \frac{\hat{\pi}_i(x, w)}{d_{in}(y)(1 - \sqrt{c})}.$$

Since $R_i(x)$ and $Z_i(x, y)$ are independent random variables, we have $E[R_i(x)Z_i(x, y)] = \frac{\sqrt{c}\hat{\pi}_i(x,w)}{d_{in}(y)(1-\sqrt{c})}$. It follows that

$$E\left[\hat{\pi}_{i+1}(y, w) \mid \hat{\pi}_i(x, w), x \in V\right] = \sum_{x\in A}\frac{\sqrt{c}\hat{\pi}_i(x, w)}{d_{in}(y)} + \sum_{x\in B}\frac{\sqrt{c}\hat{\pi}_i(x, w)(1 - \sqrt{c})}{d_{in}(y)(1 - \sqrt{c})} = \sum_{x\in\mathcal{I}(y)}\frac{\sqrt{c}\hat{\pi}_i(x, w)}{d_{in}(y)}.$$

By the induction hypothesis, we have $E[\hat{\pi}_i(x, w)] = \pi_i(x, w)$ for $x \in \mathcal{I}(y)$, and thus $E[\hat{\pi}_{i+1}(y, w)] = \sum_{x\in\mathcal{I}(y)}\frac{\sqrt{c}\pi_i(x,w)}{d_{in}(y)} = \pi_{i+1}(y, w)$, which proves the lemma. □

B.3 Proof of Lemma 3.4

Proof. Let $cost_{i+1}(y)$ denote the number of times that $\hat{\pi}_{i+1}(y, w)$ gets incremented at level $i+1$. Note that the total cost is bounded by $\sum_{i=0}^{\ell}\sum_{x\in V} cost_i(x)$. A key observation is that each increment performed by Algorithm 3 adds at least $1 - \sqrt{c}$ to $\hat{\pi}_{i+1}(y, w)$. To see this, note that Algorithm 3 increments $\hat{\pi}_{i+1}(y, w)$ by $\frac{\hat{\pi}_i(x,w)}{d_{in}(y)}$ only if $d_{in}(y) < \frac{\hat{\pi}_i(x,w)}{1-\sqrt{c}}$, or equivalently $\frac{\hat{\pi}_i(x,w)}{d_{in}(y)} > 1-\sqrt{c}$. Therefore the number of times that $\hat{\pi}_{i+1}(y, w)$ gets incremented is bounded by $\frac{\pi_{i+1}(y,w)}{(1-\sqrt{c})}$, and thus the total cost is bounded by

$$E\left[\sum_{i=0}^{\infty}\sum_{x\in V} cost_i(x)\right] = \frac{1}{1-\sqrt{c}}\sum_{i=0}^{\infty}\sum_{x\in V}\pi_i(x,w)) = O(n\pi(w)).$$

This proves the lemma. □

B.4 Proof of Lemma 3.5

Proof. We will prove $\mathrm{E}[\hat{\pi}_\ell(x,w)^2] \le \pi_\ell(x,w)$ by induction. For the base case, we have $E[\hat{\pi}_0(w,w)^2] = (1-\sqrt{c})^2 \le \pi_0(w,w)$. Assume that $\mathrm{E}[\hat{\pi}_i(x,w)^2] \le \pi_i(x,w)$ for any $x \in V$. For an node $y \in V$, we will show that $\mathrm{E}[\hat{\pi}_{i+1}(y,w)^2] \le \pi_{i+1}(y,w)$. Conditioning on $\hat{\pi}_i(x,w)$ for all $x \in V$

$$E\left[\hat{\pi}_{i+1}(y,w)^2 \mid \hat{\pi}_i(x,w), x\in V\right] = E\left[\left(\sum_{x\in A} R_i(x)\frac{\hat{\pi}_i(x,w)}{d_{in}(y)} + \sum_{x\in B} R_i(x)Z_i(x,y)(1-\sqrt{c})\right)^2\right]. \tag{18}$$

We expand equation (18) into 5 terms:

$$\begin{aligned}
&E\left[\hat{\pi}_{i+1}(y,w)^2 \mid \hat{\pi}_i(x,w), x\in V\right] = X_1+X_2+X_3+X_4+X_5\\
&= \sum_{x\in A} E\left[R_i(x)^2\right]\frac{\hat{\pi}_i(x,w)^2}{d_{in}(y)^2} + \sum_{x\in B} E\left[R_i(x)^2 Z_i(x,y)^2\right](1-\sqrt{c})^2\\
&\quad + \sum_{x_1\neq x_2\in A} E\left[R_i(x_1)R_i(x_2)\right]\frac{\hat{\pi}_i(x_1,w)\hat{\pi}_i(x_1,w)}{d_{in}(y)^2}\\
&\quad + \sum_{x_1\neq x_2\in B} E\left[R_i(x_1)Z_i(x_1,y)R_i(x_2)Z_i(x_2,y)\right]\cdot(1-\sqrt{c})^2\\
&\quad + \sum_{x_1\in A, x_2\in B} E\left[R_i(x_1)R_i(x_2)Z_i(x_2,y)\right]\frac{\hat{\pi}_i(x_1,w)}{d_{in}(y)}\cdot(1-\sqrt{c}).
\end{aligned}$$

We use X_1, X_2, X_3, X_4 and X_5 to denote these 5 terms, and calculate them individually. Since $E\left[R_i(x)^2\right] = E\left[R_i(x)\right] = \sqrt{c}$, we have $X_1 = \sum_{x\in A}\frac{\sqrt{c}\hat{\pi}_i(x,w)^2}{d_{in}(y)^2}$. Using the induction hypothesis, we have $\mathrm{E}[\hat{\pi}_i(x,w)^2] \le \pi(x,w)^2$, and thus

$$\mathrm{E}[X_1] \le \sum_{x\in A}\frac{\sqrt{c}\pi_i(x,w)}{d_{in}(y)^2} = \frac{1}{d_{in}(y)}\sum_{x\in A}\frac{\sqrt{c}\pi_i(x,w)}{d_{in}(y)} = \frac{S_A}{d_{in}(y)}, \tag{19}$$

where $S_A = \sum_{x\in A}\frac{\sqrt{c}\pi_i(x,w)}{d_{in}(y)}$. Since $\mathrm{E}[\hat{\pi}_i(x,w)] = \pi_i(x,w)$, and $E\left[R_i(x)^2 Z_i(x,y)^2\right] = \frac{\sqrt{c}\hat{\pi}_i(x,w)}{d_{in}(y)(1-\sqrt{c})}$, we have

$$\mathrm{E}[X_2] = (1-\sqrt{c})\sum_{x\in B}\frac{\sqrt{c}\pi_i(x,w)}{d_{in}(y)} = (1-\sqrt{c})S_B. \tag{20}$$

Here we define $S_B = \sum_{x\in B}\frac{\sqrt{c}\pi_i(x,w)}{d_{in}(y)}$. Note that $S_A + S_B = \sum_{x\in I(y)}\frac{\sqrt{c}\pi_i(x,w)}{d_{in}(y)} = \pi_{i+1}(y,w)$.

By the independence of $R_i(x_1), Z_i(x_1,y), R_i(x_2), Z_i(x_2,y)$ for $x_1 \neq x_2$, we have $X_3 = \sum_{x_1\neq x_2\in A}\frac{c\hat{\pi}_i(x_1,w)\hat{\pi}_i(x_2,w)}{d_{in}(y)^2}$, $X_4 = \sum_{x_1\neq x_2\in B}\frac{c\hat{\pi}_i(x_1,w)\hat{\pi}_i(x_2,w)}{d_{in}(y)^2}$, $X_5 = \sum_{x_1\in A, x_2\in B}\frac{c\hat{\pi}_i(x_1,w)\hat{\pi}_i(x_2,w)}{d_{in}(y)^2}$. Therefore, $X_3 + X_4 + X_5$ can be expressed as

$$X_3 + X_4 + X_5 = \sum_{x_1\neq x_2\in I(y)}\frac{c}{d_{in}(y)^2}\cdot\hat{\pi}_i(x_1,w)\hat{\pi}_i(x_2,w).$$

Using the inequality that $\hat{\pi}_i(x_1,w)\hat{\pi}_i(x_2,w) \le \frac{1}{2}\hat{\pi}_i(x_1,w)^2 + \frac{1}{2}\hat{\pi}_i(x_1,w)^2$, and we have

$$\begin{aligned}
X_3 + X_4 + X_5 &\le \sum_{x_1\neq x_2\in I(y)}\frac{c}{2d_{in}(y)^2}\left(\hat{\pi}_i(x_1,w)^2 + \hat{\pi}_i(x_2,w)^2\right)\\
&= \sum_{x\in I(y)}\frac{c\,(d_{in}(y)-1)}{d_{in}(y)^2}\hat{\pi}_i(x,w)^2.
\end{aligned}$$

The last equation is due to the fact that each $\hat{\pi}_i(x,w)^2$ appears exactly $d_{in}(y)-1$ times in the summation. By the induction hypothesis that $\mathrm{E}[\hat{\pi}_i(x,w)^2] \le \pi_i(x,w)$, we have

$$\begin{aligned}
\mathrm{E}[X_3+X_4+X_5] &\le \sqrt{c}\left(1-\frac{1}{d_{in}(y)}\right)\sum_{x\in I(y)}\frac{\sqrt{c}\pi_i(x,w)}{d_{in}(y)}\\
&= \sqrt{c}\left(1-\frac{1}{d_{in}(y)}\right)(S_A+S_B).
\end{aligned} \tag{21}$$

Combining Equations (19)-(21), it follows that

$$\begin{aligned}
&\mathrm{E}\left[\hat{\pi}_{i+1}(y,w)^2\right] \le \left(\sqrt{c}+\frac{1-\sqrt{c}}{d_{in}(y)}\right)S_A + \left(1-\frac{\sqrt{c}}{d_{in}(y)}\right)S_B\\
&= \left(1-\left(1-\sqrt{c}\right)\cdot\left(1-\frac{1}{d_{in}(y)}\right)\right)S_A + \left(1-\frac{\sqrt{c}}{d_{in}(y)}\right)S_B\\
&\le S_A + S_B = \pi_{i+1}(y,w).
\end{aligned}$$

And the lemma follows. □

B.5 Proof of Lemma 3.6

Proof. Recall that for $s_I(u,v)$,we have the estimator

$$\hat{s}_I(u,v) = \frac{1}{(1-\sqrt{c})^2}\sum_{\ell=0}^{\infty}\sum_{j=1}^{j_0}\widehat{\eta\pi}'_\ell(u,w_j)\psi_\ell(v,w_j),$$

where $\widehat{\eta\pi}'_\ell(u,w_j) = \widehat{\eta\pi}_\ell(u,w_j)$ if $\widehat{\eta\pi}_\ell(u,w_j) > \frac{(1-\sqrt{c})^2\varepsilon}{12}$ and $\widehat{\eta\pi}'_\ell(u,w_j) = 0$ if otherwise. $\widehat{\eta\pi}_\ell(u,w_j)$ is an estimator for $\eta(w_j)\pi_\ell(u,w_j)$ computed by Monte Carlo approach, and $\psi_\ell(v,w_j)$ is the reserve computed by ℓ-hop backward search. To bound the error of $\hat{s}_I(u,v)$, we further define

$$\hat{s}^1_I(u,v) = \frac{1}{(1-\sqrt{c})^2}\sum_{\ell=0}^{\infty}\sum_{j=1}^{j_0}\widehat{\eta\pi}_\ell(u,w_j)\psi_\ell(v,w_j),$$

and

$$\hat{s}^2_I(u,v) = \frac{1}{(1-\sqrt{c})^2}\sum_{\ell=0}^{\infty}\sum_{j=1}^{j_0}\widehat{\eta\pi}_\ell(u,w_j)\pi_\ell(v,w_j).$$

First, we claim that $\hat{s}_I(u,v)$ and $\hat{s}^1_I(u,v)$ differ by at most $\frac{\varepsilon}{6}$. More precisely, observe that $\widehat{\eta\pi}'_\ell(u,w)$ and $\widehat{\eta\pi}_\ell(u,w)$ differ by at most $\frac{(1-\sqrt{c})^2\varepsilon}{6}$, and thus

$$\begin{aligned}
\left|\hat{s}_I(u,v)-\hat{s}^1_I(u,v)\right| &= \sum_{\ell=0}^{\infty}\sum_{j=1}^{j_0}\frac{\left|\widehat{\eta\pi}'_\ell(u,w_j)-\widehat{\eta\pi}_\ell(u,w_j)\right|\psi_\ell(v,w_j)}{(1-\sqrt{c})^2}\\
&\le \sum_{\ell=0}^{\infty}\sum_{j=1}^{j_0}\frac{\frac{(1-\sqrt{c})^2\varepsilon}{6}\cdot 1\cdot\psi_\ell(v,w_j)}{(1-\sqrt{c})^2} = \frac{\varepsilon}{6}\sum_{\ell=0}^{\infty}\sum_{j=1}^{n}\psi_\ell(v,w_j) \le \frac{\varepsilon}{6}.
\end{aligned} \tag{22}$$

For the last inequality, we use the fact that the reserve $\psi_\ell(v, w_j)$ is at most $\pi_\ell(v, w_j)$, and thus $\sum_{\ell=0}^{\infty}\sum_{j=1}^{n}\psi_\ell(v, w_j) \le \sum_{\ell=0}^{\infty}\sum_{j=1}^{n}\pi_\ell(v, w_j) = 1$.

Next, we show that $\hat{s}_I^1(u, v)$ and $\hat{s}_I^2(u, v)$ differ by at most $\frac{\varepsilon}{6}$. To see this, note that by the property of backward search, we have $\left|\pi_\ell(v, w_j) - \psi_\ell(v, w_j)\right| \le 2r_{max} = \frac{(1-\sqrt{c})^2\varepsilon}{6}$ for a node w_j in the index. It follows that

$$\left|\hat{s}_I^1(u, v) - \hat{s}_I^2(u, v)\right| = \sum_{\ell=0}^{\infty}\sum_{j=1}^{j_0}\frac{\widehat{\eta\pi}_\ell(u, w_j)\left|\psi_\ell(v, w_j) - \pi_\ell(v, w_j)\right|}{(1-\sqrt{c})^2}$$

$$\le \sum_{\ell=0}^{\infty}\sum_{j=1}^{j_0}\frac{\widehat{\eta\pi}_\ell(u, w_j)\cdot 1\cdot\frac{(1-\sqrt{c})^2\varepsilon}{4}}{(1-\sqrt{c})^2} = \frac{\varepsilon}{6}\sum_{\ell=0}^{\infty}\sum_{j=1}^{n}\widehat{\eta\pi}_\ell(u, w_j) \le \frac{\varepsilon}{6}. \tag{23}$$

For the last inequality, recall that Algorithm 4 increments $\widehat{\eta\pi}$ at most n_r times, and each increment is $\frac{1}{n_r}$.

Finally, we show that $\hat{s}_I^2(u, v)$ approximates $s_I(u, v)$ with error $\frac{\varepsilon}{4}$ with target probability. Following the definition of $\widehat{\eta\pi}_\ell(u, w)$, we use a slightly different approach to construct $\hat{s}^2(u, v)$. For the i-th iteration, we sample a node w and a level ℓ with probability $\eta(w)\pi_\ell(u, w)$, and set X_i to be $\frac{\pi_\ell(v, w_j)}{(1-\sqrt{c})^2}$. It can be verify that $\hat{s}_I^2(u, v) = \frac{1}{n_r}\sum_{i=1}^{n_r} X_i$. For each X_i,

$$\mathrm{E}[X_i] = \sum_{\ell=0}^{\infty}\sum_{j=1}^{j_0}\eta(w_j)\pi_\ell(u, w_j)\frac{\pi_\ell(v, w_j)}{(1-\sqrt{c})^2} = s_I(u, v),$$

and $X_i \le \max_{\ell,v}\left\{\frac{\pi_\ell(v, w_j)}{(1-\sqrt{c})^2}\right\} \le \frac{1}{(1-\sqrt{c})^2}$. Since $n_r = \Theta(\log\frac{n}{\delta}/\varepsilon^2)$, by Chernoff bound,

$$\Pr\left[|\hat{s}_I^2(u, v) - s_I(u, v)| > \frac{\varepsilon}{6}\right] \le \frac{\delta}{2n}. \tag{24}$$

Combining Equations (22)-(24), we prove the lemma. □

B.6 Proof of Lemma 3.7

Proof. Consider a single $\sqrt{c}$-walk from u. Recall that Algorithm 4 first samples a node-level pair (w_j, ℓ) with probability $\pi_\ell(u, w_j)\eta(w_j)$. If $j > j_0$, it performs backward walk to generate an unbiased estimator $\hat{\pi}_\ell(v, w)$ for each $v \in V$, and set the estimator $\hat{s}_B(u, v)$ to be $\frac{\hat{\pi}_\ell(v, w_j)}{(1-\sqrt{c})^2}$. It follows that

$$E\left[\hat{s}_B(u, v)\right] = \sum_{\ell=0}^{\infty}\sum_{j=j_0+1}^{n}\pi_\ell(u, w_j)\eta(w_j)\cdot\frac{\hat{\pi}_\ell(v, w_j)}{(1-\sqrt{c})^2} = s_B(u, v).$$

We can bound the variance $\mathrm{Var}\left[\hat{s}_B(u, v)\right] \le E\left[\hat{s}_B(u, v)^2\right]$ by

$$E\left[\hat{s}_B(u, v)^2\right] = \sum_{\ell=0}^{\infty}\sum_{j=j_0+1}^{n}\pi_\ell(u, w_j)\eta(w_j)\cdot\frac{E\left[\hat{\pi}_\ell(v, w_j)^2\right]}{(1-\sqrt{c})^4}.$$

Lemma 3.5 implies that $E\left[\hat{\pi}_\ell(v, w_j)^2\right] \le \pi_\ell(v, w_j)$, and

$$\mathrm{Var}\left[\hat{s}_B(u, v)\right] \le \sum_{\ell=0}^{\infty}\sum_{j=j_0+1}^{n}\pi_\ell(u, w_j)\eta(w_j)\cdot\frac{\pi_\ell(v, w_j)}{(1-\sqrt{c})^4} = \frac{s_B(u, v)}{(1-\sqrt{c})^2}.$$

Recall that for a fixed i with $1 \le i \le f_r$, Algorithm 4 repeats above sampling process d_r time and use the mean over $d_r = \frac{12}{(1-\sqrt{c})^2\varepsilon^2}$ samples, denoted $\hat{s}_B^i(u, v)$, as an estimator for $s_B(u, v)$. It follows that

$$\mathrm{Var}\left[\hat{s}_B^i(u, v)\right] \le \frac{s_B(u, v)}{d_r(1-\sqrt{c})^2} = \frac{\varepsilon^2 s_B(u, v)}{12} \le \frac{\varepsilon^2}{12}.$$

By Chebyshev's inequality, we have

$$\Pr\left[|\hat{s}_B^i(u, v) - s_B(u, v)| > \frac{\varepsilon}{2}\right] \le \frac{4\mathrm{Var}\left[\hat{s}_B^i(u, v)\right]}{\varepsilon^2} \le \frac{1}{3}.$$

Finally, Algorithm 4 use $\hat{s}_B(u, v) = \mathrm{Median}_{1\le i\le f_r}\hat{s}_B^i(u, v)$ as the estimator for $\hat{s}_B(u, v)$. By setting $f_r = 3\log\frac{n}{\delta}$ and applying the Median Trick (see Lemma A.3), we have

$$\Pr\left[|\hat{s}_B(u, v) - s_B(u, v)| > \frac{\varepsilon}{2}\right] \le \frac{\delta}{2n}, \tag{25}$$

and the lemma follows. □

B.7 Proof of Lemma 3.9

Proof. Fix the source node u and consider a node w_j and a level ℓ. Recall that we retrieve all nodes v with $\psi_\ell(v, w_j)$ from the index if and only if 1) w_j is in the index, that is, $j \le j_0$, and 2) $\widehat{\eta\pi}_\ell(u, w_j) \ge \frac{(1-\sqrt{c})^2\varepsilon}{8} = \frac{\varepsilon}{c_1}$ Let $size_\ell(w_j) = \Theta\left(\frac{n\pi_\ell(w_j)}{\varepsilon}\right)$ denote the upper bound for the index size of w_j at level ℓ, and $size_\ell(w_j) = \sum_{\ell=0}^{\infty} size_\ell(w_j) = \Theta\left(\frac{n\pi(w_j)}{\varepsilon}\right)$ denote the upper bound for the index size of w_j. We further define $\widehat{\eta\pi}(u, w_j) = \sum_{\ell=0}^{\infty}\widehat{\eta\pi}_\ell(u, w_j)$. Note that $\widehat{\eta\pi}(u, w_j)$ is an unbiased estimator for $\sum_{\ell=0}^{\infty}\eta(w_j)\pi_\ell(u, w_j) = \eta(w_j)\pi(u, w_j)$. We can bound the $C_I(u)$ as

$$C_I(u) \le \sum_{\ell=0}^{\infty}\sum_{j=1}^{j_0} I\left(\widehat{\eta\pi}_\ell(u, w_j) > \frac{\varepsilon}{c_1}\right) size_\ell(w_j),$$

where $I\left(\widehat{\eta\pi}_\ell(u, w_j) > \frac{\varepsilon}{c_1}\right)$ equals 1 if $\widehat{\eta\pi}_\ell(u, w_j) > \frac{\varepsilon}{c_1}$ and equals 0 if otherwise. Since $\widehat{\eta\pi}_\ell(u, w_j) \le \widehat{\eta\pi}(u, w_j)$, we have $I\left(\widehat{\eta\pi}_\ell(u, w_j) > \frac{\varepsilon}{c_1}\right) \le I\left(\widehat{\eta\pi}(u, w_j) > \frac{\varepsilon}{c_1}\right)$, and thus

$$C_I(u) \le \sum_{\ell=0}^{\infty}\sum_{j=1}^{j_0} I\left(\widehat{\eta\pi}(u, w_j) > \frac{\varepsilon}{c_1}\right) size_\ell(w_j)$$

$$= \sum_{j=1}^{j_0} I\left(\widehat{\eta\pi}(u, w_j) > \frac{\varepsilon}{c_1}\right) size(w_j).$$

We now use two different approaches to bound $C_I(u)$. First, observe that for a given u, we have $\sum_{j=1}^{j_0}\widehat{\eta\pi}(u, w_j) \le 1$, which implies that there are at most $\frac{c_1}{\varepsilon}$ node w_j with $\widehat{\eta\pi}(u, w_j) \ge \frac{\varepsilon}{c_1}$. Since $size(w_1) \ge \ldots \ge size(w_{j_0})$, we can choose $\pi(u, w_1) \ge \varepsilon, \ldots \pi(u, w_{\frac{c_1}{\varepsilon}}) \ge \varepsilon$ to maximize the query cost $C_I(u)$. It follows that $C_I(u) \le \sum_{j=1}^{\frac{c_1}{\varepsilon}} size(w_j) \le O\left(\sum_{j=1}^{\frac{c_1}{\varepsilon}}\frac{n\pi(w_j)}{\varepsilon}\right)$ hence proves the first part of the lemma.

For the second part, note that $I\left(\widehat{\eta\pi}(u,w_j) > \frac{\varepsilon}{c_1}\right)$ is bounded by $\frac{\widehat{\eta\pi}_\ell(u,w_j)}{\varepsilon/c_1}$. It follows that

$$\mathrm{E}[C_I(u)] \le c_1 \sum_{j=1}^{j_0} \frac{\mathrm{E}[\widehat{\eta\pi}(u,w_j)]}{\varepsilon} size(w_j)$$

$$= c_1 \sum_{j=1}^{j_0} \frac{\eta(w_j)\pi(u,w_j)}{\varepsilon} size(w_j) \le c_1 \sum_{j=1}^{j_0} \frac{\pi(u,w_j)}{\varepsilon} size(w_j).$$

Here we use the fact that $\widehat{\eta\pi}(u,w_j)$ is an unbiased estimator for $\eta(w_j)\pi(u,w_j)$ and that $\eta(w_j) \le 1$. Taking average over all nodes $u \in V$, we have

$$C_I = \frac{1}{n}\sum_{u\in V} C_I(u) \le \frac{c_1}{n}\sum_{u\in V}\sum_{j=1}^{j_0} \frac{\pi(u,w_j)}{\varepsilon} size(w_j)$$

$$= c_1 \sum_{j=1}^{j_0} \frac{\frac{1}{n}\sum_{u\in V}\pi(u,w_j)}{\varepsilon} size(w_j) = c_1 \sum_{j=1}^{j_0} \frac{\pi(w_j)}{\varepsilon} size(w_j).$$

By $size(w_j) = O\left(\frac{n\pi(w_j)}{\varepsilon}\right)$, we have $C_I = O\left(\frac{n}{\varepsilon^2}\sum_{j=1}^{j_0}\pi(w_j)^2\right)$, and the lemma follows. □

B.8 Proof of Lemma 3.10

Proof. Next, we bound $C_B = \frac{1}{n}\sum_{v\in V} C_B(u)$, the average query cost for estimating the $\hat{\pi}_\ell(v,w)$ for each node w that is not in the Index. Given a source node u, for each node w_j with $j > j_0$, recall that we perform $\pi_\ell(u,w_j)n_r$ backward walk on w_j to estimate $\hat{\pi}_\ell(v,w), v \in V$. By Lemma 3.4, the cost of a single backward walk on w_j, regardless of the level ℓ, can be bounded by $O(n\pi(w_j))$. Ignoring the big-Oh,

$$\mathrm{E}[(u)] = \sum_{\ell=0}^{\infty}\sum_{j=1}^{j_0} \pi_\ell(u,w) n_r \cdot n\pi(w_j) = n_r n \sum_{j=1}^{j_0} \pi(u,w)\pi(w_j).$$

Taking average over all nodes $u \in V$, we have

$$\mathrm{E}[C_B] = \frac{1}{n}\sum_{u\in V}\mathrm{E}[C_B(u)] \le \frac{1}{n}\sum_{u\in V} n_r n \sum_{j=1}^{j_0}\pi(u,w)\pi(w_j)$$

$$= n_r n \sum_{j=1}^{j_0}\pi(w_j)\left(\sum_{u\in V}\pi(u,w)\right) = O\left(\frac{n\log n}{\varepsilon^2}\sum_{j=1}^{j_0}\pi(w_j)^2\right).$$

The last equation is due to $\sum_{u\in V}\pi(u,w) = n\pi(w)$. □

B.9 Proof of Theorem 3.11

Proof. We use $\beta = 1/\gamma$ to simplify the proof. Ignoring the big-Oh notation in Lemma 3.9, we have $\mathrm{E}[C_I] \le \frac{n}{\varepsilon}\sum_{j=1}^{\frac{c_1}{\varepsilon}}\pi(w_j)$ and $\mathrm{E}[C_I] \le \frac{n}{\varepsilon^2}\sum_{j=1}^{j_0}\pi(w_j)^2$. Plugging $\pi(w_j) = \frac{\kappa j^{-\beta}}{n^{1-\beta}}$ into $\frac{n}{\varepsilon}\sum_{j=1}^{\frac{c_1}{\varepsilon}}\pi(w_j)$, and we have

$$\mathrm{E}[C_I] \le \frac{n}{\varepsilon}\sum_{j=1}^{\frac{c_1}{\varepsilon}}\pi(w_j) = \sum_{j=1}^{\frac{c_1}{\varepsilon}}\frac{n\cdot j^{-\beta}}{n^{1-\beta}\varepsilon} = \frac{n^\beta\cdot\sum_{j=1}^{\frac{c_1}{\varepsilon}} j^{-\beta}}{\varepsilon}$$

$$= O\left(\frac{n^\beta}{\left(\frac{\varepsilon}{c_1}\right)^{1-\beta}\cdot\varepsilon}\right) = O\left(\frac{n^\beta}{\varepsilon^{1-\beta}\cdot\varepsilon}\right) = O\left(\frac{n^\beta}{\varepsilon^{2-\beta}}\right). \quad (26)$$

Plugging $\pi(w_j) = \frac{\kappa j^{-\beta}}{n^{1-\beta}}$ into $\frac{n}{\varepsilon^2}\sum_{j=1}^{j_0}\pi(w_j)^2$ follows that

$$\mathrm{E}[C_I] \le \frac{n}{\varepsilon^2}\sum_{j=1}^{j_0}\pi(w_j)^2 = \frac{n}{\varepsilon^2}\sum_{j=1}^{j_0}\frac{\kappa j^{-2\beta}}{n^{2-2\beta}} = \frac{\kappa n^{2\beta-1}}{\varepsilon^2}\sum_{j=1}^{j_0} j^{-2\beta}.$$

For $\beta < 1/2$, we have $\sum_{j=1}^{j_0} j^{-2\beta} = O(j_0^{1-2\beta}) = O(n^{1-2\beta})$, and thus $\mathrm{E}[C_I] = O\left(\frac{n^{2\beta-1}}{\varepsilon^2}\cdot n^{1-2\beta}\right) = O\left(\frac{1}{\varepsilon^2}\right)$. For $\beta = 1/2$, we have $\sum_{j=1}^{j_0} j^{-2\beta} = O(\log j_0)$. Since $\log j_0 \le \log n$ and $n^{2\beta-1} = 1$, we have $\mathrm{E}[C_I] = O\left(\frac{n^{2\beta-1}}{\varepsilon^2}\cdot\log j_0\right) = O\left(\frac{\log n}{\varepsilon^2}\right)$. For $\beta > 1/2$, we have $\sum_{j=1}^{j_0} j^{-2\beta} = O(1)$ and consequently $\mathrm{E}[C_I] = O\left(\frac{n^{2\beta-1}}{\varepsilon^2}\right)$. Combining Equation (26) and above analysis, we have the following equation:

$$\mathrm{E}[C_I] = \begin{cases} O(\frac{1}{\varepsilon^2}), & \text{for } \beta < 1/2; \\ O(\frac{\log n}{\varepsilon^2}), & \text{for } \beta = 1/2; \\ O\left(\min\left\{\frac{n^{2\beta-1}}{\varepsilon^2}, \frac{n^\beta}{\varepsilon^{2-\beta}}\right\}\right), & \text{for } \beta > 1/2. \end{cases} \quad (27)$$

By Lemma 3.10 and the assumption $\pi(w_j) = \frac{\kappa j^{-\beta}}{n^{1-\beta}}$, we have $\mathrm{E}[C_B] = O\left(\frac{c_1 n^{2\beta-1}\log n}{\varepsilon^2}\sum_{j=j_0+1}^{n} j^{-2\beta}\right)$. For $j < 1/2$, we have $\sum_{j=j_0+1}^{n} j^{-2\beta} = O(n^{1-2\beta})$. Thus

$$\mathrm{E}[C_B] = O\left(\frac{n^{2\beta-1}\log n}{\varepsilon^2}\cdot n^{1-2\beta}\right) = O\left(\frac{\log n}{\varepsilon^2}\right).$$

For $j = 1/2$, we have $\sum_{j=j_0+1}^{n} j^{-2\beta} = O(\log n)$, and thus $\mathrm{E}[C_B] = O\left(\frac{\log n\log\frac{n}{\delta}}{\varepsilon^2}\right)$. For $j > 1/2$, we have $\sum_{j=j_0+1}^{n} j^{-2\beta} = O(j_o^{1-2\beta})$. Plugging $j_0 \le n\left(\varepsilon\bar{d}\right)^{\frac{1}{1-\beta}}$ follows that

$$\mathrm{E}[C_B] = O\left(\frac{n^{2\beta-1}\log n}{\varepsilon^2}\cdot\left(n(\varepsilon\bar{d})^{\frac{1}{1-\beta}}\right)^{1-2\beta}\right)$$

$$= O\left(\frac{\log n}{\varepsilon^2}\cdot(\varepsilon\bar{d})^{\frac{1-2\beta}{1-\beta}}\right) = O\left(\frac{\log n}{\varepsilon^{\frac{1}{1-\beta}}\bar{d}^{\frac{2\beta-1}{1-\beta}}}\right).$$

By $\varepsilon \ge \log^{\frac{1-\beta}{2\beta-1}} n/n^{1-\beta}\bar{d}^{\frac{2\beta-1}{\beta}}$ and $\delta > \frac{1}{n^{\Omega(1)}}$, it follows that $\log\frac{n}{\delta}/\varepsilon^{\frac{1}{1-\beta}}\bar{d}^{\frac{2\beta-1}{1-\beta}} \le \frac{n^{2\beta-1}}{\varepsilon^2}$ and $\log n/\varepsilon^{\frac{1}{1-\beta}}\bar{d}^{\frac{2\beta-1}{1-\beta}} \le \frac{n^\beta}{\varepsilon^{2-\beta}}$, and thus $\mathrm{E}[C_B]$ is bounded by $O\left(\min\left\{\frac{n^{2\beta-1}}{\varepsilon^2}, \frac{n^\beta}{\varepsilon^{2-\beta}}\right\}\right)$ for $\beta > 1/2$. In summary, we have

$$\mathrm{E}[C_B] = \begin{cases} O(\frac{\log\frac{n}{\delta}}{\varepsilon^2}), & \text{for } \beta < 1/2; \\ O(\frac{\log n\log\frac{n}{\delta}}{\varepsilon^2}), & \text{for } \beta = 1/2; \\ O\left(\min\left\{\frac{n^{2\beta-1}}{\varepsilon^2}, \frac{n^\beta}{\varepsilon^{2-\beta}}\right\}\right), & \text{for } \beta > 1/2. \end{cases} \quad (28)$$

Combing C_F, C_I, C_B and $\beta = 1/\gamma$, the theorem follows. □