TopPPR: Top-k Personalized PageRank Queries with Precision Guarantees on Large Graphs

Zhewei Wei*
School of Infromation, Renmin University of China
zhewei@ruc.edu.cn

Xiaodong He
School of Infromation, Renmin University of China
hexiaodong_1993@ruc.edu.cn

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

Sibo Wang
University of Queensland
sibo.wang@uq.edu.au

Shuo Shang
CEMSE, King Abdullah University of Science and Technology
jedi.shang@gmail.com

Ji-Rong Wen‡
School of Infromation, Renmin University of China
jrwen@ruc.edu.cn

ABSTRACT

Personalized PageRank (PPR) is a classic metric that measures the relevance of graph nodes with respect to a *source node*. Given a graph G, a source node s, and a parameter k, a *top-k PPR query* returns a set of k nodes with the highest PPR values with respect to s. This type of queries serves as an important building block for numerous applications in web search and social networks, such as Twitter's Who-To-Follow recommendation service. Existing techniques for top-k PPR, however, suffer from two major deficiencies. First, they either incur prohibitive space and time overheads on large graphs, or fail to provide any guarantee on the *precision* of top-k results (i.e., the results returned might miss a number of actual top-k answers). Second, most of them require significant pre-computation on the input graph G, which renders them unsuitable for graphs with frequent updates (e.g., Twitter's social graph).

To address the deficiencies of existing solutions, we propose *TopPPR*, an algorithm for top-k PPR queries that ensure at least ρ precision (i.e., at least ρ fraction of the actual top-k results are returned) with at least $1 - 1/n$ probability, where $\rho \in (0, 1]$ is a user-specified parameter and n is the number of nodes in G. In addition, TopPPR offers non-trivial guarantees on query time in terms of ρ, and it can easily handle dynamic graphs as it does not require any preprocessing. We experimentally evaluate TopPPR using a variety of benchmark datasets, and demonstrate that TopPPR outperforms the state-of-the-art solutions in terms of both efficiency and precision, even when we set $\rho = 1$ (i.e., when TopPPR returns the *exact* top-k results). Notably, on a billion-edge Twitter graph, TopPPR only requires 15 seconds to answer a top-500 PPR query with $\rho = 1$.

*Work partially done at Beijing Key Laboratory of Big Data Management and Analysis Methods, and at Key Laboratory of Data Engineering and Knowledge Engineering, MOE, Renmin University of China.
†Work partially done at the Nanyang Technological University, Singapore.
‡Shuo Shang and Ji-Rong Wen are the corresponding authors.

CCS CONCEPTS

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

KEYWORDS

Personalized PageRank; Top-k Queries

ACM Reference Format:
Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. 2018. TopPPR: Top-k Personalized PageRank Queries with Precision Guarantees on Large Graphs. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3183713.3196920

1 INTRODUCTION

Given a graph G and a *source* node s in G, the *Personalized PageRank (PPR)* of any node v in G is a topology-based measure of v's relevance with respect to s [34]. PPR has been an important building block of numerous web search and social network applications, such as Twitter's *Who-To-Follow* [23], LinkedIn's connection recommendation [1], and Pinterest's *Related Pins* [28]. For these applications, the overheads incurred PPR queries are of significant concerns, due to the large sizes of the underlying graphs. This motivates a plethora of solutions [7, 8, 10–13, 15–22, 26, 27, 29–33, 35–37, 40–45] that aim to improve the efficiency of PPR queries.

Existing methods for PPR mainly utilize three types of techniques: matrix optimizations, local updates, and Monte-Carlo (MC) simulations. In particular, matrix-based methods [13, 15, 27, 32, 37, 45] formulate PPR as the solution to a linear system, and they apply matrix optimization approaches to reduce query costs. Local update methods [7, 8, 17–20, 26, 41], on the other hand, utilize graph traversals instead of matrix operations for PPR computation. Meanwhile, MC-based methods [10, 16, 29–31, 36, 39, 40] use random walks to derive approximate PPR values with probabilistic guarantees. Among these methods, matrix-based solutions are unfavorable as they incur significant space or time overheads on large graphs; because of this, the state-of-the-art techniques for PPR queries [30, 39–41] only incorporate local updates and MC simulations.

Motivation. Existing work mostly considers three types of PPR queries:

- *Point-to-point* queries, which ask for the PPR of a given node v with respect to the source node s;
- *Single-source* queries, which ask for the PPR value of every node with respect to s;
- *Top-k* queries, which ask for the k nodes whose PPR with respect to s are the largest.

Among these query types, top-k PPR queries are particularly useful for recommendation applications (e.g., [23, 28]), in which only the nodes most relevant to s are of interests. For such queries, the *precision* of query answers (i.e., the fraction of answers that are among the actual top-k) is highly important, as false positive may considerably degrade the quality of recommendations. Most existing PPR techniques, however, do not provide any guarantees on the precision of top-k results. In particular, the state-of-the-art top-k PPR algorithms [30, 39–41] only ensure that the i-th node returned has a PPR that is at least $1 - \epsilon_r$ times that of the i-th node in the actual top-k results (e.g., $\epsilon_r = 1/2$ [39, 40]); nevertheless, this does not guarantee high precisions since there may exist a large number of non-top-k nodes with PPR values above the $1 - \epsilon_r$ bar.

To our knowledge, there exist only a few methods [15, 17–20, 41, 42] that offer precision assurance for top-k PPR queries. Unfortunately, these methods either are matrix-based or rely on (less-optimized) local update techniques, due to which they are unable to handle massive graphs efficiently. Specifically, as we show in Section 7, even the most advanced method [15] among them requires 50 seconds on average to answer an exact top-500 PPR query on a medium-size *LiveJournal* graph, which is far from sufficient for real-time applications.

Contributions. To remedy the deficiencies of existing techniques, this paper presents *TopPPR*, a top-k PPR algorithm that provides both practical efficiency and strong theoretical guarantees. Given a precision parameter $\rho \in (0, 1]$, TopPPR answers any top-k PPR query while ensuring at least ρ precision with at least $1 - 1/n$ probability, where n is the number of nodes in the input graph G. (That is, no less than ρ fraction of the results returned are among the actual top-k.) The expected time complexity of TopPPR is $O\left(\frac{m+n\log n}{\sqrt{gap_\rho}}\right)$ on worst case graphs and $O\left(\frac{k^{\frac{1}{4}} n^{\frac{3}{4}} \log n}{\sqrt{gap_\rho}}\right)$ on power law graphs, where m is the number of edges in G and gap_ρ is a value that quantifies the difference between the top-k and non-top-k PPR values (see Section 4 for details). In particular, when $\rho = 1$, gap_ρ equals the k-th largest PPR value minus the $(k + 1)$-th.

The basic idea of TopPPR is to adopt the filter-refinement paradigm for top-k processing. In the filter step, it computes a rough estimation of each node's PPR, based on which it identifies a candidate node set C. Then, in the refinement step, it iteratively refines the PPR estimation for each node in C, until it derives the top-k results with high confidence. The main challenge in this filter-refinement approach is that we need accurate PPR estimations to avoid missing actual top-k results, and yet, we cannot afford to pay high computation cost in PPR derivation. TopPPR addresses this challenge with an *adaptive* approach that enables it to focus only on a small set of nodes whose PPR estimations really matter for top-k precision, while omitting those nodes that have no chance to be part of the top-k results. Furthermore, *TopPPR* incorporates an advanced sample approach that helps it reduce computation cost without sacrificing accuracy.

We experimentally evaluate TopPPR on a variety of benchmark datasets with up to 40 million nodes and 1 billion edges. Our results demonstrate that TopPPR outperforms state-of-the-art methods for both exact and approximate top-k queries. Notably, on a billion-edge *Twitter* graph, TopPPR only requires 15 seconds to answer a top-500 PPR query with $\rho = 1$.

2 PRELIMINARIES

2.1 Problem Definition

Let $G = (V, E)$ be a directed graph with n nodes and m edges. Given a source node $s \in V$ and a decay factor α, a random walk from s is a traversal of G that starts from s and, at each step, either (i) terminates at the current node with α probability, or (ii) proceeds to a randomly selected out-neighbor of the current node. For any node $v \in V$, the *personalized PageRank (PPR)* $\pi(s, v)$ of v with respect to s is the probability that a random walk from s terminates at v [34].

We define the *i-th node with respect to s* as the node whose PPR with respect to s is the i-th largest, and we denote it as t_k. Accordingly, we refer to $V_k = \{t_1, \dots, t_k\}$ as the *top-k node set with respect to s*. We consider the following type of top-k PPR queries.

Definition 2.1 (ρ-precise Top-k PPR Queries). Given a source node s and $\rho \in (0, 1]$, a ρ-precise top-k PPR query returns a set S of k nodes, such that at least $\rho \cdot k$ nodes in S are in V_k.

Note that for fixed ρ, there exist n^2 possible top-k PPR queries (for different choices of s and k). We say that a top-k PPR algorithm is *ρ-precise with at least $1 - x$ probability*, if there is at most x probability that the algorithm answers one of the n^2 top-k PPR queries with a precision lower than ρ. Our objective is to develop a top-k PPR method that (i) is ρ-precise with at least $1 - 1/n$ probability, (ii) does not require preprocessing, and (iii) is computationally efficient. Table 1 lists notations frequently used in our paper.

Remark. Note that Definition 2.1 imposes a requirement on the top-k results' precision, but does not consider the order of the results returned. This is because, in practical applications, top-k PPR queries are often used only to identify a candidate set of k nodes, which are then re-ranked using other algorithms for recommendation; as a consequence, the order of the top-k answers are irrelevant. For example, when Twitter's Who-To-Follow service [23] attempts to recommend other users to a given user s, it first performs a top-500 PPR queries to derive a candidate set of 500 users, and then re-score those users using a different approach for recommendation. Pinterest's Related Pins service [28] also adopts a similar mechanism. Nonetheless, as we show in Section 7, the order of the top-k answers produced by our algorithm is still comparable to the state of the art.

2.2 Basic Techniques and State of the Art

In what follows, we introduce three basic techniques for PPR computation: random walk sampling [16], forward search [8], and backward search [7]. We then discuss how these techniques have been adopted in the state-of-the-art solutions [30, 39–41].

Table 1: Frequently used notations.

Notation	Description
G=(V, E)	The input graph G with node set V and edge set E
n, m	The number of nodes and edges in G, respectively
$N^{out}(v)$, $N^{in}(v)$	The set of out-neighbors and in-neighbors of node v
$d_{out}(v)$, $d_{in}(v)$	The out-degree and in-degree of node v
$\pi(s, t)$	The exact PPR value of t with respect to s
α	The termination probability
$r^b(s, t)$, $\pi^b(s, t)$	The reserve and residue of t from s in the backward search
$r^f(s, t)$, $\pi^f(s, t)$	The reserve and residue of t from s in the forward search
r^f_{sum}	The sum of all nodes' residues during in the forward search from s
gap_ρ	the gap between the $\lceil \rho k \rceil$-th and $(k+1)$-th node, $gap_\rho = \pi(s, t_{\lceil \rho k \rceil}) - \pi(s, t_{k+1})$

Algorithm 1: Forward Search

```
Input: Graph $G$, source node $s$, decay factor $\alpha$, threshold $r^f_{max}$
Output: Forward residue $r^f(s, v)$ and reserve $\pi^f(s, v)$ for all $v \in V$
$r^f(s, s) \leftarrow 1$ and $r^f(s, v) \leftarrow 0$ for all $v \neq s$;
$\pi^f(s, v) \leftarrow 0$ for all $v$;
while $\exists u \in V$ such that $r^f(s, u)/d_{out}(u) \geq r^f_{max}$ do
    for each $v \in \mathcal{N}^{out}(u)$ do
        $r^f(s, v) \leftarrow r^f(s, v) + (1 - \alpha) \cdot \frac{r^f(s,u)}{d_{out}(u)}$;
    $\pi^f(s, u) \leftarrow \pi^f(s, u) + \alpha \cdot r^f(s, u)$;
    $r^f(s, u) \leftarrow 0$;
```

Random Walk Sampling [16]. Given two nodes s and v, the simplest approach to estimate $\pi(s, v)$ is to generate a number ω of random walks from s, and then use the fraction of walks that terminate at v as an approximation of $\pi(s, v)$. To obtain an estimation of $\pi(s, v)$ with at most ε absolute error with probability at least $1 - 1/n$, the number of random walks required is $O\left(\frac{\log n}{\varepsilon^2}\right)$, which is significant when ε is small.

Forward Search [8]. Roughly speaking, forward search can be regarded as a deterministic counterpart of random walk sampling. (See Algorithm 1 for a pseudo-code.) It first assigns a *forward residue* $r^f(s, v)$ and a *forward reserve* $\pi^f(s, v) = 0$ for each node v, such that $r^f(s, s) = 1$ and $r^f(s, v) = 0$ for any $v \neq s$ (Lines 1-2). After that, it starts a traversal of G from s and updates the forward residue and reserve of each node that it visits (Lines 3-7). In particular, there is a node u whose forward residue $r^f(s, u)$ is larger than its out-degree $d_{out}(u)$ times a threshold r^f_{max} (Line 3), then it increases u's forward reserve $\pi^f(s, u)$ by $\alpha \times r^f(s, u)$ (Line 6), and it increases each of u's out-neighbors' forward residue by $\frac{(1-\alpha)}{d_{out}(u)} \times r^f(s, u)$ (Lines 4-5), after which it sets $r^f(s, u) = 0$ (Line 7). In other words, it converts α fraction of u's forward residue into its forward reserve, and the divide the other $1 - \alpha$ fraction among the out-neighbors of u.

Algorithm 2: Backward Search

```
Input: Graph $G$, target node $t$, decay factor $\alpha$, threshold $r^b_{max}$
Output: Backward residue $r^f(s, v)$ and reserve $\pi^f(s, v)$ for all
        $v \in V$
$r^b(t, t) \leftarrow 1$ and $r^b(v, t) \leftarrow 0$ for all $v \neq t$ and $i = 1, \ldots, j$;
$\pi^b(v, t) \leftarrow 0$ for all $v$;
while $\exists v$ such that $r^b(v, t) > r^b_{max}$ do
    for each $u \in \mathcal{N}^{in}(v)$ do
        $r^b(u, t) \leftarrow r^b(u, t) + (1 - \alpha) \cdot \frac{r^b(v,t)}{d_{out}(u)}$
    $\pi^b(v, t) \leftarrow \pi^b(v, t) + \alpha \cdot r^b(v, t)$;
    $r^b(s, v) \leftarrow 0$;
```

The algorithm terminates when no such node u exists. It is shown in [8] that the algorithm runs in $O(1/r^f_{max})$ time, and that when r^f_{max} approaches 0, $\pi^f(s, v)$ converges to $\pi(s, v)$. However, for fixed $r^f_{max} > 0$, there is no known result regarding the approximation guarantee of $\pi^f(s, v)$ with respect to $\pi(s, v)$.

Backward Search [7]. Backward search is a reversed version of forward search that traverses the incoming edges of G to derive PPR. (See Algorithm 2 for a pseudo-code.) In particular, given a *destination* node t, it employs a traversal from t to compute t's PPR value $\pi(v, t)$ with respect to any other node v. It starts by assigning a *backward residue* $r^b(v, t)$ and a *backward reserve* $\pi^b(v, t) = 0$ to each node v, and setting $r^b(t, t) = 1$ and $r^b(v, t) = 0$ for any $v \neq t$ (Lines 1-2). Subsequently, it traverses from t, following the incoming edges of each node. For any node v that it visits, it checks if v's backward residue $r^b(v, t)$ is larger than a given threshold r^b_{max}. If so, then it increases v's backward reserve by $\alpha \times r^b(v, t)$ and, for each in-neighbor u of v, increases the backward residue of u by $(1 - \alpha) \cdot \frac{r^b(v,t)}{d_{out}(u)}$ (Lines 4-6). After that, it reset v's backward residue $r^b(v, t)$ to 0 (Line 7). Lofgren et al. [30] prove that the algorithm has an amortized time complexity of $O\left(\frac{m}{n \cdot r^b_{max}}\right)$. When the algorithm terminates, it ensures that $|\pi^b(v, t) - \pi(v, t)| < r^b_{max}$ for any v. Backward search requires the destination node t instead of the source node s to be fixed.

State of the Art. Although each of the above basic techniques has its limitations, recent work [30, 39–41] shows that they can be integrated to construct advanced solutions with enhanced guarantees. In particular, Lofgren et al. [31] propose to answer point-to-point PPR queries by combining random walks from the source node s with a backward search from the destination node t, and show that the combination leads to improved time complexity. This approach is further improved in [30, 39]. Wang et al. propose to process single-source PPR queries by first performing a forward search from the source node s, and then generating random walks from those nodes with non-zero residues. They demonstrate that this method provides even better efficiency (for single-source and top-k PPR queries) than the solutions in [30, 31, 39] do. However, none of these methods leverage the combined strengths of random walks, forward search, and backward search simultaneously. In contrast, our top-k PPR algorithm carefully integrates all three basic techniques in a non-trivial manner that enables us to maximize the efficiency of top-k PPR queries, as we will elaborate in Section 3.

3 TOPPPR ALGORITHM

In this section, we present TopPPR, an index-free algorithm for ρ-precise top-k PPR queries on large graphs. Before diving into the details, we give some high-level ideas of the algorithm.

3.1 Challenges and Main Ideas

Challenges. The main challenge for ρ-precise top-k PPR query lies in the case when $\rho = 1$, i.e., when the algorithm is required to return the exact top-k node set. Previous work on exact top-k PPR queries, such as *FLoS* [41] and *Chopper* [15], rely on matrix optimizations or local updates to compute the exact values of the personalized PageRanks, which makes them inefficient on large graphs. On the other hand, sampling-based PPR algorithms, such as FastPPR [31], BiPPR [30], HubPPR [39], and FORA [40], focus on relative error guarantees and are unable to obtain accurate top-k results if the the gap between the k-th and $(k+1)$-th largest PPR values is small. For example, in a typical exact top-500 PPR query on the *Twitter* data, the k-th largest PPR value is at the order of 10^{-5}, while the gap between the k-largest and $(k+1)$-th largest PPR values ranges from 10^{-7} to 10^{-10}. To ensure 100% precision for such top-k queries, the relative error allowed ranges from 10^{-5} to 10^{-2}, which leads to tremendous computation costs. In particular, our experiments show that previous approaches require at least 1000 seconds to answer a top-500 PPR query exactly on *Twitter*.

High Level Ideas. Our TopPPR algorithm for top-k PPR queries utilizes the three basic techniques introduced in Section 2.2, namely, random walk sampling, forward search, and backward search. Figure 1 illustrates the high level idea of TopPPR. In a nutshell, TopPPR first performs forward search from the source node s, and then conducts random walks from those nodes with non-zero forward residues; after that, it applies backward search from some target nodes and combines the results with the random walks to estimate PPR values for top-k derivation.

Compared with FORA [40] (which utilizes only forward search and random walk sampling), TopPPR leverages backward search to significantly reduce the variances of PPR estimators and achieve much higher efficiency, as we explain in Section 4. In addition, TopPPR's application of backward search is *adaptive*, in the sense that it varies the *depth* of the backward search from each node to avoid unnecessary computation. Specifically, for any node v, if TopPPR can easily decide whether v is a top-k or non-top-k node, then the backward search from v would be shallow (or even empty), so as to reduce computation overheads; on the other hand, if it is difficult to identify whether v is among the top-k results, then TopPPR would perform a deep backward search from v, so as to obtain a more accuracy estimation of v's PPR. This paradigm is made possible by maintaining a set C that consists of possible top-k nodes, and by adaptively pruning the nodes in C using confidence bounds. To guarantee a precision of at least ρ, TopPPR maintains a set V_k of nodes that are bound to be among the true top-k, and stops when $|V_k| \geq \rho k$.

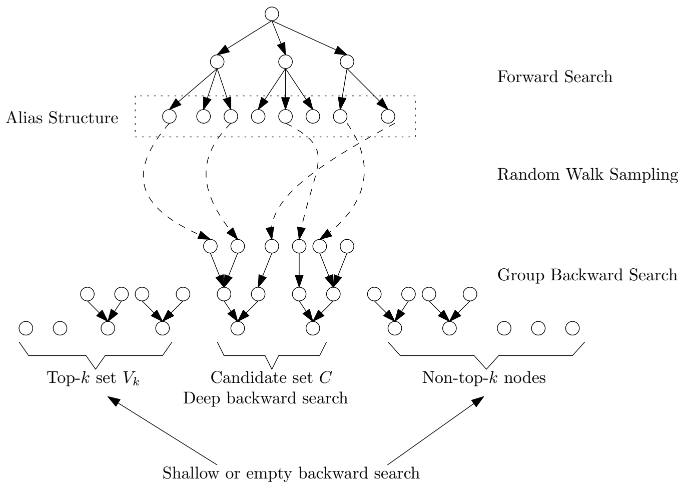


Figure 1: Illustration of three phase estimation

3.2 Techniques

We now present several key techniques used in the TopPPR algorithm.

Estimation Formula. We use an estimation formula that combines the forward search, backward search, and random walk sampling. It is proven in [8] that a forward search from node s gives the following formula:

$$\pi(s,t) = \pi^f(s,t) + \sum_{u \in V} r^f(s,u)\pi(u,t).$$

Similarly, a backward search [7] from node t gives

$$\pi(s,t) = \pi^b(s,t) + \sum_{v \in V} \pi(s,v) r^b(v,t).$$

Combining the above two equations leads to

$$\begin{aligned}\pi(s,t) &= \pi^f(s,t) + \sum_{u \in V} r^f(s,u)\pi(u,t)\\ &= \pi^f(s,t) + \sum_{u \in V} r^f(s,u)\left(\pi^b(u,t) + \sum_{v \in V} \pi(u,v) r^b(v,t)\right)\\ &= \pi^f(s,t) + \sum_{u \in V} r^f(s,u)\pi^b(u,t) + \sum_{u,v \in V} r^f(s,u)\pi(u,v)r^b(v,t).\end{aligned}$$

Note that once the forward and backward search stops, $\pi^f(s,t) + \sum_{u \in V} r^f(s,u)\pi^b(u,t)$ is determined. We then use random walk samples to obtain an estimation $\hat{\pi}(u,v)$ of $\pi(u,v)$, after which we apply the following formula to estimate $\pi(s,t)$:

$$\hat{\pi}(s,t) = \pi^f(s,t) + \sum_{u \in V} r^f(s,u)\pi^b(u,t) + \sum_{u,v \in V} r^f(s,u)\hat{\pi}(u,v)r^b(v,t). \tag{1}$$

Forward Push and Alias Method. As mentioned in Section 1, we perform forward search to compute the forward residues $r^f(s,u)$ and reserves $\pi^f(s,u)$ for each $u \in V$. We then construct an Alias structure [38] (see Section B for a description) on the forward residues $r^f(s,u)$, so that a node u can be sampled according to probability $r^f(s,u)/r^f_{sum}$ in constant time, where $r^f_{sum} = \sum_{u \in V} r^f(s,u)$.

Algorithm 3: Group Backward Search

```
Input: Graph $G$, candidate set $C$, decay factor $\alpha$, threshold $r^b_{max}$, estimators $\hat{\pi}_b$
Output: Inverted lists of backward residues $r^b$ and reserves $\pi^b$
1  for each $t \in C$ do
2      $r^b(t,t) \leftarrow 1$ and $r^b(v,t) \leftarrow 0$ for all $v \neq t$ and $i = 1, \dots, j$;
3      $\pi^b(v,t) \leftarrow 0$ for all $v$, $\hat{\pi}(s,t) \leftarrow \frac{1}{n}$ if $\hat{\pi}(s,t) = 0$;
4      while $\exists v$ such that $r^b(v,t) > r^b_{max}\sqrt{\frac{d_{in}(v)}{\hat{\pi}_b(s,v)}}$ do
5          for each $u \in \mathcal{N}^{in}(v)$ do
6              $r^b(u,t) \leftarrow r^b(u,t) + (1-\alpha) \cdot \frac{r^b(v,t)}{d_{out}(u)}$
7          $\pi^b(v,t) \leftarrow \pi^b(v,t) + \alpha \cdot r^b(v,t)$;
8          $r^b(s,v) \leftarrow 0$;
9      for each $v \in V$ with non-zero $r^b(v,t)$ do
10         Append $(t, r^b(v,t))$ to inverted list $r^b$ at node $v$;
11     for each $v \in V$ with non-zero $\pi^b(v,t)$ do
12         Append $(t, \pi^b(v,t))$ to inverted list $\pi^b$ at node $v$;
```

Bernstein Inequalities and Confidence Bounds. We compute confidence bounds using Berstein inequality and its empirical version, as shown in the following lemma.

Lemma 3.1 (Bernstein inequality and empirical Bernstein inequality [9]). *Let $X_1, \dots, X_\ell$ be real-valued i.i.d. random variables, such that $X_i \in [0, r]$, $E[X_i] = \mu$ and $Var[X_i] = \sigma^2$. Let $\bar{X}_\ell = \frac{1}{\ell}\sum_{i=1}^{\ell} X_i$ denote the empirical mean, and $\bar{\sigma}^2 = \frac{1}{\ell}\sum_{i=1}^{\ell}(X_i - \bar{X}_\ell)^2$ denote the empirical variance. With probability $1 - p_f$, we have*

$$|\bar{X}_\ell - \mu| \le \sqrt{\frac{\sigma^2 \log \frac{2}{p_f}}{\ell}} + \frac{2r \log \frac{1}{p_f}}{\ell},$$

and

$$|\bar{X}_\ell - \mu| \le \sqrt{\frac{2\bar{\sigma}^2 \log \frac{3}{p_f}}{\ell}} + \frac{3r \log \frac{3}{p_f}}{\ell}.$$

Compared with the Chernoff bounds, Bernstein inequalities are able to provide much tighter estimation when the variances of the random variables are small. In TopPPR, we use the empirical Bernstein inequality since the upper bound on the actual variance is hard to obtain.

Based on the empirical Bernstein inequality, we use the following confidence bounds to estimate the range of $\pi(s,t)$. After n_r samples, let $\hat{\pi}(s,t)$ be the estimator of $\pi(s,t)$, and $\bar{\sigma}^2(s,t)$ be the empirical variance. We define the confidence bound of estimator $\hat{\pi}(s,t)$ to be $[\hat{\pi}(s,t) - \beta(s,t), \hat{\pi}(s,t) + \beta(s,t)]$, where $\beta(s,t)$ is a parameter calculated by the Bernstein inequality and union bound. As we shall see in Section 4, if we set

$$\beta(s,t) = \sqrt{\frac{2\bar{\sigma}^2(s,t)\ln(3n^3\log^2 n_r)}{n_r}} + \frac{3r^f_{sum}\ln(3n^3\log^2 n_r)}{n_r},$$

then our algorithm guarantees that with high probability, for any $t \in V$, $\pi(s,t) \in [\hat{\pi}(s,t) - \beta(s,t), \hat{\pi}(s,t) + \beta(s,t)]$. Note that here we set p_f lower than $1/n^3$, so that we can apply union bounds over $O(n^2)$ events in our algorithm. We use the confidence bounds for pruning candidate nodes and designing stopping rule of TopPPR.

Group Backward Search. Backward search is an effective tool for reducing the variances of the estimators provided by the random walk sampling process. However, if we naively apply backward search to all target nodes $t \in V$ with the same maximum residue threshold r^b_{max}, it will incur significant cost due to the massive number of nodes in V. Intuitively, we should spend less time on nodes with PPR values that are far away from the k-largest PPR value $\pi(s,t_k)$, and spend more time on nodes with PPR values that are close to $\pi(s,t_k)$. Based on this intuition, we propose the *group backward search* algorithm. Group backward search takes a subset $C \in V$, referred to as the *candidate set*, and performs backward search on each node in C. The TopPPR algorithm adaptively shrinks the size of C using confidence bounds, such that it can apply deeper backward search to nodes in C for more accurate estimations.

The key idea of the group backward search algorithm is to allow each node to have a different threshold r^b_{max} on its backward residue. Recall that given a target node t, the original backward search pushes the residue of a node v to its in-neighbours whenever the residue $r^b(v,t) > r^b_{max}$. This ensures that the residue $r^b(v,t) \le r^b_{max}$ for any node $v \in V$ at the end of the backward search. Setting a unified threshold r^b_{max} for each internal node $v \in V$ is crucial to apply the Chernoff inequality, as it gives an upper bound for the range of the random variables. In the group backward search, however, Bernstein inequality allows us to set r^b_{max} for different $v \in V$, as long as we can bound the variance of the estimation. In particular, let $r^b_{max}(v,t)$ denote the maximum backward residue allowed for node v and target node t in the group backward search, we propose to set $r^b_{max}(v,t) = r^b_{max}(t) \cdot \sqrt{\frac{d_{in}(v)}{\hat{\pi}_b(s,v)}}$ where $d_{in}(v)$ is the indegree of v, $r^b_{max}(t)$ is the threshold for target node t, and $\hat{\pi}_b(s,v)$ is constant approximation of the actual PPR value $\pi(s,t)$. Intuitively, a backward push on node v is expensive if its indegree $d_{in}(v)$ is large, and thus we should avoid backward push on v unless it significantly reduces the residue of v. To see why the PPR value $\pi(s,v)$ takes a part in the threshold function, recall that the goal of the group backward search is to minimize the variance $\bar{\sigma}^2(s,t)$. Since $\bar{\sigma}^2(s,t) \le \sum_{v \in V} r^b(v,t)^2 \pi(s,v)$, the contribution of $r^b(v,t)^2\pi(s,v)$ to $\bar{\sigma}^2(s,t)$ depends on the quantity of $\pi(s,v)$: if $\pi(s,v)$ is very small, then $r^b(v,t)^2\pi(s,v)$ is ignorable, an thus spending time to reduce $r^b(v,t)$ is meaningless; if $\pi(s,v)$ is large, reducing $r^b(v,t)$ can significantly reduce the variance and therefore lead to better estimation.

Algorithm 3 shows the pseudocode of group backward search. For each $t \in C$, we first set $r^b(v,t) = 0$ for $v \neq t$ and $r^b(v,t) = 0$ for $v = t$ (Line 1). We also set $\pi(v,t) = 0$ for $v \in V$ and estimator $\hat{\pi}_b(s,v) = \frac{1}{n}$ if $\hat{\pi}_b(s,v) = 0$ (Line 2). Then, for each v with residue $r^b(v,t) > r^b_{max}\sqrt{\frac{d_{in}(v)}{\hat{\pi}_b(s,v)}}$, we transfer $(1-\alpha)$ fraction of $r^b(v,t)$ to its in-neighbours (Lines 4-6), and α-fraction of $r^b(v,t)$ to the reserve $\pi^b(s,v)$ (Lines 7-8). Finally, we append $(t, r^b(v,t))$ and $(t, \pi^b(v,t))$ to inverted lists r^b and π^b, so that given v, we can find each t with non-zero $r^b(v,t)$ or $\pi^b(v,t)$ in linear time (Line 12).

3.3 Main Algorithm

Algorithm 4 shows the pseudocode of TopPPR. Given a graph G, a source node s, a decay factor α, a parameter k, and a precision

Algorithm 4: TopPPR

```
Input: Graph $G$, source node $s$, decay factor $\alpha$, precision $\rho$,
       parameter $k$
Output: $T_k(s) = \{t_1, \dots, t_k\}$, the exact top-$k$ node set of $s$
1  $V_k \leftarrow \emptyset, C \leftarrow V$;
2  $n_r \leftarrow 4\sqrt{mn\log n}$, $r^f_{max} \leftarrow \frac{4}{\sqrt{mn\log n}}$;
3  $[r^f, \pi^f] \leftarrow$ ForwardSearch($r^f_{max}$);
4  $[\hat{\pi}_b, V_k, C] \leftarrow$ CandidateUpdate($r^f, \pi^f, V_k, C$);
5  $n_r \leftarrow \frac{n\log n}{|C|}$, $r^b_{max} \leftarrow \frac{1}{\sqrt{m}}$, $r^f_{max} \leftarrow \frac{1}{m}$;
6  while $|V_k| < \rho k$ do
7      $[r^b, \pi^b] \leftarrow$ GroupBackwardSearch($r^b_{max}, \hat{\pi}_b$);
8      $[r^f, \pi^f] \leftarrow$ ForwardSearch($r^f_{max}$);
9      $[\hat{\pi}, V_k, C] \leftarrow$ CandidateUpdate($r^f, \pi^f, r^b, \pi^b, V_k, C$);
10     $r^b_{max} \leftarrow \frac{r^b_{max}}{2}$, $r^f_{max} \leftarrow \frac{r^f_{max}}{2}$, $n_r \leftarrow 2n_r$;
11 return $V_k$;
```

parameter ρ, TopPPR finds the top-k node set of s with precision at least ρ. We divide the node set V into three categories:

(1) Top-k nodes set V_k, which consists of nodes that the algorithm is confident to classify as top-k nodes;
(2) Candidate set C, which consists of nodes that the algorithm is unable to classify based on current confidence bounds;
(3) Non-top-k node set $V \setminus (V_k \cup C)$, which consists of nodes that the algorithm is confident to classify as not in the top-k node set.

Initially, we set $V_k = \emptyset$ and $C = V$ (Line 1 in Algorithm 4).

In the first phase of the algorithm, we perform forward search and random walks to obtain a rough estimation of PPR value $\pi(s,t)$ for each $t \in V$. More precisely, we first perform forward search to construct Alias structure r^f and forward reserve array π^f (Line 3). Then, we invoke CandidateUpdate to compute $\hat{\pi}_b(s,t), t \in V$ and update top-k node set V_k and candidate set C (Line 4). Note that we do not have backward residues r^b and reserves π^b yet, so by default, we set $r^b(v,t) = 1$ for $v = t$ and $r^b(v,t) = 0$ for $v \neq t$, and $\pi(s,v) = 0$ for $v \in V$. We set the number of random walks n_r and the forward threshold r^f_{max} such that with probability $1-1/n$, we obtain an estimator $\hat{\pi}_b(s,t)$ for $\pi(s,t)$ with error at most $\pi(s,t)/4$ for any $\pi(s,t) > 1/n$ (Line 2). There are two purposes of this phase: (i) we can prune most nodes in the candidate set C with CandidateUpdate; (ii) we will use the estimators $\hat{\pi}_b(s,t), t \in V$ in the group backward search to set the threshold $r^b_{max}(v,t)$.

In the next phase, we adaptively perform forward search, random walk sampling, and group backward search to refine C and to construct V_k. We first set $n_r = \frac{n\log n}{|C|}$, $r^b_{max} = \frac{1}{\sqrt{m}}$, and $r^f_{max} = \frac{1}{m}$ (Line 5). In each iteration, we perform group backward search for C with threshold r^b_{max} to construct r^b, the inverted list of the backward residues, and π^b, the inverted list of backward reserves (Line 7). We also perform forward search with r^f_{max}, and we construct r^f, an Alias structure of forward residue, and π^f, an array of forward reserves (Line 8). Then, we call the CandidateUpdate algorithm to perform random walk sampling and update candidate set C and top-k set V_k (Line 9). If the number of nodes in V_k exceeds ρk, it

Algorithm 5: CandidateUpdate

```
Input: Graph $G$, source node $s$, decay factor $\alpha$, forward reserves $\pi^f$,
       Alias structure of forward residues $r^f$, inverted lists of
       backward reserves $\pi^b$ and residues $r^b$, candidate set $C$, top-$k$
       set $V_k$, number of random samples $n_r$
Output: Updated candidate set $C$ and top-$k$ set $V_k$
1  for $j$ from 1 to $n_r$ do
2      Sample a node $u$ from Alias structure $r^f$ with probability
       $r^f(s,u)/r^f_{sum}$ and set $v \leftarrow u$;
3      while $rand() > 1-\alpha$ do
4          Uniformly pick $w$ from $N^{out}(v)$ and set $v \leftarrow w$;
5      for each $t$ with non-zero $r^b(v,t)$ do
6          $\hat{\pi}(s,t) \leftarrow \frac{j-1}{j}\hat{\pi}(s,t) + \frac{1}{j}r^f_{sum} \cdot r^b(v,t)$;
7          $\bar{\sigma}^2(s,t) \leftarrow \frac{j-1}{j}\bar{\sigma}^2(s,t) + \frac{1}{j}{r^f_{sum}}^2 \cdot r^b(v,t)^2$;
8  for each node $t \in C$ do
9      $\hat{\pi}(s,t) \leftarrow \hat{\pi}(s,t) + \pi^f(s,t) + \sum_{u\in V} r^f(s,u)\pi^b(u,t)$;
10     $\beta(s,t) = \sqrt{\frac{2\bar{\sigma}^2(s,t)\ln(3n^3\log^2 n_r)}{n_r}} + \frac{3r^f_{sum}\ln(3n^3\log^2 n_r)}{n_r}$;
11     if Number of $t' \in C$ such that
       $\hat{\pi}(s,t') + \beta(s,t') < \hat{\pi}(s,t) - \beta(s,t)$ exceeds $|C| + |V_k| - k$
       then
12         $V_k \leftarrow V_k \cap \{t\}, C \leftarrow C \setminus \{t\}$;
13     else if Number of $t' \in C$ such that
       $\hat{\pi}(s,t') - \beta(s,t') > \hat{\pi}(s,t) + \beta(s,t)$ exceeds $k - |V_k|$ then
14         $C \leftarrow C \setminus \{t\}$;
```

implies that the algorithm has found at least ρk top-k nodes, in which case we can return V_k as the results. Note that if $|V_k| < \rho k$ (Line 6), we can report some nodes in C as the top-k nodes (Line 11). If after this iteration, the number of nodes in V_k is still smaller than ρk, we halve r^f_{max} and r^b_{max}, and double the number of random walks n_r (Line 10), such that the forward search, group backward search, and random walk sampling take approximately twice as much time in the next iteration.

Updating the Candidate Set. The CandidateUpdate algorithm uses random walk samples to updates the candidate set C and top-k set V_k. The algorithm starts by performing n_r random walk samples. For each random walk, we sample a node u from the Alias structure r^f according to probability $r^f(s,u)/r^f_{sum}$ (Line 2), and perform random walk with termination probability α from u (Lines 3-4). If the random walk terminates at node v, we update the estimator $\hat{\pi}(v,t)$ and empirical variance $\bar{\sigma}^2(s,t)$ for each node $t \in C$ with non-zero backward residue $r^b(v,t)$ (Lines 6-7). Note that r^b is maintained as an inverted list, so we can retrieve the nodes with non-zero backward residue $r^b(v,t)$ in linear time.

After all random walks are processed, we compute the estimator $\hat{\pi}(s,t)$ (Line 9) and confidence bound $\beta(s,t)$ for each $t \in C$ (Line 10). Note that to compute $\sum_{u\in V} r^f(s,u)\pi^b(u,t)$, we do not have to go through all $u \in V$, as the algorithm maintains an inverted list for $\pi^b(u,t)$ that allows us to find all non-zero $\pi^b(u,t)$ in linear time for a given u. Finally, we update the top-k node set V_k and the candidate set C as follows. For each node $t \in C$, we count the number of nodes t' in C with upper confidence bounds $\hat{\pi}(s,t') + \beta(t')$ that are less or equal to the lower confidence bound $\hat{\pi}(s,t) - \beta(t)$ of node t (Line 11). If this number exceeds $|C| + |V_k| - k$, it implies that with high probability, there are more than $n-k$ nodes with PPR values that are lower than $\pi(s,t)$, and thus t is a true top-k node. Note that there already are $n - (|C| + |V_k|)$ nodes that were evicted from C, which all have PPR values smaller than $\pi(s,t)$. In this case, we move t from C to top-k node set V_k (Line 12). On the other hand, we also count the number of nodes nodes t' in C with lower confidence bounds $\hat{\pi}(s,t') - \beta(t')$ that are higher or equal to the upper confidence bound $\hat{\pi}(s,t) + \beta(t)$ of node t (Line 13). If this number exceeds $k - |V_k|$, it implies that with high probability, there are more than k nodes with PPR values that are higher than $\pi(s,t)$, and thus t is a true non-top-k node. Note that there already are $|V_k|$ nodes in V_k with PPR values that are larger than $\pi(s,t)$. In this case, we evict t from C (Line 14).

4 ANALYSIS

In this section, we analyze the correctness and time complexity of the TopPPR algorithm. The proofs of all theorems and lemmas are included in the appendix.

4.1 Correctness

We first prove that with high probability, TopPPR answers a ρ-precise top-k PPR query correctly. The following Lemma shows that the estimators of TopPPR are unbiased.

Lemma 4.1. *Consider an iteration of TopPPR. For any $t \in C$, we have $E[\hat{\pi}(s,t)] = \pi(s,t)$.*

Using Lemma 4.1 and the empirical Bernstein inequality, we can prove the the following lemma, which states that TopPPR computes all confidence bounds correctly with high probability.

Lemma 4.2. *With probability $1 - 1/n^3$, at any iteration of the TopPPR, we have $\pi(s,t) \in [\hat{\pi}(s,t) - \beta(s,t), \hat{\pi}(s,t) + \beta(s,t)]$ for any $t \in V$.*

From Lemma 4.2, we can derive the following lemma, which states that with high probability, the nodes in V_k are true top-k nodes and the nodes evicted from C are true non-top-k nodes.

Lemma 4.3. *With probability $1 - 1/n^3$, at any iteration of the TopPPR, if a node t is moved from C to V_k, then there are at least $n-k$ nodes with PPR values less or equal to $\pi(s,t)$. If a node t is evicted from C, then there are at least k nodes with PPR values larger or equal to $\pi(s,t)$.*

By Lemma 4.3, at the end of TopPPR, with high probability, there are at least $|V_k| \geq \rho k$ true top-k nodes in V_k, and thus the precision of TopPPR is at least ρ.

Theorem 4.4. *Given a source node s, parameter k and precision ρ, TopPPR returns the top-k node set of s with precision at least ρ, with probability at least $1 - 1/n^3$.*

By applying union bound over all possible source nodes in V and all possible choices of k, we have with probability at least $1 - 1/n$, the TopPPR algorithm returns the top-k node set with precision at least ρ for any source node s and any choice of k.

4.2 Time Complexity

We first analyze the worst-case time complexity of group backward search and forward search. Then, we bound the number of random walk samples needed for the algorithm to stop. Finally, we combine the cost of group backward search, forward search, and random walk sampling in worst case model and in power law graph model.

Group Backward Search. The following lemma bounds the total cost of the group backward search.

Lemma 4.5. *Let r^b_{max} be the threshold of the group backward search in the last iteration of TopPPR. The total cost of the group backward search of TopPPR is bounded by $O\left(\frac{\sqrt{m}}{r^b_{max}}\right)$.*

Forward Search. The following lemma bounds the cost of the forward search.

Lemma 4.6. *Let r^f_{max} be the forward residue threshold used in the last iteration of TopPPR. The total cost of the forward search is bounded by $O\left(1/r^f_{max}\right)$.*

Random Walk Sampling. To bound the number of random walk samples, we need Lemma 4.7, which gives the stopping condition of the TopPPR algorithm based on a constant gap_ρ. The lemma uses a constant gap_ρ, which is defined as the difference between the $\lceil \rho k \rceil$-th and $(k+1)$-th largest PPR with respect to s, i.e.,

$$gap_\rho = \pi(s, t_{\lceil \rho k \rceil}) - \pi(s, t_{k+1}).$$

Lemma 4.7. *Consider an iteration of TopPPR. If $\beta(s,t) \leq \frac{1}{4} gap_\rho$ for any $t \in C$, then the TopPPR algorithm will stop at the end of the iteration.*

Lemma 4.7 suggests that TopPPR stops if it is able to approximate $\pi(s,t)$ with additive error gap_ρ for any $t \in C$. Intuitively, for any t_j with $j \leq \rho k$, $\beta(s,t_j) \leq \frac{1}{4} gap_\rho$ implies that TopPPR can estimate $\pi(s,t_j)$ with additive error at most $\frac{1}{4} gap_\rho$. This suggests that the algorithm should be able to distinguish $\pi(s,t_j)$ from $\pi(s,t_{k+1})$ and move t_j to V_k at the end of the iteration. Therefore, the top-ρk nodes are in V_k, and the algorithm will stop. The following lemma bounds the number of random walks needed for the algorithm to stop.

Lemma 4.8. *Consider an iteration of TopPPR and assume $r^b_{max} \geq \sqrt{\frac{gap_\rho}{m}}$. If the number of walks n_r exceeds $\Omega\left(\frac{m \cdot r^f_{sum} \cdot \left(r^b_{max}\right)^2}{gap^2_\rho} \log n\right)$, then for any $t \in C$, $\beta(s,t) \leq \frac{1}{4} gap_\rho$.*

Total Query Cost on Worst-Case Graphs. To bound the worst case query cost of TopPPR, we note that for each random walk, it visits at most $|C| = O(n)$ nodes with non-zero backward residues. Therefore the cost for random walk sampling can be bounded as $O\left(\frac{nm \cdot r^f_{sum} \cdot \left(r^b_{max}\right)^2}{gap^2_\rho} \log n\right)$. Accordingly, we have the following theorem.

Theorem 4.9. *The expected cost of TopPPR on worse-case graphs is $O\left(\frac{m + n\log n}{\sqrt{gap_\rho}}\right)$.*

Total Query Cost on Power-Law Graphs. We note that in order for the worst-case to happen, the candidate C must remain $\Theta(n)$ after the initial pruning phase. This is rarely the case in practice, as the first phase is usually able to prune most nodes from C. To capture this essence, we analyze the cost of TopPPR under the power-law graph model. It is observed in [12] that the PPR values on power-law graphs also follow a power-law distribution. Formally, for any source node s, one can assume that $\pi(s, t_k) = \Theta\left(\frac{k^{-\gamma}}{n^{1-\gamma}}\right)$, where $0 < \gamma < 1$ is the extent of the power law. This assumption has been adapt in previous work on top-k PPR queries [12, 30]. In our analysis, we only need the following assumption.

Assumption 1 (Power-Law graph). *Givenv a source node s, the number of nodes with PPR values in $\left[\frac{1}{4}\pi(s, t_k), 4\pi(s, t_k)\right]$ is $O(k)$.*

Note that Assumption 1 is weaker than assuming a power-law distribution of the PPR values, as $\pi(s, t_k) = \Theta\left(\frac{k^{-\gamma}}{n^{1-\gamma}}\right)$ for $k = 1, \ldots, n$ directly implies Assumption 1. We have the following theorem that bounds the total query cost of TopPPR given the power-law assumption.

Theorem 4.10. *The expected cost of TopPPR algorithm on power-law graphs is $O\left(\frac{k^{\frac{1}{4}} n^{\frac{3}{4}} \log n}{\sqrt{gap_\rho}}\right)$.*

Remark. Notice that Theorems 4.9 and 4.10 implicitly assume $gap_\rho > 0$. When $gap_\rho = 0$ (i.e., when the $\lceil \rho k \rceil$-th and $(k+1)$-th largest PPR are identical), TopPPR does not terminate, since it can never distinguish the $\lceil \rho k \rceil$-th node from the $(k+1)$-th one. To tackle this pathological case of $gap_\rho = 0$, we introduce a constant $gap_{min} = 10^{-10}$, and we let TopPPR terminates whenever $\beta(s, t) \leq \frac{1}{4} gap_{min}$ for all $t \in C$. This, by Lemma 4.3, ensures that gap_{min} is an upper bound of the absolute errors in the PPR estimations that TopPPR makes on the candidate set C. In other words, by setting $gap_{min} = 10^{-10}$, TopPPR achieves an absolute error of at most 10^{-10} for the top-k results. This is in accordance with the standard practice in the literature [27, 40] that considers PPR estimations with at most 10^{-10} absolute error as ground truths. When incorporating gap_{min}, the query time of TopPPR is bounded by $O\left(\frac{m+n\log n}{\sqrt{gap_{min}}}\right)$ on worst-case graphs and by $O\left(\frac{k^{\frac{1}{4}} n^{\frac{3}{4}} \log n}{\sqrt{gap_{min}}}\right)$ on power law graphs.

5 $(\sqrt{1-\alpha})$-WALKS

In this section, we propose a technique called $(\sqrt{1-\alpha})$-*walk*. This technique is used to improve the real-world performance of TopPPR, and is potentially of independent interest.

Recall that a random walk with restart from s is a traversal of G that starts from s and, at each step, either (i) terminates at the current node with α probability, or (ii) proceeds to a randomly selected out-neighbor of the current node. For any node $t \in V$, the personalized PageRank (PPR) $\pi(s, t)$ of t with respect to s is then the probability that a random walk from s terminates at t. For a single random walk sample, suppose we set $\hat{\pi}(s, t) = 1$ if the random walk terminates at node t and $\hat{\pi}(s, t) = 0$ otherwise, then $E[\hat{\pi}(s, t)] = \pi(s, t)$.

A $(\sqrt{1-\alpha})$-walk from s is a traversal of G that starts from s and, at each step, either (i) terminates at the current node with $1 - \sqrt{1-\alpha}$ probability, or (ii) proceeds to a randomly selected out-neighbor of the current node. Fix a node $t \in V$, if the $(\sqrt{1-\alpha})$-walk visits t at the i-th step, we add $\alpha(\sqrt{1-\alpha})^i$ to $\hat{\pi}(s, t)$. The following Lemma states that the expected value of $\hat{\pi}(s, t)$ is the PPR value $\pi(s, t)$.

Lemma 5.1. *Let $\hat{\pi}(s, t)$ denote the random variable computed by a single $(\sqrt{1-\alpha})$-walk from s. The expected value of $\hat{\pi}(s, t)$ is $\pi(s, t)$.*

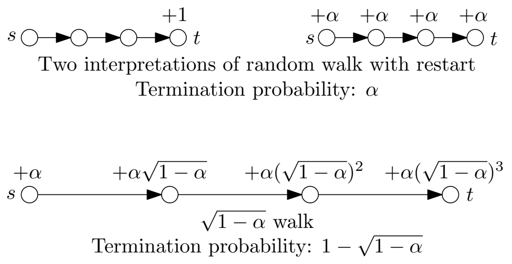

+1
s
t
+α +α +α +α
s
t
Two interpretations of random walk with restart
Termination probability: α
+α
+α√(1−α)
+α(√(1−α))²
+α(√(1−α))³
s
t
√(1−α) walk
Termination probability: 1 − √(1−α)

Figure 2: Random walk with restart and $\sqrt{1-\alpha}$ walks

Intuitions of $(\sqrt{1-\alpha})$-Walk. There are two intuitions for using the $(\sqrt{1-\alpha})$-walk instead of the normal random walk with restart. First of all, a sample in random walk with restart only updates the estimator of the termination node, which means the other nodes in the walk are "wasted". On the other hand, a $(\sqrt{1-\alpha})$-walk is able to update the estimator of each node visited.

Secondly, we note that in random walk with restart, each sample will add a score of 1 to $\hat{\pi}(s, t)$ where t is the termination node. On the other hand, the $(\sqrt{1-\alpha})$-walk add a score of $(\sqrt{1-\alpha})^i \alpha$ to the i-th node on the walk. On real-world graphs, a $(\sqrt{1-\alpha})$ walk rarely visit the same node twice, which means the variance of the estimator provided by $(\sqrt{1-\alpha})$-walk is usually α^2 smaller than that of random walk with restart.

Finally, we note that there is another possible interpretation for the personalized pagerank: we still use random walk with termination probability α, but for each visited node t, we add a score of α to $\hat{\pi}(s, t)$. See Figure 2 for an illustration. Using arguments similar to the proof of Lemma 5.1, we can prove that $\hat{\pi}(s, t)$ is also an unbiased estimator of $\pi(s, t)$. This approach will also reduce the variance of the estimator by a factor of α^2. However, the drawback of this interpretation is that it may lead to unbounded estimators. More precisely, consider a graph with a single node s and a self loop from s to s. If the random walk from s takes unbounded number of steps, then the score added to $\hat{\pi}(s, s)$ in a single sample is unbounded. This will violate the conditions for applying concentration inequalities such as Chernoff bounds or Bernstein inequalities. On the other hand, in $(\sqrt{1-\alpha})$-walk, the scored added to $\hat{\pi}(s, s)$ is bounded by $\sum_{i=0}^{\infty} \alpha(\sqrt{1-\alpha})^i = 1 + \sqrt{1-\alpha} \leq 2$, and thus we are able to use Chernoff inequality or Bernstein inequality to analyze its concentration behavior.

6 OTHER RELATED WORK

Among the large number of existing studies on PPR [7, 8, 10–13, 15–22, 26, 27, 29–33, 35–37, 40–45], the ones most related to ours focus on algorithms for exact top-k PPR queries [15, 17–20, 41, 42]. Almost all of these algorithms [17–20, 41, 42] are based on local updates, and their basic idea is to (i) traverse the input graph G from the source node s while maintaining lower and upper bounds of each node's PPR, and (ii) terminate the traversal once the lower and upper bounds can pinpoint the exact top-k results. The only exception is *Chopper* [15], which is a matrix-based technique that utilizes Chebyshev polynomials for acceleration. As with TopPPR for $\rho = 1$, these methods implicitly require that $gap_\rho > 0$; to address the pathological case of $gap_\rho = 0$, they also allow a (very) small amount gap_{min} of absolute error in PPR estimations. In our experiments, we compare the state of the art among these methods [15, 41] against TopPPR with $\rho = 1$.

There are also several methods [29–32, 39, 40, 45] for approximate top-k PPR queries. Among them, *BiPPR* [30], *HubPPR* [39] and *FORA* [40] are the state-of-the-art approximate PPR algorithms. In our setting, the three algorithms ensure a relative error of at most ϵ_r for any PPR value larger than $1/n$, with probability at least $1 - 1/n$. It is shown in [40] that the query cost of these three algorithms is $O(\frac{1}{\epsilon_r} n \log n)$. We note that this bound is not comparable to the query cost of TopPPR, as none of the above approximate algorithms are able to provide non-trivial guarantees on the precision of the results. Our experiments in Section 7 show that these methods [30, 39, 40] are inferior to TopPPR in terms of the trade-off between precision and efficiency.

In addition, considerable efforts [13, 27, 32, 34, 37, 45] have been made to investigate algorithms for single-source PPR queries. The methods proposed are mostly built upon the *power method* [34], which is a matrix-based iterative algorithm that can answer single-source PPR queries with any given threshold ϵ_a on the absolute errors of PPR estimations. Observe that if we set ϵ_a to a sufficient small value, then we can obtain very accurate PPR estimations from a single-source method, based on which we can derive precise results for top-k PPR queries. Therefore, we include the state-of-the-art single-source PPR algorithm [27] in our experiments on exact top-k PPR queries.

Finally, it is worth mentioning that existing work has studied other variants of PPR queries, such as point-to-point PPR queries [16, 29–31, 39], PPR queries on dynamic graphs [12, 13, 33, 35, 43, 44], distributed algorithms for PPR [11, 21]. These studies, however, are orthogonal to our work.

7 EXPERIMENTS

This section experimentally evaluates the proposed solutions against the states of the art. All experiments are conducted on a machine with a Xeon(R) CPU E5-2620@2.10GHz CPU and 96GB memory.

7.1 Experimental Settings

Methods. Table 2 summarizes the methods evaluated in our experiments. We compare TopPPR with six algorithms: FLoS_RWR, Chopper, FORA, FORA+, HubPPR, and BiPPR. As mentioned in Section 1, FLoS_RWR and Chopper are the state-of-the-art algorithms for exact top-k PPR queries. We also include BePI, the most advanced index-based algorithm for single-source PPR queries, as a baseline solution for exact top-k queries. Note that BePI computes the PPR value $\pi(s,t)$ with a tiny error for each $t \in V$, and thus, is able to return the exact top-k nodes in correct order. For experiments on approximate top-k PPR queries, we compare TopPPR against FORA and BiPPR, the state-of-the-art index-free algorithms for approximate top-k PPR queries. We also include two index-based algorithms, FORA+ and HubPPR. We obtain the codes of Chopper from [2], FLoS_RWR from [3], BePI from [4], and we implement all other algorithms in C++.

Table 2: Methods.

Methods	Exactness	Preprocessing?	Reference
TopPPR	Exact & Approximate	No	Our method
FLoS_RWR	Exact	No	[41]
Chopper	Exact	No	[15]
BePI	Exact	Yes	[27]
FORA+	Approximate	Yes	[40]
FORA	Approximate	No	[40]
HubPPR	Approximate	Yes	[39]
BiPPR	Approximate	No	[30]

Table 3: Datasets.

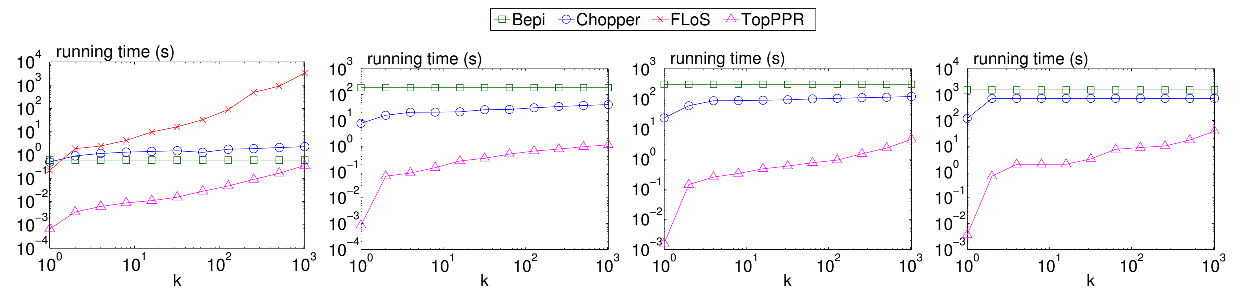
Dataset	**Type**	***n***	***m***
DBLP	undirected	317,080	1,049,866
LiveJournal	undirected	3,997,962	34,681,189
Orkut	undirected	3,072,441	117,185,083
Twitter	directed	41,652,230	1,468,365,182

Datasets and Metrics. We use 4 benchmark datasets that are obtained from public sources [5, 6] and are frequently used in previous work, as shown in Table 3. On each dataset, we randomly select 100 query nodes, and apply the power method [34] with 100 iterations to compute the ground-truth PPR values. This ensures that each ground-truth value has at most 10^{-10} absolute error. For each query node, we use the top-k nodes computed by the power method as the ground truth node set. We set $k = 1, 2, 4, 8, \ldots, 1024$, which is the typical setting for top-k PPR queries [40]. Following previous work [30, 31, 39, 40], we set the decay factor α of PPR to 0.2.

We evaluate the accuracy of each method using two classic metrics for evaluating ranking results: *precision* and *Normalized Discounted Cumulative Gain (NDCG)* [25]. Specifically, given a query node s, let $V_k = \{t_1, \ldots, t_k\}$ denote the ground-truth top-k node set, and $V'_k = \{t'_1, \ldots, t'_k\}$ denote the top-k node set returned by the algorithm to be evaluated. The precision of V'_k is defined as $\frac{|V_k \cap V'_k|}{k}$. The NDCG of V'_k, on the other hand, evaluates whether V'_k orders the important nodes correctly, and it is defined as $\frac{1}{Z_k}\sum_{i=1}^{k} \frac{2^{\pi(s,t'_i)}-1}{\log(i+1)}$, where $Z_k = \sum_{i=1}^{k} \frac{2^{\pi(s,t_i)-1}}{\log(i+1)}$.

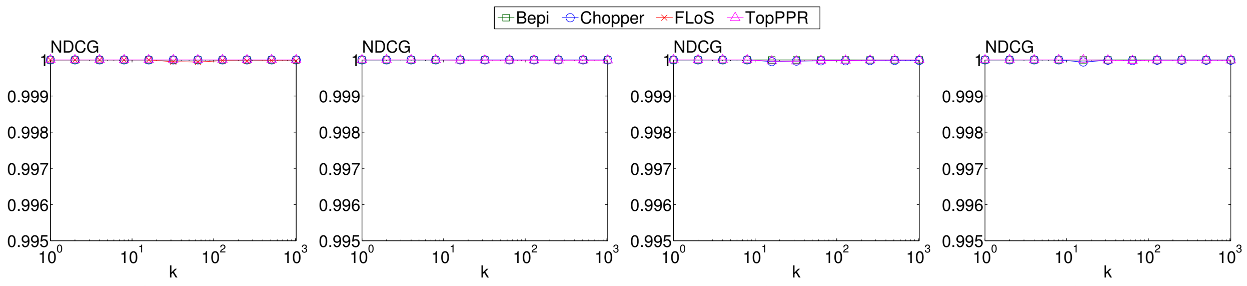
7.2 Exact Top-k PPR Queries

In our first set of experiments, we evaluate the efficiency of each method for exact top-k PPR queries. We compare TopPPR with FLoS_RWR and Chopper, two state-of-the-art algorithms for exact top-k PPR, and with BePI, the most advanced method for single-source PPR. We set $\rho = 1$ for TopPPR to guarantee the exactness of the top-k results with high probability. FLoS_RWR has two internal parameters τ and u, where τ is the error allowed for the iterative method used by FLoS_RWR as a subroutine, and u is the absolute error allowed for the PPR estimators. In accordance with the settings in [41], we set $\tau = u = 10^{-5}$. BePI has a internal parameter ε, which is the error allowed for the iterative methods used in BePI. We set $\varepsilon = 10^{-10}$, following the experiments in [27].



(a) DBLP (b) LiveJournal (c) Orkut (d) Twitter

Figure 3: Query time v.s. *k* for exact top-*k* PPR queries



(a) DBLP (b) LiveJournal (c) Orkut (d) Twitter

Figure 4: NDCG for exact top-*k* PPR queries

Table 4: Average precision for exact top-*k* queries.

Methods	*DBLP*	*LiveJournal*	*Orkut*	*Twitter*
TopPPR	1	1	1	1
BePI	1	1	1	1
Chopper	1	1	1	1
FLoS_RWR	0.99994	N/A	N/A	N/A

Query Time. Figure 3 reports the average query time of each method on all four datasets. Note that both the x-axis and y-axis are in log-scale. FLoS_RWR requires more than 3600 seconds per query on datasets larger than *DBLP*, and is thus excluded from the experiments on *LiveJournal*, *Orkut* and *Twitter*. We first observe that TopPPR outperforms the competitors by 2 to 3 orders of magnitude. Notably, TopPPR is able to answer an exact top-512 query on *Twitter* in 15 seconds, while the closest competitor, Chopper, takes 1500 seconds to answer a query. The reason is that (i) BePI and Chopper use matrix-based methods to compute PPR values, which leads to significant query cost, and (ii) FLoS_RWR uses a less-optimized version of local search, such that when the decay factor $\alpha = 0.2$, it requires traversing a large number of edges in each of its iterations, which results in inferior efficiency.

Precision and NDCG. Table 4 reports the precision of each method on the four datasets. For each graph, we perform 100 top-k queries for $k = 1, 2, 4, 8, \ldots, 1024$, and take the average precision over these queries. We observe that by setting $\rho = 1$, TopPPR indeed achieves precision 1. In addition, Chopper, and BePI are also able to answer all queries exactly. Meanwhile, FLoS_RWR achieves a precision slightly less than 1 on *DBLP*. This is because it allows an absolute error of 10^{-5} in PPR estimations, which is larger than the difference between the k-th and $(k+1)$-th largest PPR values on *DBLP* when k is large.

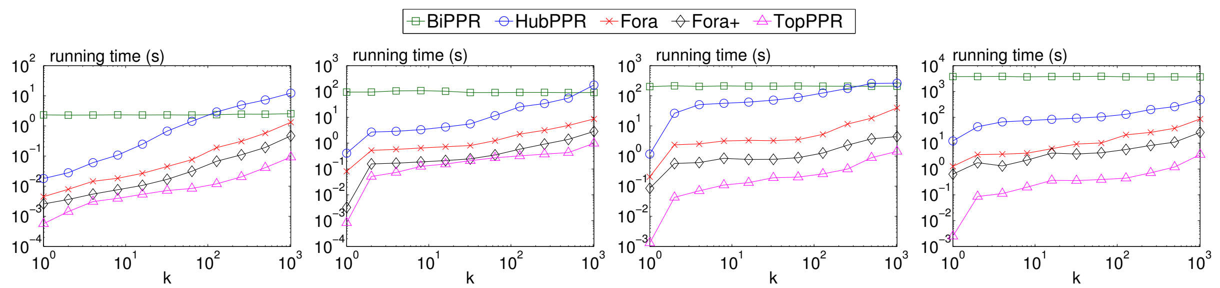
Figure 4 illustrates the NDCG of each method. Notice that TopPPR achieves near-perfect NDCG on all four graphs, even though it does not provide formal guarantees on the order of the top-k results. The other three methods also offer high NDCG.

7.3 Approximate Top-k PPR Queries

In our second set of experiments, we evaluate the efficiency and accuracy of each method for approximate top-k PPR queries. We compare TopPPR against FORA, FORA+, HubPPR, and BiPPR, which are the state of the art for approximate top-k PPR. We set $\rho = 0.99$ for TopPPR to guarantee a precision of at least 0.99. Following [30, 31, 39], we set $\delta = 1/n, p_f = 1/n$, and $\varepsilon_r = 0.5$ for FORA, FORA+, HubPPR, and BiPPR.

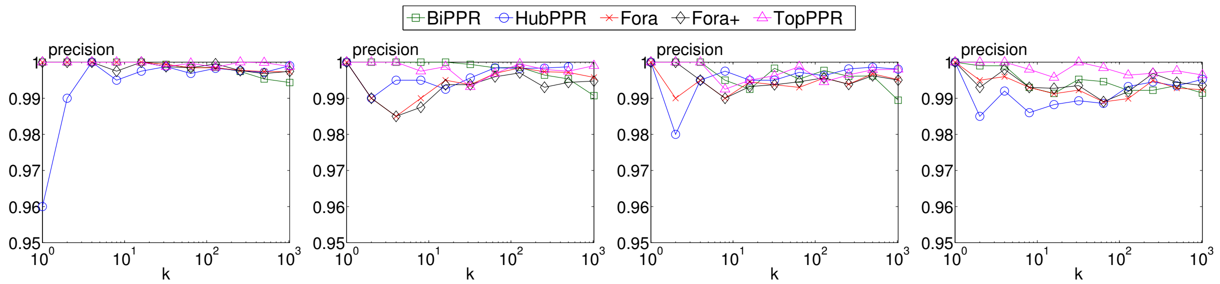
Query Time. Figure 5 reports the average query time of each method. Note that both the x- and y-axis are in log-scale. We first observe that TopPPR outperforms all competitors in terms of query time. In particular, TopPPR processes a top-512 PPR query on *Twitter* in 2 seconds on average, while the closest index-free competitor FORA uses 30 seconds. Furthermore, TopPPR also outperforms FORA+, which requires hours of preprocessing time on *Twitter* and index size several times of the original dataset (See Table 5).

Precision. Figure 6 shows the precision of each method on each dataset. In general, all approximate methods achieve high precision. In particular, the precision of TopPPR is always higher than the required $\rho = 0.99$. Meanwhile, the precisions of FORA, FORA+, and BiPPR are dominated by that of TopPPR, while HubPPR achieves lower precision than TopPPR does on all datasets but *Orkut*. This makes TopPPR a more preferable method than FORA, FORA+,



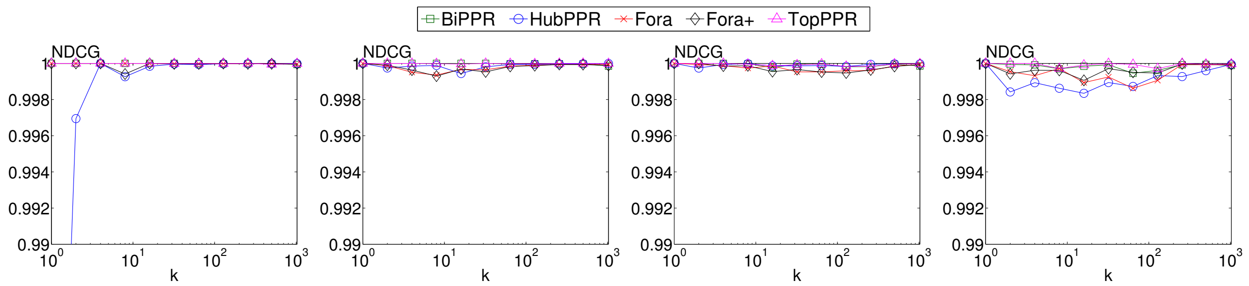
(a) DBLP (b) LiveJournal (c) Orkut (d) Twitter

Figure 5: Query time for approximate top-*k* PPR queries



(a) DBLP (b) LiveJournal (c) Orkut (d) Twitter

Figure 6: Precision for approximate top-*k* PPR queries



(a) DBLP (b) LiveJournal (c) Orkut (d) Twitter

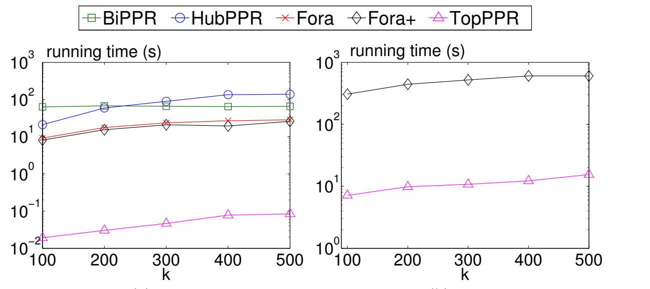
Figure 7: NDCG for approximate top-*k* PPR queries

BiPPR, and HubPPR, since it outperforms the latter in terms of efficiency, accuracy, and space consumption.

NDCG. Figure 7 illustrates the *NDCG* of each method. Observe that TopPPR offers high NDCGs on all graphs. In contrast, the NDCGs of BiPPR and HubPPR are notably worse than that of TopPPR on all datasets but *DBLP*, while FORA and FORA+ are inferior to TopPPR on *Orkut* and *Twitter*.

7.4 Exact Top-k PPR by Approximate Methods

From Figure 6, we observe that BiPPR, HubPPR, FORA, and FORA+ all achieve relatively high precisions for top-k queries. As such, a natural question is: if we allow these methods to spend more query time, can they achieve a precision of 1 and therefore be able to answer exact top-k queries? To answer this question, we design the following experiment that evaluates the time needed for each approximate method to improve its precision to 1. For each method, we repeatedly double the query time by resetting its error parameter ε_r, and report the first query time for which the method's precision



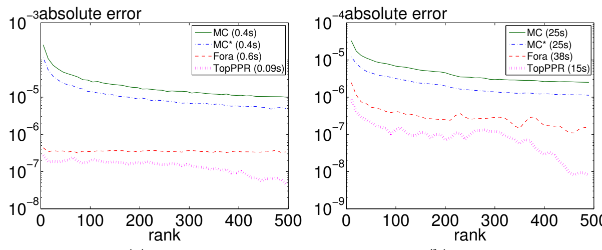
(a) DBLP (b) Twitter

Figure 8: Query time for approximate methods to achieve precision 1.

reaches 1. Figure 8 reports the query time needed for FORA, FORA+, HubPPR, and BiPPR on *DBLP* and *Twitter* under this setting. For comparison, we also include TopPPR with $\rho = 1$. We omit BiPPR, HubPPR and FORA from the experiments on *Twitter*, as they are unable to achieve 100% precision when their query time is within 1000 seconds. We observe that TopPPR outperforms the closest

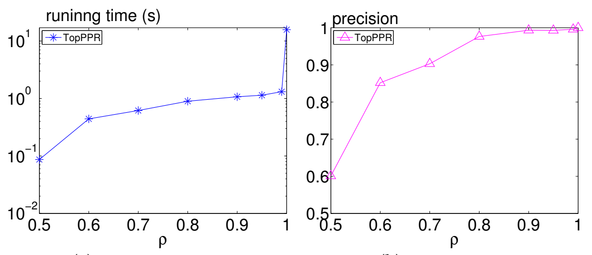
Table 5: Space overheads and preprocessing costs for index-based algorithms.

Dataset	Preprocessing Time (seconds)			Space Overhead (GBs)			Graph Size
	FORA+	HubPPR	BePI	FORA+	HubPPR	BePI	
DBLP	6.4	30.2	2.95	0.074 GB	0.22GB	0.014 GB	0.036 GB
LiveJournal	100.2	323.8	10574.3	1.87 GB	5.4 GB	61.6GB	0.5 GB
Orkut	234.1	606.7	3578.2	2.9 GB	13.8GB	34.0 GB	1.2 GB
Twitter	3572.4	5498.5	13528.4	21.2 GB	40.0 GB	81.5 GB	12.6 GB



(a) DBLP (b) Twitter

Figure 9: Absolute error v.s. Rank.



(a) Query time v.s. ρ (b) Precision v.s.ρ

Figure 10: Varying ρ.

competitor by at least two orders of magnitude. This indicates that existing approximate methods are not suitable for exact queries.

7.5 Absolute Error Analysis

In our next set of experiments, we analyze the absolute error of individual PPR estimates of TopPPR. We compare TopPPR with FORA, the state-of-the-art index-free approximate PPR algorithm. We also include Monte Carlo algorithm (MC) and Monte Carlo algorithm with $(\sqrt{1-\alpha})$-walks (MC*), for evaluating the effectiveness of $(\sqrt{1-\alpha})$-walks. For each method, we issue 100 different top-k queries for $k = 500$, and report the average absolute error of the estimator for the j-th largest PPR, for $1 \leq j \leq 500$.Following previous settings, we set $\delta = 1/n, p_f = 1/n$, and $\varepsilon_r = 0.5$ for FORA. We set the number of random walks n_r for MC and MC* to be the same as FORA, so that we can evaluate the effectiveness of the forward search by FORA. For a fair comparison, we set $\rho = 1$ for TopPPR so that the query time of TopPPR is close to that of FORA.

Figure 9 shows the average absolute error as well as the average query time of each method on *DBLP* and *Twitter*. We make the following observations from Figure 9: (i) By employing forward search, FORA significantly outperforms MC in terms of accuracy; (ii) $(\sqrt{1-\alpha})$-walks improves the original Monte Carlo algorithm by a factor of 2 to 4 times, which concurs with our theoretical analysis. (iii) By employing $(\sqrt{1-\alpha})$-walks and group backward search, TopPPR outperforms FORA for all top-500 nodes, with less query time. (iv) The gap between TopPPR and FORA becomes larger as rank i approach 500, which proves the effectiveness of the group backward search algorithm. In general, forward search and $(\sqrt{1-\alpha})$-walks are two useful techniques for improving accuracy for ALL top-k nodes, while group backward search significantly reduces the absolute error for nodes that are close to top-500, which is crucial for achieving a precision of 1.

7.6 Effects of parameter ρ

Figure 10 shows the how the query time and precision of TopPPR change when we vary the parameter ρ. We perform 100 top-500 PPR queries with TopPPR for $\rho = 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99, 1$ on *Twitter*, and report the average query time and precision in Figure 10. We observe that for $\rho = 0.5$, TopPPR achieves a precision of 0.56 using a query time of 0.09 seconds. As we increase ρ, TopPPR provides provides more accurate top-k results with longer query time. For $\rho = 1$, TopPPR is able to return the exact top-500 nodes with a query time of 15 seconds.

7.7 Preprocessing Time and Space Overhead.

Table 5 shows the preprocessing time and space overhead for each index-based algorithm. Note that TopPPR completely avoids preprocessing and space overheads, which makes it more preferable for large graphs that are subject to frequent updates.

8 CONCLUSIONS

This paper presents TopPPR, an algorithm that enables fast responses for exact top-k PPR queries on large graphs, and naturally supports dynamic graphs as it does not require any index. TopPPR answers any top-k PPR query in $O\left((m + n\log n)/\sqrt{gap_\rho}\right)$ expected time, and it ensures that, with $1 - 1/n$ probability, a top-k node set is returned with precision at least ρ. Our experiments show that the algorithm significantly outperforms the existing methods in terms of query efficiency, accuracy, and scalability. For future work, we plan to investigate how TopPPR can be applied on massive graphs that do not fit in the main memory of a single machine.

9 ACKNOWLEDGEMENTS

This research was supported in part by National Natural Science Foundation of China (No. 61502503), by the National Key Basic Research Program (973 Program) of China (No. 2014CB340403), by the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China under Grant 18XNLG21, by MOE, Singapore under grant MOE2015-T2-2-069, and by NUS, Singapore under an SUG.

REFERENCES

[1] https://spark-summit.org/2017/events/random-walks-on-large-scale-graphs-with-apache-spark/.

[2] http://compbio.case.edu/omics/software/chopper/index.html.

[3] https://github.com/YubaoWu/FLoS.

[4] https://datalab.snu.ac.kr/bepi/.

[5] http://snap.stanford.edu/data/index.html.

[6] http://law.di.unimi.it/datasets.php.

[7] Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. Local computation of pagerank contributions. In *WAW*, pages 150–165, 2007.

[8] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Local graph partitioning using pagerank vectors. In *FOCS*, pages 475–486, 2006.

[9] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. Tuning bandit algorithms in stochastic environments. In *ALT*, volume 4754, pages 150–165. Springer, 2007.

[10] Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, pages 635–644, 2011.

[11] Bahman Bahmani, Kaushik Chakrabarti, and Dong Xin. Fast personalized pagerank on mapreduce. In *SIGMOD*, pages 973–984, 2011.

[12] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *VLDB*, 4(3):173–184, 2010.

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

[14] Fan R. K. Chung and Lincoln Lu. Survey: Concentration inequalities and martingale inequalities: A survey. *Internet Mathematics*, 3(1):79–127, 2006.

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

[16] 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.

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

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

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

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

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

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

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

[24] Ali Hadian, Sadegh Nobari, Behrooz Minaei-Bidgoli, and Qiang Qu. Roll: Fast in-memory generation of gigantic scale-free networks. In *SIGMOD*, pages 1829–1842. ACM, 2016.

[25] Kalervo Järvelin and Jaana Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR*, pages 41–48, 2000.

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

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

[28] David C. Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C. Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. Related pins at pinterest: The evolution of a real-world recommender system. In *WWW (Companion)*, pages 583–592, 2017.

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

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

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

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

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

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

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

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

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

[38] Alastair J. Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 10(8):127–128, 1974.

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

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

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

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

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

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

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

A CHERNOFF BOUND

Lemma A.1 (Chernoff Bound [14]). *Consider a set $\{x_i\}$ ($i \in [1, n_r]$) of i.i.d. random variables with mean μ and $x_i \in [0, r]$,*

$$\Pr\left[\left|\frac{1}{n_r}\sum_{i=1}^{n_r} x_i - \mu\right| \geq \varepsilon\right] \leq \exp\left(-\frac{n_r \cdot \varepsilon^2}{r(\frac{2}{3}\varepsilon + 2\mu)}\right).$$

B WALKER'S ALIAS METHOD

Walker's alias method [38] is an optimal data structure for the weighted sampling problem. In this problem, the input is a set of non-negative real numbers $w_1, \ldots, w_n$, and the goal is to build a data structure, such that we can extract an index i with probability $p_i = w_i / \sum_{j=1}^{n} w_j$. Consider n bottles, where bottle i contains some (probability) mass r_i. Initially, r_i is equal to p_i. During the algorithm, we iteratively modify the values v_i's in the following way. In each iteration, we pick a bottle i with $r_i > 1/n$ and another bottle j with $r_j < 1/n$ of mass, and pour $1/n - r_j$ probability mass of bottle i to bottle j. This makes the probability mass in bottle j equal to $1/n$. After at most n such iterations, each bottle contains $1/n$ mass that comes from at most two bottles. To draw a sample, we first uniformly sample a bottle $k \in \{1, \ldots, n\}$. Assume that the bottle k contains a probability mass p from p_i and a probability mass $1/n - p$ from p_j. We then draw a random number $z \in [0, 1/n]$, and return i as the final sample if $z < p$, and j as the final sample otherwise. It is easy to see that drawing a sample takes $O(1)$ time.

C PROOFS

C.1 Proof of Lemma 4.1

Proof. Recall that TopPPR uses the following formula to estimate $\pi(s, t)$:

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

The first two terms $\pi^f(s,t)$ and $\sum_{u\in V} r^f(s,u)\pi^b(u,t)$ are deterministically provided by the forward and backward search algorithms. Let $\mu = \sum_{u,v\in V} r^f(s,u)\pi(u,v)r^b(v,t)$, we only need to show that a random walk sample gives an unbiased estimation for μ. Fix the termination node t. Let X_i denote the estimator of the i-th random walk, and $X = \frac{1}{n_r}\sum_{i=1}^{n_r} X_i$ denote the estimator for μ. Recall that the X_i's are i.i.d. random variables computed as follow: during the i-th random walk, each node u is sampled from the Alias structure r^f with probability $r^f(s,u)/r^f_{sum}$ to be the start node, and if the random walk terminates at a node v, we set $X_i = r^f_{sum} r^b(v,t)$. By the definition of personalized pageranks, we have that the probability that this random walk terminates at v is $\pi(u,v)$. We can compute the expectation of X_i as follows:

$$E[X_i] = \sum_{u\in V}\sum_{v\in V} \frac{r^f(s,u)}{r^f_{sum}} \cdot \pi(u,v) \cdot r^f_{sum} r^b(u,v)$$
$$= \sum_{u\in V}\sum_{v\in V} r^f(s,u)\pi(u,v)r^b(v,t) = \mu.$$

Therefore, $E[X] = \frac{1}{n_r}\sum_{i=1}^{n_r} E[X_i] = \mu$, and TopPPR gives an unbiased estimator for each $\pi(s,t)$, $t \in V$. □

C.2 Proof of Lemma 4.2

Proof. Recall that the TopPPR starts with $n_0 = \frac{n\log n}{|C|}$ random walk samples, and at each iteration, doubles the number of samples. Therefore, at iteration i of TopPPR, the number of random walk samples in this iteration n_r is $2^i n_0$, and thus the total number of samples N_r is at most $2^{i+1}n_0$. Fix a target node t, recall that

$$\beta(s,t) = \sqrt{\frac{2\bar{\sigma}^2(s,t)\ln(3n^3\log^2 n_r)}{n_r}} + \frac{3r^f_{sum}\ln(3n^3\log^2 n_r)}{n_r}.$$

By the Empirical Bernstein inequality, we have $|\pi(s,t) - \hat{\pi}(s,t)| \geq \beta(s,t)$ with probability at most $\frac{1}{n^3\log^2 n_r} = \frac{1}{n^3\log^2 2^i n_0} = \frac{1}{n^3(\log n_0 + i)^2}$. By union bound over $i = 1,\ldots,\infty$, it follows that the probability that there exists a target node t in an iteration such that $|\pi(s,t) - \hat{\pi}(s,t)| \geq \beta(s,t)$ is at most $\sum_{i=0}^{\infty}\frac{1}{2n^3(\log n_0+i)^2} \leq \frac{1}{n^3}$, and the Lemma follows. □

C.3 Proof of Lemma 4.4

Proof. We prove the lemma by induction. Assume that at the end of the $(i-1)$-th iteration, all nodes in V_k are top-k nodes, and all nodes in $V\setminus(C\cup V_k)$ are non-top-k nodes. At the i-th iteration, recall that TopPPR moves a node t from C to V_k if the number of $t' \in C$ such that $\hat{\pi}(s,t') + \beta(s,t') \leq \hat{\pi}(s,t) - \beta(s,t)$ exceeds $|C| + |V_k| - k$. By Lemma 4.2, all confidence bounds are correctly computed by TopPPR, and thus the number $t' \in C$ such that $\pi(s,t') \leq \pi(s,t)$ exceeds $|C| + |V_k| - k$. This prove that there are less than k nodes in V_k and C with PPR values larger than $\pi(s,t)$. Since by the induction hyperthesis, nodes outside C and V_l are non-top-k nodes, it follows that the top-k nodes are either in C or V_k, and thus t is also a top-k node. Similarly, we can prove that if the TopPPR algorithm removes a node t from C, then there are at least k nodes with PPR values larger or equal to $\pi(s,t)$ and thus t is a non-top-k node. Therefore the induction holds and the Lemma follows. □

C.4 Proof of Lemma 4.5

We need the following lemma that bounds the relative error of the first pruning phase.

Lemma C.1 ([40]). *Let $\hat{\pi}_b(s,t)$ denote the estimator for $\pi(s,t)$ obtained by the first phase of TopPPR. With probability $1 - 1/n^3$, we have $\frac{1}{4}\pi(s,t) \leq \hat{\pi}_b(s,t) \leq 4\pi(s,t)$ for any $t \in V$ with $\pi(s,t) \geq \frac{1}{n}$, and $\hat{\pi}_b(s,t) \leq \frac{4}{n}$ for any $t \in V$ with $\pi(s,t) \leq \frac{1}{n}$.*

Proof. We sketch the proof for completeness. By the property of forward search, we have $\pi(s,t) = \pi^f(s,t) + \sum_{u\in V} r^f(s,u)\pi(u,t)$. Recall that $\pi^f(s,t)$ is deterministically computed by the forward search, and $\mu = \sum_{u\in V} r^f(s,u)\pi(u,t)$ is estimated by the following random walk process: for each walk, we set sample each node u according to probability $r^f(s,u)/r^f_{sum}$, and set $X_i = r^f_{sum}$ if the walk terminates at t and $X_i = 0$ otherwise. Let $X = \sum_{i=1}^{n_r} X_i$, we have $E[X_i] = \sum_{u\in V}\frac{r^f(s,u)}{r^f_{sum}}\pi(u,t)r^f_{sum} = \sum_{u\in V} r^f(s,u)\pi(u,t) = \mu$. We also note that by the property of the forward search,

$$r^f_{sum} = \sum_{u\in V} r^f(s,u) \leq \sum_{u\in V} r^f_{max} d_{out}(u) = m r^f_{max} = \sqrt{\frac{m}{16n\log n}}.$$

Therefore, we have $n_r = 4\sqrt{mn\log n} \geq 16 r^f_{sum} n\log n$, and thus by the Chernoff inequality

$$\Pr\left[|E[X] - \mu| \geq \frac{3}{4}\max\left\{\mu, \frac{1}{n}\right\}\right] \leq e^{\frac{-n_r\max\{\mu,\frac{1}{n}\}}{4r^f_{sum}}} \leq 1/n^3,$$

and the Lemma follows. □

Proof of Lemma 4.5. Consider the i-iteration of TopPPR, and let $r^b_{max}(i)$ denote the r^b_{max} value that is used by the group backward search in this iteration. We use $r^b_{max}(v,t) = r^b_{max}(i)\cdot\sqrt{\frac{d_{in}(v)}{\hat{\pi}_b(s,v)}}$ to denote the customized threshold for node v and target node $t \in C$. By Lemma C.1 we have $\frac{1}{4}\pi(s,v) \leq \hat{\pi}_b(s,v) \leq 4\pi(s,v)$) for $\pi(s,v) \geq 1/n$, and $\hat{\pi}_b(s,v) = 1/n$ for $\pi(s,v) < 1/n$ with probability at least $1 - 1/n^3$. Therefore, we have $\hat{\pi}(s,t) \leq 4\pi(s,t) + \frac{1}{n}$ for any $t \in V$, and thus

$$\sum_{v\in V}\hat{\pi}_b(s,t) \leq \sum_{v\in V}\left(4\pi(s,t) + \frac{1}{n}\right) = 5. \tag{2}$$

Fix a target node $t \in C$ and an internal node v. Since each backward push on v transfers at least $\alpha r^b_{max}(v,t)$ portion from its residue to its reserve, the number of time that backward pushes is performed on v is at most $\frac{\pi(v,t)}{\alpha r^b_{max}(v,t)}$. Each backward push on v visits all its in-neighbours, so the cost for v and t is at most $\frac{\pi(v,t)}{\alpha r^b_{max}(v,t)} d_{in}(v)$. Summing over all $t \in C$ and $v \in V$, and we have $cost(backward) \leq \sum_{t\in C}\sum_{v\in V}\frac{\pi(v,t)}{\alpha r^b_{max}(v,t)} d_{in}(v)$. Since $r^b_{max}(v,t) = r^b_{max}(i)\cdot\sqrt{\frac{d_{in}(v)}{\hat{\pi}_b(s,v)}}$, we have

$$cost(backward) \leq \sum_{t\in C}\sum_{v\in V}\frac{\pi(v,t)\sqrt{\hat{\pi}_b(s,v)d_{in}(v)}}{r^b_{max}(i)}$$
$$= \sum_{v\in V}\frac{\sqrt{\hat{\pi}_b(s,v)d_{in}(v)}}{r^b_{max}(i)}\sum_{t\in C}\pi(v,t) \leq \frac{1}{r^b_{max}(i)}\sum_{v\in V}\sqrt{\hat{\pi}_b(s,v)d_{in}(v)}.$$

The last equation is due to the fact that $\sum_{t\in C}\pi(v,t) \le \sum_{t\in V}\pi(v,t) = 1$. By the Cauchy-Schwarz inequality, we have

$$\left(\sum_{v\in V}\sqrt{\hat{\pi}_b(s,v)d_{in}(v)}\right)^2 \le \left(\sum_{v\in V}\left(\sqrt{\hat{\pi}_b(s,v)}\right)^2\right)\cdot\left(\sum_{v\in V}\left(\sqrt{d_{in}(v)}\right)^2\right)$$
$$= \sum_{v\in V}\hat{\pi}_b(s,v)\sum_{v\in V}d_{in}(v) \le 5m.$$

The last equation is due to the fact that $\sum_{v\in V} d_{in}(v) = m$ and equation (2). Therefore, we have $\sum_{v\in V}\sqrt{\pi(s,v)d_{in}(v)} \le \sqrt{5m}$, and thus $cost(backward) \le \frac{\sqrt{5m}}{r^b_{max}(i)}$ in this iteration. Finally, recall that at each iteration, we halve the value of r^b_{max}, it follows that the total cost of the group backward search is dominated by $O\left(\frac{\sqrt{m}}{r^b_{max}}\right)$, where r^b_{max} is the threshold used in the last iteration. □

C.5 Proof of Lemma 4.6

Proof. This lemma is proved in [40], and we sketch the proof here for completeness. Consider the i-th iteration, and let $r^f_{max}(i)$ denote the forward residue threshold . For each $u \in V$, let $r^f_{max}(s,u) = r^f_{max}(i)d_{out}(u)$ denote the maximum forward residue threshold of u. Since each forward push on u transfers at least $\alpha r^f_{max}(s,u)$ portion from its residue to its reserve, it follows that the total number of forward pushes on u is bounded by $\frac{\pi(s,u)}{\alpha r^f_{max}(s,u)}$. Note that each push visits all out-neighbours of u, and thus the total cost of the forward pushes on node u is bounded by $\frac{\pi(s,u)}{\alpha r^f_{max}(s,u)}d_{out}(u)$. Summing up over all $u \in V$ and we have $cost(forward) \le \sum_{u\in V}\frac{\pi(s,u)\cdot d_{out}(u)}{\alpha r^f_{max}(s,u)}$. Combining with $r^f_{max}(s,u) = r^f_{max}d_{out}(u)$ follows that

$$cost(forward) \le \sum_{u\in V}\frac{\pi(s,u)\cdot d_{out}(u)}{\alpha r^f_{max}\cdot d_{out}(u)}$$
$$= \sum_{u\in V}\frac{\pi(s,u)}{\alpha r^f_{max}(i)} = \frac{\sum_{u\in V}\pi(s,u)}{\alpha r^f_{max}(i)} = \frac{1}{\alpha r^f_{max}(i)}.$$

The last equation uses the fact that $\sum_{u\in V}\pi(s,u) = 1$. Finally, recall that at each iteration, the algorithm halves the value of r^f_{max}, it follows that the total cost of the forward search is dominated by the $O\left(\frac{1}{r^f_{max}}\right)$, where r^f_{max} is the threshold used in the last iteration. □

C.6 Proof of Lemma 4.7

Proof. For simplicity, we assume ρk is an integer. First, we claim that if for any $t_i \in C$ with $i \le \rho k$ and any $t_j \in C$ with $k+1 \le j \le n$, we have $\hat{\pi}(s,t_i) - \beta(s,t_i) \ge \hat{\pi}(s,t_j) + \beta(s,t_j)$, then the lemma follows. To see why this is true, note that this implies that there are at least $n-k$ nodes with upper confidence bounds higher or equal to the lower confidence bound of t_i, and t_i will be moved to V_k at the end of this iteration. On the other hand, if a node t_i with $i \le \rho k$ was already removed from C in previous iterations, it must be moved to V_k, since $V \setminus (C \cup V_k)$ only contains non-top-k nodes. Therefore, all top-ρk nodes will be in V_k at the end of this iteration, and the algorithm stops.

Next, we prove that if at the end of an iteration, we have $\beta(s,t) \le \frac{1}{4}gap_\rho$ for any $t \in C$, then for any $t_j \in C, k+1 \le j \le n$ and any $t_i \in C, i \le \rho k$, we have $\hat{\pi}(s,t_i) - \beta(s,t_i) \ge \hat{\pi}(s,t_j) + \beta(s,t_j)$. We observe that

$$\beta(s,t_i) \le \frac{1}{4}gap_\rho = \frac{1}{4}\left(\pi(s,t_{\rho k}) - \pi(s,t_{k+1})\right) \le \frac{1}{4}\left(\pi(s,t_i) - \pi(s,t_j)\right),$$

and similarly $\beta(s,t_j) \le \frac{1}{4}\left(\pi(s,t_i) - \pi(s,t_j)\right)$. Therefore, $\beta(s,t_i) + \beta(s,t_j) \le \frac{1}{2}\left(\pi(s,t_i) - \pi(s,t_j)\right)$, and thus

$$\beta(s,t_i) + \beta(s,t_j) \le \left(\pi(s,t_i) - \pi(s,t_j)\right) - \left(\beta(s,t_i) + \beta(s,t_j)\right)$$
$$= (\pi(s,t_i) - \beta(s,t_i)) - \left(\pi(s,t_j) + \beta(s,t_j)\right) \le \hat{\pi}(s,t_i) - \hat{\pi}(s,t_j).$$

The last inequality uses the fact that $\pi(s,t_i) - \beta(s,t_i) \le \hat{\pi}(s,t_i)$ and $\pi(s,t_j) + \beta(s,t_j) \ge \hat{\pi}(s,t_j)$. Therefore we have $\hat{\pi}(s,t_i) - \beta(s,t_i) \ge \hat{\pi}(s,t_j) + \beta(s,t_j)$ for any $t_i, t_j \in C$ and $i \le \rho k, j \ge k+1$, and the Lemma follows. □

C.7 Proof of Lemma 4.8

We need the following lemma that bounds the empirical variance of the estimators in the candidate set C.

Lemma C.2. *Consider an iteration of TopPPR, and let r^b_{max}, r^f_{sum} and n_r denote the backward residue threshold, forward residue summation and the number of random walks in this iteration. For any $t \in C$, we have $\bar{\sigma}^2(s,t) \le 4m\cdot r^f_{sum}\cdot(r^b_{max})^2 + \frac{3\left(r^f_{sum}\right)^2\log n}{n_r}$ with probability at least $1 - 1/n^3$.*

Proof. Following the same notation as in the proof of Lemma 4.1, we let X_i be the estimator provided by the i-th random walk. Recall that the X_i's are i.i.d. random variables computed as follow: during the i-th random walk, each node u is sampled from the Alias structure r^f with probability $r^f(s,u)/r^f_{sum}$ to be the start node, and if the random walk terminates at a node v, we set $X_i = r^f_{sum}r^b(v,t)$. By the definition of personalized pageranks, we have that the probability that this random walk terminates at v is $\pi(u,v)$.

To bound the empirical variance $\bar{\sigma}^2(s,t)$, we first note that $\bar{\sigma}^2(s,t) \le \frac{1}{n_r}\sum_{i=1}^{n_r}X_i^2$. Let $Y = \frac{1}{n_r}\sum_{i=1}^{n_r}X_i^2$, it is easy to see that X_i^2, $i = 1,\ldots,n_r$ are also i.i.d. random variables. We can bound the expectation of X_i^2 as follows

$$E[X_i^2] \le \sum_{u\in V}\sum_{v\in V}\frac{r^f(s,u)}{r^f_{sum}}\cdot\pi(u,v)\cdot(r^f_{sum}r^b(u,v))^2$$
$$= r^f_{sum}\sum_{u,v\in V}r^f(s,u)\pi(u,v)r^b(v,t)^2.$$

Recall that TopPPR set $r^b(v,t) \le r^b_{max}(v,t) = r^b_{max}\sqrt{\frac{d_{in}(v)}{\hat{\pi}_b(s,v)}}$, and by Lemma C.1 we have $\hat{\pi}_b(s,v) \ge \pi(s,t)/4$, it follows that

$$E[X_i^2] \le 2r^f_{sum}\sum_{u,v\in V}r^f(s,u)\pi(u,v)\left(r^b_{max}\cdot\sqrt{\frac{4d_{in}(v)}{\pi(s,v)}}\right)^2$$
$$\le 2r^f_{sum}\left(r^b_{max}\right)^2\sum_{u,v\in V}r^f(s,u)\pi(u,v)\cdot\frac{d_{in}(v)}{\pi(s,v)}$$
$$= 2r^f_{sum}\left(r^b_{max}\right)^2\sum_{v\in V}\left(\frac{d_{in}(v)}{\pi(s,v)}\sum_{u\in V}r^f(s,u)\pi(u,v)\right).$$

By forward search, $\pi(s,v) = \pi^f(s,v) + \sum_{u\in V} r^f(s,u)\pi(u,v)$, it follows that $\sum_{u\in V} r^f(s,u)\pi(u,v) \le \pi(s,v)$, and thus

$$\begin{aligned} E[X_i^2] &\le 2r^f_{sum}\left(r^b_{max}\right)^2 \sum_{v\in V}\left(\frac{d_{in}(v)}{\pi(s,v)}\cdot \pi(s,v)\right) \\ &= 2r^f_{sum}\left(r^b_{max}\right)^2 \sum_{v\in V} d_{in}(v) = 2r^f_{sum}\left(r^b_{max}\right)^2\cdot m. \end{aligned}$$

Therefore, we have $E[Y] \le 2mr^f_{sum}\left(r^b_{max}\right)^2$. Let $z = \frac{8\left(r^f_{sum}\right)^2\log n}{n_r}$, since each $X_i^2 \le \left(r^f_{sum}\right)^2$, by Chernoff inequality, the probability that $Y \le 2E[Y] + z \le 4mr^f_{sum}\left(r^b_{max}\right)^2 + \frac{8\left(r^f_{sum}\right)^2\log n}{n_r}$ is at most

$$\begin{aligned} &exp\left(-n_r\left(E[Y]+z\right)^2 \Big/ \left(\left(r^f_{sum}\right)^2\cdot\left(\frac{8}{3}E[Y]+\frac{2}{3}z\right)\right)\right) \\ &\le exp\left(-\frac{3}{8}n_r\cdot\left(E[Y]+z\right)\Big/\left(r^f_{sum}\right)^2\right) \le exp\left(-\frac{3n_r\log n}{n_r}\right) \le \frac{1}{n^3}, \end{aligned}$$

and the lemma follows. □

Proof of Lemma 4.8. By Lemma 4.7, we need to show that if at some iteration, the number of random walks $n_r = \frac{cmr^f_{sum}\left(r^b_{max}\right)^2}{gap_\rho^2}\log n$ for constant c, we have $\beta(s,t) \le gap_\rho/4$ for any $t\in C$. Recall that

$$\beta(s,t) = \sqrt{\frac{2\bar{\sigma}^2(s,t)\ln\left(3n^3\log^2 n_r\right)}{n_r}} + \frac{3r^f_{sum}\ln\left(3n^3\log^2 n_r\right)}{n_r}. \quad (3)$$

By the assumption that $r^b_{max} \ge \sqrt{\frac{gap_\rho}{m}}$, we have

$$n_r \ge \frac{cmr^f_{sum}\left(r^b_{max}\right)^2}{gap_\rho^2}\log n \ge \frac{cmr^f_{sum}\cdot\frac{gap_\rho}{m}}{gap_\rho^2}\log n \ge \frac{cr^f_{sum}\log n}{gap_\rho},$$

and thus we can bound the the second term of equation (3) as

$$\frac{6r^f_{sum}\ln\left(3n^3\log^2 n_r\right)}{n_r} \le gap_\rho\frac{6\log\left(3n^3\log^2 n_r\right)}{c\log n} \le gap_\rho/8. \quad (4)$$

For c sufficiently large. The last inequality is due to $n \ge \log\frac{m}{gap_\rho}$.

To bound the first term of equation (3), we observe ~~that by~~ Lemma 4.1, the empirical variance $\bar{\sigma}^2(s,t) \le 2mr^f_{sum}\left(r^b_{max}\right)^2 + \frac{8\left(r^f_{sum}\right)^2\log n}{n_r}$ with probability at least $1-1/n^3$. If $2mr^f_{sum}\left(r^b_{max}\right)^2 \le \frac{8\left(r^f_{sum}\right)^2\log n}{n_r}$, we have $\bar{\sigma}^2(s,t) \le \frac{16\left(r^f_{sum}\right)^2\log n}{n_r}$ and the first term of equation (3) can be bounded by

$$\sqrt{\frac{2\bar{\sigma}^2(s,t)\ln 3n^3\log^2 n_r}{n_r}} \le \sqrt{\frac{\frac{16\left(r^f_{sum}\right)^2\log n}{n_r}\ln 3n^3\log^2 n_r}{n_r}} \le \frac{1}{8}gap_\rho.$$

The last inequality is due to inequality (4). On the other hand, if $2mr^f_{sum}\left(r^b_{max}\right)^2 \ge \frac{8\left(r^f_{sum}\right)^2\log n}{n_r}$, we have $\bar{\sigma}^2(s,t) \le 4mr^f_{sum}\left(r^b_{max}\right)^2$, and thus

$$\begin{aligned} \sqrt{\frac{2\bar{\sigma}^2(s,t)\ln\left(3n^3\log^2 n_r\right)}{n_r}} &\le \sqrt{\frac{8mr^f_{sum}\left(r^b_{max}\right)^2\ln\left(3n^3\log^2 n_r\right)}{n_r}} \\ &= \sqrt{\frac{8mr^f_{sum}\left(r^b_{max}\right)^2\ln 3n^3\log^2 n_r}{\frac{cmr^f_{sum}\left(r^b_{max}\right)^2}{gap_\rho^2}\log n}} \le gap_\rho\sqrt{\frac{24\log n\log n_r}{c\log n}} \le \frac{gap_\rho}{8} \end{aligned}$$

for c sufficiently large. Thus we have $\beta(s,t) \le gap_\rho/4$ for c sufficiently large, and the Lemma follows. □

C.8 Proof of Theorem 4.9 and Theorem 4.10

Proof. Recall that at the i-th iteration, we have $r^f_{max} = \frac{1}{2^i m}$ and $r^b_{max} = 1/2^i\sqrt{m}$ and walk number $n_r = c2^i n\log n/|C|$. The total query cost is $cost(forward) + cost(backward) + cost(walk)$, which can be bounded by

$$O\left(\frac{1}{r^f_{max}}\right) + O\left(\frac{\sqrt{m}}{r^b_{max}}\right) + n_r|C| = O\left(2^i m + c2^i n\log n\right). \quad (5)$$

For worst-case graph, if we can prove that TopPPR stops before iteration when $2^i = 1/\sqrt{gap_\rho}$, then the total query cost is bounded by $O\left(\frac{m+n\log n}{\sqrt{gap_\rho}}\right)$ and Theorem 4.9 follows. More precisely, we have $r^f_{max} = \sqrt{gap_\rho}/m$, $r^b_{max} = \sqrt{gap_\rho/m}$. By Lemma 4.8 and $r^b_{max} = \sqrt{gap_\rho/m}$, we have $n_r \ge \frac{c2^i n\log n}{|C|} \ge \frac{cn\log n}{\sqrt{gap_\rho}\cdot|C|} \ge \frac{c\log n}{\sqrt{gap_\rho}}$. The last inequality uses the fact that $|C| \le n$. Therefore, we have $r^f_{max}\left(r^b_{max}\right)^2 = \frac{\left(\sqrt{gap_\rho}\right)^3}{m^2}$ and $n_r \ge \frac{c\log n}{\sqrt{gap_\rho}} \ge \frac{cm^2\frac{\left(\sqrt{gap_\rho}\right)^3}{m^2}}{gap_\rho^2}\log n \ge \frac{cmr^f_{sum}\left(r^b_{max}\right)^2}{gap_\rho^2}\log n$. This proves that the number of walks $n_r = \Omega\left(\frac{cmr^f_{sum}\left(r^b_{max}\right)^2}{gap_\rho^2}\log n\right)$, and by Lemma 4.7, the algorithm stops at this iteration.

Similarly, for power law graph, we can prove that TopPPR stops before iteration when $2^i = \frac{1}{\sqrt{gap_\rho}}\cdot\frac{k^{\frac{1}{4}}}{n^{\frac{1}{4}}}$, under Assumption 1. Thus Theorem 4.10 follows from equation 5. □

C.9 Proof of Lemma 5.1

Proof. We define two types of random walks. A non-stop random walk from s is a traversal of G that starts from s and, at each step, proceeds to a randomly selected out-neighbor of the current node. An $(1-\alpha)$-walk is a random walk in which at each step the random walk proceeds to the next node with probability $1-\alpha$ and terminates with probability α.

Let $p_i(s,t)$ denote the probability that a non-stop random walk from s visits t in the i step. We have $\pi(s,t) = \sum_{i=0}^{\infty}\alpha(1-\alpha)^i p_i(s,t)$. To see this, note that $\alpha(1-\alpha)^i p_i(s,t)$ is the probability that a $(1-\alpha)$-walk starts at s and terminates at t using i steps. Summing i over 0 to ∞ and the equation follows. For $(\sqrt{1-\alpha})$-walk, recall that $(\sqrt{1-\alpha})^i p_i(s,t)$ is the probability that the $\sqrt{1-\alpha}$-walk visits t at the i-th step, and $(\sqrt{1-\alpha})^i$ is the value added to $\hat{\pi}(s,t)$, we have

$$E[\hat{\pi}(s,t)] = \sum_{i=0}^{\infty}\alpha(\sqrt{1-\alpha})^i\cdot(\sqrt{1-\alpha})^i p_i(s,t) = \pi(s,t),$$

and the Lemma follows. □