Efficient Algorithms for Approximate Single-Source Personalized PageRank Queries

SIBO WANG, The Chinese University of Hong Kong
RENCHI YANG, Nanyang Technological University
RUNHUI WANG, Rutgers University
XIAOKUI XIAO, National University of Singapore
ZHEWEI WEI, Renmin University of China
WENQING LIN, Tencent
YIN YANG, Hamad Bin Khalifa University
NAN TANG, Qatar Computing Research Institute, HBKU

Given a graph G, a source node s, and a target node t, the *personalized PageRank* (*PPR*) of t with respect to s is the probability that a random walk starting from s terminates at t. An important variant of the PPR query is *single-source PPR* (*SSPPR*), which enumerates all nodes in G and returns the top-k nodes with the highest PPR values with respect to a given source s. PPR in general and SSPPR in particular have important applications in web search and social networks, e.g., in Twitter's Who-To-Follow recommendation service. However, PPR computation is known to be expensive on large graphs and resistant to indexing. Consequently, previous solutions either use heuristics, which do not guarantee result quality, or rely on the strong computing power of modern data centers, which is costly.

Motivated by this, we propose effective index-free and index-based algorithms for approximate PPR processing, with rigorous guarantees on result quality. We first present FORA, an approximate SSPPR solution that combines two existing methods—Forward Push (which is fast but does not guarantee quality) and Monte Carlo Random Walk (accurate but slow)—in a simple and yet non-trivial way, leading to both high accuracy and efficiency. Further, FORA includes a simple and effective indexing scheme, as well as a module for top-k selection with high pruning power. Extensive experiments demonstrate that the proposed solutions are orders of magnitude more efficient than their respective competitors. Notably, on a billion-edge Twitter dataset, FORA answers a top-500 approximate SSPPR query within 1s, using a single commodity server.

Sibo Wang is supported by CUHK Direct Grant No. 4055114, CUHK University Startup Grant No. 4930911 and No. 5501570, and a donation from Tencent. Xiaokui Xiao is supported by MOE, Singapore, under grant MOE2015-T2-2-069, and by NUS, Singapore, under an SUG. Zhewei Wei is supported in part by National Natural Science Foundation of China (No. 61972401, 61932001 and 61832017) and by the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China under Grant 18XNLG21. Yin Yang is supported by NPRP grant #NPRP10-0208-170408 from the Qatar National Research Fund.
Authors' addresses: S. Wang, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong; email: swang@se.cuhk.edu.hk; R. Yang, Nanyang Technological University, 50 Nanyang Ave, Singapore; email: yang0461@ntu.edu.sg; R. Wang, Rutgers University, New Brunswick, NJ, United States; email: runhui.wang@rutgers.edu; X. Xiao, National University of Singapore, 13 Computing Drive, Singapore; email: xkxiao@nus.edu.sg; Z. Wei (corresponding author), Renmin University of China, Beijing, China; email: zhewei@ruc.edu.cn; W. Lin, Tencent, Shenzhen, China; email: edwlin@tencent.com; Y. Yang, Hamad Bin Khalifa University, Doha, Qatar; email: yyang@hkbu.edu.qa; N. Tang, Qatar Computing Research Institute, HBKU, Doha, Qatar; email: ntang@hkbu.edu.qa.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2019 Association for Computing Machinery.
0362-5915/2019/10-ART18 $15.00
https://doi.org/10.1145/3360902

CCS Concepts: • **Mathematics of computing → Graph algorithms**;

Additional Key Words and Phrases: Personalized PageRank, forward push, random walk

ACM Reference format:
Sibo Wang, Renchi Yang, Runhui Wang, Xiaokui Xiao, Zhewei Wei, Wenqing Lin, Yin Yang, and Nan Tang. 2019. Efficient Algorithms for Approximate Single-Source Personalized PageRank Queries. *ACM Trans. Database Syst.* 44, 4, Article 18 (October 2019), 37 pages.
https://doi.org/10.1145/3360902

1 INTRODUCTION

Personalized PageRank (*PPR*) is a fundamental operation first proposed by Google [Page et al. 1999], a major search engine. Specifically, given a graph G and a pair of nodes s, t in G, the PPR value $\pi(s, t)$ is defined as the probability that a random walk starting from s (called the *source node*) terminates at t (the *target node*), which reflects the importance of t with respect to s. One particularly useful variant of PPR is the *single-source PPR* (*SSPPR*), which takes as input a source node s and a parameter k and returns the top-k nodes in G with the highest PPR values with respect to s. According to a recent paper [Gupta et al. 2013], Twitter, a leading microblogging service, applies SSPPR in their Who-To-Follow application, which recommends to a user s (who is a node in the social graph) a number of other users (with high PPR values with respect to s) that user s might want to follow. Clearly, such an application computes SSPPR for every user in the social graph on a regular basis. Hence, accelerating PPR computation may lead to improved user experience (e.g., faster response time), as well as reduced operating costs (e.g., lower power consumption in the data center).

Similar to PageRank [Page et al. 1999], PPR computation on a web-scale graph is immensely expensive, which involves extracting eigenvalues of a $n \times n$ matrix, where n is the number of nodes that can reach millions or even billions in a social graph. Meanwhile, unlike PageRank, PPR values cannot be easily materialized: Since each pair of source/target nodes lead to a different PPR value, storing all possible PPR values requires $O(n^2)$ space, which is infeasible for large graphs. For these reasons, much previous work focuses on *approximate PPR* computation (defined in Section 2.1), which provides a controllable tradeoff between the execution time and result accuracy. Meanwhile, compared to heuristic solutions, approximate PPR provides rigorous guarantees on result quality.

However, even under the approximate PPR definition, SSPPR computation remains a challenging problem, since it requires sifting through all nodes in the graph. To our knowledge, the majority of existing methods (e.g., References [Lofgren et al. 2014, 2016; Wang et al. 2016]) focus on approximate pair-wise (i.e., with given source and target nodes) PPR computations. A naive solution is to compute pair-wise PPR $\pi(s, v)$ for each possible target node v and subsequently apply top-k selection. Clearly, the running time of this approach grows linearly to the number of nodes in the graph, which is costly for large graphs.

Motivated by this, we propose *FORA* (short for **FO**ward Push and **RA**ndom Walks), an efficient algorithm for approximate SSPPR computation. The basic idea of FORA is to combine two existing solutions in a simple and yet non-trivial way, which are (i) Forward Push [Andersen et al. 2007], which can either compute the exact SSPPR results at a high cost or terminate early but with no guarantee at all on the result quality, and (ii) Monte Carlo [Fogaras et al. 2005], which samples and executes random walks and provides rigorous guarantees on the accuracy of SSPPR results, but is rather inefficient. In fact, this idea is so effective that even without any indexing, basic FORA already outperforms its main competitors BiPPR [Lofgren et al. 2016] and HubPPR [Wang et al. 2016]. Then, we describe a simple and effective indexing scheme for FORA, as well as a novel algorithm for top-k selection. Extensive experiments using several real graphs demonstrate that FORA is more than two orders of magnitude faster than BiPPR, and more than an order of

magnitude faster than HubPPR. In particular, on a billion-edge Twitter graph, FORA answers top-500 SSPPR query within 1s, using a single commodity server.

2 BACKGROUND

2.1 Problem Definition

Let $G = (V, E)$ be a directed graph. In case the input graph is undirected, we simply convert it to a directed one by treating each edge as two directed edges of opposing directions. Given a source node $s \in V$ and a decay factor α, a random walk (or more precisely, random walk with restart [Fujiwara et al. 2012]) 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 then the probability that a random walk from s terminates at v [Page et al. 1999].

A *single-source PPR* (*SSPPR*) query takes as input a graph G, a source node s, and a parameter k, and returns the top-k nodes with the highest PPR values with respect to s, together with their respective PPR values. This article focuses on approximate SSPPR processing, and we first define a simpler version of the approximate SSPPR without top-k selection (called approximate whole-graph SSPPR), as follows:

Definition 2.1 (Approximate Whole-graph SSPPR). Given a source node s, a threshold δ, an error bound ϵ, and a failure probability p_f, an approximate whole-graph SSPPR query returns an estimated PPR $\hat{\pi}(s, v)$ for each node $v \in V$, such that for any $\pi(s, v) > \delta$,

$$|\pi(s, v) - \hat{\pi}(s, v)| \leq \epsilon \cdot \pi(s, v) \tag{1}$$

holds with at least $1 - p_f$ probability.

The above definition is consistent with existing work, e.g., References [Lofgren et al. 2014, 2016; Wang et al. 2016]. Next, we define the approximate top-k SSPPR, as follows:

Definition 2.2 (Approximate Top-k SSPPR). Given a source node s, a threshold δ, an error bound ϵ, a failure probability p_f, and a positive integer k, an approximate top-k SSPPR query returns a sequence of k nodes, $v_1, v_2, \ldots, v_k$, such that with probability $1 - p_f$, for any $i \in [1, k]$ with $\pi(s, v_i^*) > \delta$,

$$\hat{\pi}(s, v_i) \geq (1 - \epsilon)\pi(s, v_i) \tag{2}$$

$$\pi(s, v_i) \geq (1 - \epsilon) \cdot \pi(s, v_i^*) \tag{3}$$

hold with at least $1 - p_f$ probability, where v_i^* is the node whose actual PPR with respect to s is the ith largest.

Note that Equation (2) ensures the accuracy of the estimated PPR values, while Equation (3) guarantees that the ith result returned has a PPR value close to the ith largest PPR score. This definition is consistent with previous work [Wang et al. 2016]. Following previous work [Lofgren et al. 2014, 2016; Wang et al. 2016], we assume that $\delta = O(1/n)$, where n is the number of nodes in G. The intuition is that, we provide approximation guarantees for nodes with above-average PPR values.

In addition, most applications of personalized PageRank concern web graphs and social networks, in which case the underlying input graphs are generally *scale-free*; that is, for any $k \geq 1$, the fraction $f(k)$ of nodes in G that have k edges satisfies

$$f(k) = c \cdot k^{-\gamma}, \tag{4}$$

where γ is a parameter with $2 \leq \gamma \leq 3$ and c is a constant smaller than 1. It can be verified that, in a scale-free graph with $2 \leq \gamma \leq 3$, the average node degree $m/n = O(\log n)$. We will analyze the

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
$N^{out}(v)$	The set of out-neighbors of node v
$N^{in}(v)$	The set of in-neighbors of node v
$\pi(s, t)$	The exact PPR value of t with respect to s
α	The probability that a random walk terminates at a step
δ, ϵ, p_f	Parameters of an approximate PPR query, as in Definitions 2.1 and 2.2
r_{max}	The residue threshold for local update
$r(s, v)$	The residue of v during a local update process from s
$\pi^{\circ}(s, v)$	The reserve of v during a local update process from s
r_{sum}	The sum of all nodes' residues during a local update process from s
$\pi(s, v_k^*)$	The kth largest PPR value with respect to s

asymptotic performance of our algorithm on both general graphs and scale-free graphs. Table 1 lists the frequently used notations throughout the article.

Remark. Our algorithms can also handle SSPPR queries where the source s is not fixed but sampled from a node distribution. Interested readers are referred to Section 6.3 for details.

2.2 Main Competitors

Monte-Carlo. A classic solution for approximate PPR processing is the *Monte-Carlo (MC)* approach [Fogaras et al. 2005]. Given a source node s, MC generates ω random walks from s, and it records, for each node v, the fraction of random walks $f(v)$ that terminate at v. It then uses $f(v)$ as an estimation of the PPR $\hat{\pi}(s, v)$ of v with respect to s. According to Fogaras et al. [2005], MC satisfies Definition 2.1 with a sufficiently large number of random walks: $\omega = \Omega(\frac{\log(1/p_f)}{\epsilon^2 \delta})$. According to Lofgren et al. [2014], Lofgren et al. [2016], and Wang et al. [2016], as well as our experiments in Section 8, MC is rather inefficient. Specifically, the time complexity of MC is $O(\frac{\log(1/p_f)}{\epsilon^2 \delta})$. As will be explained later in Section 3.2, when $\delta = O(1/n)$ and the graph is scale-free, in which case $m/n = O(\log n)$, this time complexity is a factor of $1/\epsilon$ larger than that of *FORA* even without indexing or top-k pruning.

BiPPR and HubPPR. *BiPPR* [Lofgren et al. 2016] and its successor *HubPPR* [Wang et al. 2016] are currently the state-of-the-art for answering *pairwise* PPR queries, in which both the source node s and the target node t are given, and the goal is to approximate the PPR value $\pi(s, t)$ of t with respect to s. The main idea of BiPPR is a bi-direction search on the input graph G. The forward direction simply samples and executes random walks, akin to MC described above. Unlike MC, however, BiPPR requires a much smaller number of random walks, thanks to additional information provided by the backward search.

The backward search in BiPPR (dubbed as *reverse push*) is originally proposed in Andersen et al. [2007], and is rather complicated. In a nutshell, the reverse push starts from the target node t and recursively propagates *residue* and *reserve* values along the reverse directions of edges in G. Initially, the residue is 1 for node t and 0 for all other nodes. The original reverse push [Andersen et al. 2007] requires complete propagation until the residues of all nodes become very small, which is rather inefficient as pointed out in Lofgren et al. [2016]. BiPPR performs the same backward propagations but terminates early when the residues of all nodes are below a pre-defined threshold.

ALGORITHM 1: Forward Push

Input: Graph G, source node s, probability α, residue threshold r_{max}
Output: $\pi^\circ(s,v)$, $r(s,v)$ for all $v \in V$

1 $r(s,s) \leftarrow 1; r(s,v) \leftarrow 0$ for all $v \neq s$;
2 $\pi^\circ(s,v) \leftarrow 0$ for all v;
3 **while** $\exists v \in V$ *such that* $r(s,v)/|N^{out}(v)| > r_{max}$ **do**
4 **for** *each* $u \in \mathcal{N}^{out}(v)$ **do**
5 $r(s,u) \leftarrow r(s,u) + (1-\alpha) \cdot \frac{r(s,v)}{|N^{out}(v)|}$
6 $\pi^\circ(s,v) \leftarrow \pi^\circ(s,v) + \alpha \cdot r(s,v)$;
7 $r(s,v) \leftarrow 0$;

Then, the method performs forward search, i.e., random walks, utilizing the residue and reserve information computed during backward search. The main tricky part in BiPPR is how to set this residue threshold to minimize computation costs, while satisfying Inequality 1. Intuitively, if the residue threshold is set too high, then the forward search requires numerous random walks to reach the approximation guarantee; conversely, if the residue threshold is too low, then the cost of backward search dominates. Lofgren et al. [2016] provide a careful analysis and report that a residue threshold of $O(\epsilon \cdot \sqrt{\frac{m \cdot \delta}{n \cdot \log(1/p_f)}})$ strikes a good balance between forward and backward searches and achieves a low overall cost for pair-wise PPR computation.

To extend BiPPR to SSPPR, one simple method is to enumerate all nodes in G and compute the PPR value for each of them with respect to the source node s. The problem, however, is that the residue threshold designed in Lofgren et al. [2016] is not optimized for SSPPR, leading to poor performance. To explain, observe that applying BiPPR for SSPPR involves one backward search at each node in G, but only one single forward search from s. Therefore, we improve the performance of BiPPR by tuning down overhead of each backward search at the cost of a less-efficient forward search. This optimization turns out to be non-trivial, and we present it in Section 6.1. Nevertheless, the properly optimized version of BiPPR still involves high costs, since it either *(i)* degrades to the Monte-Carlo approach if the residue threshold is large or *(ii)* incurs a large number of backward searches if the residue threshold is small.

HubPPR [Wang et al. 2016] is an index structure based on BiPPR that features an improved algorithm for top-k queries. Since HubPPR inherits the deficiencies of the BiPPR, it is not suitable for SSPPR, either. We will demonstrate this in our experiments in Section 8.

Forward Push. *Forward Push* [Andersen et al. 2006] is an earlier solution that is not as efficient as BiPPR and HubPPR. We describe it in detail here, since the proposed solution FORA uses its components. Specifically, Forward Push can compute the *exact* PPR values at a high cost. It can also be configured to terminate early but without any guarantee on result quality. Algorithm 1 shows the pseudo-code of Forward Push for whole-graph SSPPR processing. It takes as input G, a source node s, a probability value α, and a threshold r_{max}; its output consists of two values for each node v in G: a *reserve* $\pi^\circ(s,v)$ and a *residue* $r(s,v)$. The reserve $\pi^\circ(s,v)$ is an approximation of $\pi(s,v)$, while the residue $r(s,v)$ is a by-product of the algorithm. In the beginning of the algorithm, it sets $r(s,s) = 1$ and $\pi^\circ(s,s) = 0$, and sets $r(s,v) = \pi^\circ(s,v) = 0$ for any $v \neq s$ (Lines 1–2 in Algorithm 1). Subsequently, the residue of s is converted into other nodes' reserves and residues in an iterative process (Lines 3–7).

Specifically, in each iteration, the algorithm first identifies every node v with $\frac{r(s,v)}{|N^{out}(v)|} > r_{max}$, where N^{out} denotes the set of out-neighbors of v (Line 3). After that, it *propagates* part of v's

residue to each u of v's out-neighbors, increasing u's residue by $(1-\alpha)\cdot\frac{r(s,v)}{|N^{out}(v)|}$. Then, it increases v's reserve by $\alpha\cdot r(s,v)$ and resets v's residue to $r(s,v)=0$. This iterative process terminates when every node v has $\frac{r(s,v)}{|N^{out}(v)|}\leq r_{max}$ (Line 3).

Andersen et al. [2006] show that Algorithm 1 runs in $O(1/r_{max})$ time and that the reserve $\pi^{\circ}(s,v)$ can be regarded as an estimation of $\pi(s,v)$. This estimation, however, does not offer any worst-case assurance in terms of absolute or relative error. As a consequence, Algorithm 1 itself is insufficient for addressing the problem formulated in Definitions 2.1 and 2.2.

TopPPR. Most recently, *TopPPR* [Wei et al. 2018] is proposed to combine the Forward Push, Monte-Carlo, and the backward search to process the top-k queries. The main idea is to use the Filter-Refinement paradigm to accelerate the top-k query processing. They first use the Forward Push and Monte-Carlo approach to derive the upper and lower bound of the PPR $\pi(s,v)$ for each target node v. It further maintains a set C of candidates that are the potential top-k answers by examining the upper and lower bound of each node; for example, if a node v has an upper bound $\pi(s,v)$ that is larger than that of the kth largest lower bound. Then, we know this node will not be in the top-k answer. When the candidate set is sufficiently small, backward search is started from this node and refines the upper bound and lower bound of the candidate nodes adaptively. The algorithm explores the power-law property and achieves a time complexity of $O(\frac{k^{\frac{1}{4}}\cdot n^{\frac{3}{4}}\cdot\log n}{\sqrt{gap_{\rho})}})$, where gap_{ρ} is a value that quantifies the difference between the top-k and non-top-k PPR values, and ρ is a precision parameter to guarantee that at least ρ fraction of the returned nodes are among the true top-k answers. Notice that gap_{ρ} should be no larger than $\pi(s,v_k^*)$ where $\pi(s,v_k^*)$ is the kth largest PPR with respect to s. Therefore, their time complexity can be written as $O(\frac{k^{\frac{1}{4}}\cdot n^{\frac{3}{4}}\cdot\log n}{\sqrt{\pi(s,v_k^*))}})$. On general graphs, the time complexity will degrade to $O(\frac{m+n\cdot\log n}{\sqrt{\pi(s,v_k^*)}})$. As we will see in Section 5.2 and Section 8, our top-k algorithm achieves both better theoretical result and practical performance.

Comparison with the conference version [Wang et al. 2017]. We make the following new contributions over the conference version.

- For whole-graph SSPPR queries, we revised the time complexity analysis to derive a refined bound (Section 3.2). Then, in Section 4, we further present optimization techniques for whole-graph SSPPR queries. With the new optimization technique, our index-free method improves over the solution in Wang et al. [2017] by 2×. Our new index-based method improves over the index-based solution in Wang et al. [2017] by at least 2× and up to 3× with 2× space consumption, which demonstrates the effectiveness of the new algorithm and a good trade-off of the new algorithms between the space consumption and query efficiency.
- For top-k SSPPR queries, the solution proposed in Wang et al. [2017] has a worst time complexity similar to the whole-graph SSPPR queries. In this article, we derive a new top-k algorithm in Section 5.2, whose time complexity depends on the kth largest PPR value, denoted as $\pi(s,v_k^*)$. When $\pi(s,v_k^*)$ is a constant, we then improve over the whole-graph SSPPR query by $O(1/n)$; when $\pi(s,v_k^*)$ is $O(1/n)$, then the time complexity of the top-k algorithm is identical to that of the whole-graph SSPPR algorithm. Since the kth largest PPR is typically in between $O(1)$ and $O(1/n)$, the new proposed algorithm improves over the solution proposed in Wang et al. [2017], and is shown to outperform the solution in Wang et al. [2017] by around 5× in our experimental evaluation.
- In Section 6, we further extend our results to global PageRank. The results show that it outperforms the classic Monte-Carlo approach and the Power-Iteration method. In addition,

Table 2. Comparison of Approximate Whole-graph SSPPR and top-k Algorithms with $\mathbf{1-1/n}$ Success Probability (Ref. Table 1 for the Definition of ϵ, δ, and $\pi(s, v_k^*)$)

Algorithm	**Space Overhead**	**Query Time**	
		whole-graph	Top-k
MC [Fogaras et al. 2005]	0	$O\left(\frac{\log n}{\delta \cdot \epsilon^2}\right)$	$O\left(\frac{\log n}{\pi(s, v_k^*)\epsilon^2}\right)$ (Using our top-k algorithm)
BiPPR [Lofgren et al. 2016]	0	$O\left(\frac{1}{\epsilon}\sqrt{\frac{mn \cdot \log n}{\delta}}\right)$	
HubPPR [Wang et al. 2016]	$O(n+m)$	$O\left(\frac{1}{\epsilon}\sqrt{\frac{mn \cdot \log n}{\delta}}\right)$	
TopPPR [Wei et al. 2018]	0	N.A.	$O(\frac{m+n \cdot \log n}{\sqrt{\pi(s, v_k^*)}})$ (general graphs)
			$O(\frac{k^{\frac{1}{4}} \cdot n^{\frac{3}{4}} \cdot \log n}{\sqrt{\pi(s, v_k^*)}})$ (power-law graphs)
FORA	0	$O(\min\{\frac{\sqrt{m \cdot \log n}}{\epsilon \cdot \sqrt{\delta}}, \frac{\log n}{\epsilon^2 \cdot \delta}\})$	$O(\min\{\frac{\sqrt{m \log n}}{\epsilon \cdot \sqrt{\pi(s, v_k^*)}}, \frac{\log n}{\epsilon^2 \cdot \pi(s, v_k^*)}\})$
FORA+	$O(\min\{n + \frac{1}{\epsilon \cdot \sqrt{\delta}}\sqrt{m \log(1/p_f)}, m\})$		

our new top-k algorithm can be further used to return the top-k nodes with the highest global PageRank with a time complexity that linearly depends on the inverse of the kth largest global PageRank.

- In the experimental evaluation, we have added four game social networks from Tencent Games to examine the effectiveness of our algorithms in real applications and two large synthetic datasets to examine the scalability of our proposed algorithms. Extensive experiments demonstrate that our solution is also effective on real social networks and is scalable to huge graphs with up to 8.6B edges.

Table 2 lists the time complexity and space consumption of all approximate algorithms to provide ϵ relative error guarantee for PPR values no smaller than δ with at least $1-1/n$ probability. As we can see, the proposed FORA/FORA+ achieves the best time complexity for both the whole-graph SSPPR and top-k queries.

3 FORA

This section presents the proposed FORA algorithm. We first describe a simpler version of FORA for whole-graph SSPPR (Definition 2.1) without indexing in Sections 3.1 and Sections 3.2. Then, we present the indexing scheme of FORA in Section 3.3. We present optimization techniques for whole-graph SSPPR in Section 4 and top-k query in Section 5.

3.1 Main Idea

As reviewed in Section 2.2: (i) MC is inefficient due to a large number of random walks required to satisfy the approximation guarantee, (ii) BiPPR and HubPPR either degrade to MC or require fewer forward random walks but still incur high cost due to numerous backward search operations, and (iii) Forward Push with early termination provides no formal guarantee on result quality. The

proposed solution FORA can be understood as a combination of these methods. In particular, FORA first performs Forward Push with early termination and subsequently runs random walks. Similar to BiPPR and HubPPR, FORA utilizes information obtained through Forward Push to significantly cut down the number of required random walks while satisfying the same result quality guarantees. But unlike BiPPR and HubPPR, in FORA there is a single invocation of Forward Push starting from the source node s, while BiPPR and HubPPR invoke numerous backward search operations.

Novelty. The main difference between the proposed approach and BiPPR/HubPPR is that the former combines forward push and MC, whereas the latter uses MC with backward propagation. The proposed idea (i.e., using forward instead of backward push) is indeed natural, and highly effective as shown in experiments (Section 8). Intuitively, forward push limits the search to nodes in the vicinity of the source node, which is more efficient for SSPPR compared to backward propagation, since the latter involves numerous nodes far from the source, as explained in Sections 2.2 (which explains BiPPR).

Combining forward push with MC, however, is far from straightforward. The devil is in the details. Specifically, in backward propagation, the residue $r(v,t)$ for a given node v is bounded by r_{max}, which is a controllable parameter (t is the destination node). In Forward Push (Algorithm 1), the corresponding concept is $r(s,v)$ (s being the source node), which depends on both r_{max} and the out-degree of v, which can be as large as $O(n)$ in the worst case. The proposed solution addresses this challenge with a novel mechanism that utilizes r_{sum} (Algorithm 2), whose correctness is rigorously established in this section. The novelty of our method lies in the fact that we realize a natural, effective, and yet challenging combination of forward push and MC with a non-trivial algorithmic design, explained below.

Details. Specifically, the reason that Forward Push with early termination fails to obtain any result quality guarantee is that it uses $\pi^\circ(s,v)$ to approximate $\pi(s,v)$, and yet, the two values are not guaranteed to be close. To mitigate this deficiency, we aim to utilize the residue $r(s,v)$ to improve the accuracy of $\pi^\circ(s,v)$. Towards this end, we utilize the following result from Andersen et al. [2006]:

$$\pi(s,t) = \pi^\circ(s,t) + \sum_{v \in V} r(s,v) \cdot \pi(v,t), \tag{5}$$

for any s, t, v in G. Our idea is to derive a rough approximation of $\pi(v,t)$ for each node v (denoted as $\pi'(v,t)$), and then combine it with the reserve of each node to compute an estimation of $\pi(s,t)$:

$$\pi(s,t) = \pi^\circ(s,t) + \sum_{v \in V} r(s,v) \cdot \pi'(v,t).$$

In particular, we derive $\pi'(v,t)$ by performing a number of random walks from v and set $\pi'(v,t)$ to the fraction of walks that ends at t.

It remains to answer two key questions in FORA: (i) how many random walks do we need for each node v?; and (ii) how should we set the residue threshold r_{max} in Forward Push? It turns out that although the FORA algorithm itself is simple, deriving the proper values for its parameters is rather challenging, since they must optimize efficiency while satisfying the result quality guarantee. In the following, we first present the complete FORA and answer question (i); then we answer question (ii) in Section 3.2.

Algorithm 2 illustrates the pseudo-code of FORA. Given G, a source node s, a probability value α, and a residue threshold r_{max}, FORA first invokes Algorithm 2 on G to obtain a reserve $\pi^\circ(s,v_i)$ and a residue $r(s,v_i)$ for each node v_i (Line 1 in Algorithm 2). After that, it computes the total residue of all nodes r_{sum}, based on which it derives a value ω that will be used to decide the number of random walks required from each node v_i (Line 2). Then, it initializes the PPR estimation of each

ALGORITHM 2: *FORA* for Whole-Graph SSPPR

Input: Graph G, source node s, probability α, threshold r_{max}, relative error threshold ϵ
Output: Estimated PPR $\hat{\pi}(s, v)$ for all $v \in V$

1 Invoke Algorithm 1 with input parameters G, s, α, and r_{max};
2 let $r(s, v_i), \pi^\circ(s, v_i)$ be the returned residue and reserve of node v_i;
3 Let $r_{sum} = \sum_{v_i \in V} r(s, v_i)$ and $\omega = r_{sum} \cdot \frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2 \cdot \delta}$;
4 Let $\hat{\pi}(s, v_i) = \pi^\circ(s, v_i)$ for all $v_i \in V$;
5 **for** $v_i \in V$ *with* $r(s, v_i) > 0$ **do**
6 Let $\omega_i = \lceil r(s, v_i) \cdot \omega / r_{sum} \rceil$;
7 Let $a_i = \frac{r(s, v_i)}{r_{sum}} \cdot \frac{\omega}{\omega_i}$;
8 **for** $i = 1$ *to* ω_i **do**
9 Generate a random walk W from v_i;
10 Let t be the end point of W;
11 $\hat{\pi}(s, t) += \frac{a_i \cdot r_{sum}}{\omega}$;
12 return $\hat{\pi}(s, v_1), \ldots, \hat{\pi}(s, v_n)$;

v_i to be $\hat{\pi}(s, v_i) = \pi^\circ(s, v_i)$, and it proceeds to inspect the nodes whose residues are larger than zero (Lines 3–4).

For each v_i of those nodes, it performs ω_i random walks from v_i, where

$$\omega_i = \left\lceil \frac{r(s, v_i)}{r_{sum}} \cdot \omega \right\rceil.$$

If a random walk ends at a node t, then FORA increases $\hat{\pi}(s, v_i)$ by $\frac{a_i \cdot r_{sum}}{\omega}$, where

$$a_i = \frac{r(s, v_i)}{r_{sum}} \cdot \frac{\omega}{\omega_i}.$$

After all v_i are processed, the algorithm returns $\hat{\pi}(s, v_i)$ as the approximated PPR value for v_i (Line 11).

To explain why FORA can provide accurate results, let us consider the ω_i random walks that it generates from a node v_i. Let $X_j(t)$ be a Bernoulli variable that takes value 1 if the jth random walk terminates at t, and value 0 otherwise. By definition,

$$\mathbb{E}[X_j] = \pi(v_i, t).$$

Then, based on the definition of ω, ω_i, and a_i, we have

$$\mathbb{E}\left[\frac{r_{sum}}{\omega} \cdot \sum_{j=1}^{\omega_i} \left(a_i \cdot X_j\right)\right] = r(s, v_i) \cdot \pi(v_i, t). \tag{6}$$

Observe that $\frac{r_{sum}}{\omega} \cdot \sum_{j=1}^{\omega_i} (a_i \cdot X_j)$ is exactly the amount of increment that $\hat{\pi}(s, t)$ receives when FORA processes v_i (see Lines 7–10 in Algorithm 2). We denote this increment as ψ_i. It follows that

$$\mathbb{E}\left[\sum_{i=1}^{n} \psi_i\right] = \sum_{i=1}^{n} r(s, v_i) \cdot \pi(v_i, t). \tag{7}$$

Combining Equations (5) and (7), we can see that FORA returns, for each node v, an estimated PPR $\hat{\pi}(s, v)$ whose expectation equals $\pi(s, v)$. Next, we will show that $\hat{\pi}(s, v)$ is very close to $\pi(s, v)$ with a high probability. For this purpose, we utilize the following concentration bound:

Theorem 3.1 ([Chung and Lu 2006]). *Let $X_1, \ldots, X_\omega$ be independent random variables with*

$$\Pr[X_i = 1] = p_i \quad \textit{and} \quad \Pr[X_i = 0] = 1 - p_i.$$

Let $X = \frac{1}{\omega} \cdot \sum_{i=1}^{\omega} a_i X_i$ with $a_i > 0$, and $\nu = \frac{1}{\omega} \sum_{i=1}^{\omega} a_i^2 \cdot p_i$. Then,

$$\Pr[|X - \mathbb{E}[X]| \geq \phi] \leq 2 \cdot \exp\left(-\frac{\phi^2 \cdot \omega}{2\nu + 2a\phi/3}\right),$$

where $a = \max\{a_1, \ldots, a_\omega\}$.

To apply Theorem 3.1, let us consider the $\omega' = \sum_{i=1}^{n} \omega_i$ random walks generated by FORA. Let $b_j = a_i$ if the jth random walk starts from v_i. Then, we have $\max_j b_j = 1$, and $b_j^2 \leq b_j$ for any j. In addition, let $Y_j(t)$ be a random variable that equals 1 if the jth walk terminates at t, and 0 otherwise. Then, by Theorem 3.1 and Equations (5) and (7), we have the following lemma:

Lemma 3.2. *For any node t, given an arbitrary relative error threshold ϵ, an arbitrary absolute threshold λ, we have that:*

$$\Pr[|\pi(s,t) - \hat{\pi}(s,t)| \geq \epsilon \cdot \pi(s,t)] \leq 2 \cdot \exp\left(-\frac{\epsilon^2 \cdot \omega \cdot \pi(s,t)}{r_{sum} \cdot (2 + 2\epsilon/3)}\right). \tag{8}$$

$$\Pr[|\pi(s,t) - \hat{\pi}(s,t)| \geq \lambda] \leq 2 \cdot \exp\left(-\frac{\lambda^2 \cdot \omega}{r_{sum} \cdot (2\pi(s,t) + 2\lambda/3)}\right). \tag{9}$$

Proof. First, define $Y' = \frac{1}{\omega'} \sum_{j=1}^{\omega'} b_j Y_j(t)$, and $\nu = \frac{1}{\omega'} \sum_{j=1}^{\omega'} b_j^2 \mathbb{E}[Y_j(t)]$. Let $a = \max\{b_1, \ldots, b_{\omega'}\}$. By definition, $b_j^2 \leq 1$, and hence, $\nu \leq \mathbb{E}[Y']$ and $a \leq 1$. By Theorem 3.1, for any ϕ, we have that $\Pr[|Y' - \mathbb{E}[Y']| \geq \phi] \leq 2 \cdot \exp(-\frac{\phi^2 \cdot \omega'}{2\nu + 2a\phi/3})$. Apply $\nu \leq \mathbb{E}[Y']$, we have that:

$$\Pr[|Y' - \mathbb{E}[Y']| \geq \phi] \leq 2 \cdot \exp\left(-\frac{\phi^2 \cdot \omega'}{2\mathbb{E}[Y'] + 2a\phi/3}\right).$$

Observe that $\frac{\omega' \cdot r_{sum}}{\omega}(\mathbb{E}[Y'] - Y') = \pi(s,t) - \hat{\pi}(s,t)$, the above inequality can be rewritten as:

$$\Pr\left[|\pi(s,t) - \hat{\pi}(s,t)| \geq \frac{\omega' \cdot r_{sum}}{\omega}\phi\right] \leq 2 \cdot \exp\left(-\frac{\phi^2 \cdot \omega'}{2\mathbb{E}[Y'] + 2a\phi/3}\right).$$

Besides, by Equation (6), we have that: $\mathbb{E}[Y'] \leq \frac{\omega}{\omega' \cdot r_{sum}} \cdot \pi(s,t)$, it is satisfied that:

$$\Pr\left[|\pi(s,t) - \hat{\pi}(s,t)| \geq \frac{\omega' \cdot r_{sum}}{\omega}\phi\right] \leq 2 \cdot \exp\left(-\frac{\phi^2 \cdot \omega'}{2\frac{\omega \cdot \pi(s,t)}{\omega' \cdot r_{sum}} + 2a\phi/3}\right).$$

Let $\phi = \frac{\omega \cdot \epsilon \cdot \pi(s,t)}{\omega' \cdot r_{sum}}$, we have:

$$\Pr[|\pi(s,t) - \hat{\pi}(s,t)| \geq \epsilon \cdot \pi(s,t)] \leq 2 \cdot \exp\left(-\frac{\epsilon^2 \cdot \omega \cdot \pi(s,t)}{r_{sum} \cdot (2 + 2a \cdot \epsilon/3)}\right).$$

Since $a \leq 1$, we get that:

$$\Pr[|\pi(s,t) - \hat{\pi}(s,t)| \geq \epsilon \cdot \pi(s,t)] \leq 2 \cdot \exp\left(-\frac{\epsilon^2 \cdot \omega \cdot \pi(s,t)}{r_{sum} \cdot (2 + 2\epsilon/3)}\right).$$

By setting $\epsilon = \lambda/\pi(s,t)$, we further have that:

$$\Pr[|\pi(s,t) - \hat{\pi}(s,t)| \geq \lambda] \leq 2 \cdot \exp\left(-\frac{\lambda^2 \cdot \omega}{r_{sum} \cdot (2\pi(s,t) + 2\lambda/3)}\right).$$

This finishes the proof. □

Lemma 3.3. *For any node t with $\pi(s,t) > \delta$, Algorithm 2 returns an approximated PPR $\hat{\pi}(s,t)$ that satisfies Equation (1) with at least $1 - p_f$ probability.*

Proof. Since $\omega = r_{sum} \cdot \frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2\cdot\delta}$, according to Lemma 3.2, we have that:

$$\Pr[|\pi(s,t) - \hat{\pi}(s,t)| \geq \epsilon \cdot \pi(s,t)] \leq \exp\left(-\frac{\epsilon^2 \cdot \pi(s,t)}{r_{sum} \cdot (2 + 2\epsilon/3)} \cdot r_{sum} \cdot \frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2 \cdot \delta}\right)$$
$$\leq 2\exp\left(-\frac{\pi(s,t)}{\delta} \cdot \log(2/p_f)\right).$$

Since $\pi(s,t) > \delta$, we have that:

$$\Pr[|\pi(s,t) - \hat{\pi}(s,t)| \geq \epsilon \cdot \pi(s,t)] < 2\exp(\log(2/p_f)) = p_f.$$

Also notice that the target t is arbitrarily chosen, and we can derive this bound for all nodes $t \in V$. Hence, the returned answer for the single-source PPR query satisfies Definition 2.1, which finishes the proof. □

3.2 Choosing r_{max}

Recall from Sections 2.2 and 3.1 that parameter r_{max} determines how quickly we can terminate Forward Push. A high value for r_{max} leads to low cost for Forward Push (since it can terminate early) but high cost for random walks (since a large number of them are required) and vice versa. Thus, finding the appropriate value of r_{max} requires modelling the overall running time of *FORA*. Recall that the Forward Push runs in $O(\frac{1}{r_{max}})$ time. In addition, the expected time complexity of the random walk phase is $O(r_{sum} \cdot \frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2})$, since each random walk takes $O(1)$ expected time to generate. Observe that

$$r_{sum} = \sum_{v_i \in V} r(s, v_i) \leq \sum_{v_i \in V} r_{max} \cdot |N^{out}(v_i)| = m \cdot r_{max}.$$

Therefore, the expected running time of Algorithm 2 is

$$O\left(\frac{1}{r_{max}} + m \cdot r_{max} \cdot \frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2\cdot\delta}\right).$$

Using the method of Lagrange multipliers, we can see that the above time complexity is minimized when

$$r_{max} = \frac{\epsilon}{\sqrt{m}} \cdot \sqrt{\frac{\delta}{(2\epsilon/3+2)\cdot\log(2/p_f)}}. \tag{10}$$

Accordingly, the expected time complexity of Algorithm 2 becomes

$$O\left(\frac{1}{\epsilon\cdot\sqrt{\delta}}\sqrt{m\cdot(2\epsilon/3+2)\cdot\log(2/p_f)}\right).$$

However, we also note that r_{sum} can be bounded by 1. Therefore, we have to consider two cases:

- **Case 1: $\mathbf{m \cdot r_{max} \leq 1}$.** Then, it is easy to verify that $\frac{1}{\epsilon\cdot\sqrt{\delta}}\sqrt{m\cdot(2\epsilon/3+2)\cdot\log(2/p_f)} \leq \frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2\cdot\delta}$. Then the time complexity can be bounded by $O(\frac{1}{\epsilon\cdot\sqrt{\delta}}\sqrt{m\cdot(2\epsilon/3+2)\cdot\log(2/p_f)})$.
- **Case 2: $\mathbf{m \cdot r_{max} > 1}$.** In this case, $\frac{1}{\epsilon\cdot\sqrt{\delta}}\sqrt{m\cdot(2\epsilon/3+2)\cdot\log(2/p_f)} > \frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2\cdot\delta}$. Therefore, if we set r_{max} according to Equation (10), we will have sub-optimal performance.

To remedy this issue, we set r_{max} to $\frac{\epsilon^2 \cdot \delta}{(2\epsilon/3+2)\cdot \log(2/p_f)}$. Then the time complexity can be bounded by $O(\frac{(2\epsilon/3+2)\cdot \log(2/p_f)}{\epsilon^2 \cdot \delta})$.

Therefore, combining the above two cases, the time complexity of FORA can be bounded by

$$O\left(\min\left\{\frac{1}{\epsilon\cdot\sqrt{\delta}}\sqrt{m\cdot\log(2/p_f)}, \frac{\log(2/p_f)}{\epsilon^2\cdot\delta}\right\}\right).$$

When $\delta = O(1/n)$, $p_f = O(1/n)$, the above time complexity becomes

$$O\left(\frac{1}{\epsilon}\min\left\{\sqrt{m\cdot n\cdot\log n}, \frac{n\cdot\log n}{\epsilon^2}\right\}\right)$$

for general graphs. When the graph is scale-free, in which case $m/n = O(\log n)$, the time complexity becomes $O(\frac{1}{\epsilon}n\cdot\log n)$, improving over the MC approach by $1/\epsilon$.

3.3 Indexing Scheme

Based on FORA, we propose a simple and effective index structure to further improve the efficiency of whole-graph SSPPR queries. The basic idea is to pre-compute a number of random walks from each node v and then store the destination of each walk. During query processing, if FORA requires performing x random walks from v, we would inspect the set S of random walk destinations pre-computed for v and then retrieve the first x nodes in S. As such, we avoid generating any random walks on-the-fly, which considerably reduces query overheads.

A natural question to ask is: How many random walks should we pre-compute for each node v? To answer this question, we first recall that, when the local update phase of FORA terminates, the residue of each node v is at most $|N^{out}(v)|\cdot r_{max}$. Combining this with Lemma 3.3, we can see that the number of random walks from v required by FORA is

$$\omega_{max}(v) = \left\lceil |N^{out}(v)|\cdot r_{max}\cdot\frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2\cdot\delta}\right\rceil. \tag{11}$$

If we set r_{max} according to Equation (10) in which case $m\cdot r_{max}\le 1$, then we have

$$\omega_{max}(v) = \left\lceil |N^{out}(v)|\cdot\frac{1}{\epsilon\cdot\sqrt{m\cdot\delta}}\cdot\sqrt{(2\epsilon/3+2)\cdot\log(2/p_f)}\right\rceil.$$

Otherwise, r_{max} is set to $\frac{\epsilon^2\cdot\delta}{(2\epsilon/3+2)\cdot\log(2/p_f)}$, we have

$$\omega_{max}(v) = |N^{out}(v)|.$$

In summary, we pre-compute $\omega_{max}(v)$ random walks from each node v and record the last nodes of those walks in our index structure. The total space overhead incurred is then bounded by

$$\begin{aligned}\sum_v \omega_{max}(v) &\le \min\left\{\sum_v\left\lceil |N^{out}(v)|\cdot\frac{\sqrt{(2\epsilon/3+2)\cdot\log(2/p_f)}}{\epsilon\cdot\sqrt{m\cdot\delta}}\right\rceil, \textstyle\sum_v |N^{out}(v)|\right\}\\ &\le \min\left\{n+\frac{\sqrt{m}}{\epsilon\cdot\sqrt{\delta}}\cdot\sqrt{(2\epsilon/3+2)\cdot\log(2/p_f)}, m\right\}.\end{aligned}$$

Therefore, we have the following lemma:

Lemma 3.4. *The space consumption of our index structure is*

$$O\left(\min\left\{n+\frac{1}{\epsilon}\sqrt{\frac{m\log(1/p_f)}{\delta}}, m\right\}\right). \tag{12}$$

When $\delta = O(1/n)$, $p_f = O(1/n)$, *and* $m/n = O(\log n)$, *the above space complexity becomes* $O(\min\{\frac{1}{\epsilon} n \cdot \log n, m\})$.

Remark. One may wonder whether we can also pre-compute the Forward Push result for each node so we can answer each query by a simple combination of pre-processed Forward Push and random walks, which could lead to higher query efficiency. However, we note that storing the Forward Push results for all nodes incurs significant space overheads. In particular, it requires $O(\min\{n, 1/r_{max}\})$ space for each node, where r_{max} is set according to Equation (10). As such, the total space consumption for preprocessing Forward Push results is

$$O\left(\min\left\{n^2, \frac{n}{\epsilon} \cdot \sqrt{\frac{m \cdot \log(1/p_f)}{\delta}}\right\}\right),$$

which is prohibitive for large graphs. Therefore, we do not store Forward Push results in our index structure.

4 OPTIMIZATIONS FOR WHOLE-GRAPH SSPPR QUERIES

In this section, we present optimization techniques to reduce the index size or improve the query efficiency of our FORA algorithm on whole-graph SSPPR queries. In particular, in Section 4.1, we will present our technique to reduce the index size by avoiding zero-hop nodes in the index structure. In Section 4.2, we will present our technique to reduce the query time by balancing the forward push and the random walk costs.

4.1 Pruning Zero-hop Random Walks

Recall that in FORA, after the local update phase, we sample random walks from each source with non-zero residues. Our main observation is that: α portion of the random walks is expected to stop at the current node, and with $O(1)$ time, we can immediately record the portion of such random walks and hence avoid simulating α portion of the total random walks. However, a question is that, can we still provide approximation guarantee while exploring this pruning strategy? We next demonstrate how to use the reduction of zero-hop random walk idea to ensure the approximation guarantee. We first define two random variables $\pi_0(s,t)$ and $\pi_1(s,t)$. We define $\pi_0(s,t)$ as the probability that a random walk from s immediately stopped at node t, i.e., the length of the random walk is 0; we further define $\pi_1(s,t)$ as the probability that a random walk from s that stopped at node t after traversing at least one node, i.e., the length of the random walk is at least 1. Then it is clear that the personalized PageRank $\pi(s,t)$ satisfies the following equation:

$$\pi(s,t) = \pi_0(s,t) + \pi_1(s,t).$$

We hence rewrite Equation (5) as follows:

$$\pi(s,t) = \pi^\circ(s,t) + \sum_{v \in V} r(s,v) \cdot (\pi_0(v,t) + \pi_1(v,t)).$$

Also notice that $\pi_0(s,t)$ either equals α or 0, depending on whether $s = t$ or not. Therefore, the above equation can be further rewritten as:

$$\pi(s,t) = \pi^\circ(s,t) + r(s,t) \cdot \alpha + \sum_{v \in V} r(s,v) \cdot \pi_1(v,t).$$

We define a random variable X_v as follows: We randomly select one of a out-neighbor u of v and then start a random walk from u. If the random walk stops at t, then $X_v = 1$; otherwise, $X_v = 0$. Then, it is not difficult to verify that $(1-\alpha) \cdot \mathbb{E}[X] = \pi_1(s,t)$.

$$\pi(s,t) = \pi^\circ(s,t) + r(s,t) \cdot \alpha + \sum_{v \in V} (1-\alpha) \cdot r(s,v) \cdot \mathbb{E}[X_v].$$

Define $r'(s,v) = (1-\alpha) \cdot r(s,v)$, we have that:

$$\pi(s,t) = \pi^{\circ}(s,t) + r(s,t) \cdot \alpha + \sum_{v \in V} r'(s,v) \cdot \mathbb{E}[X_v].$$

With the above equation, we can then further apply the same technique proposed in Section 3.1. We define $r'_{sum} = \sum_{v \in V} r'(s,v)$. Then, by sampling $\omega' = r'_{sum} \cdot \frac{(2\epsilon/3+2) \cdot \log(2/p_f)}{\epsilon^2 \cdot \delta}$ random walks, we can provide approximation guarantee for the whole-graph SSPPR queries. With this approach, when the same r_{max} is used as Algorithm 5, the maximum number of random walks sampled from a node v can be bounded by

$$(1-\alpha) \cdot r_{max} \cdot N^{out}(v) \cdot \frac{(2\epsilon/3+2) \cdot \log(2/p_f)}{\epsilon^2 \cdot \delta},$$

while previously, it requires

$$r_{max} \cdot N^{out}(v) \cdot \frac{(2\epsilon/3+2) \cdot \log(2/p_f)}{\epsilon^2 \cdot \delta}.$$

Therefore, with this optimization, we can also reduce the index size by α portion. Next, we further demonstrate our second optimization technique to improve the query efficiency.

4.2 Balancing Forward Push and Random Walk Cost

Recall that in FORA, we set r_{max} according to Equation (10) to minimize the time complexity of FORA. However, in practice, the derived r_{max} may not be the best choice, since Equation (10) considers the worst case while in practice the running time might be quite different. Of course, we may tune r_{max} for a different dataset and choose r_{max} that derives the best piratical performance. However, the tuned r_{max} will typically be data-dependent. Here, we are aiming to propose a solution that balances the forward push and random walk costs without any dependency on the datasets.

In Lofgren [2015], they propose a balanced approach for *BiPPR* by doing the backward propagation and maintaining the largest residue using a max-heap. Since the total number of random walks depends linearly on the maximum residue, it estimates the running time of the random walk and stops the backward propagation as soon as the running time is around the same as the forward random walk. Our proposed balancing strategy shares the similar spirit as theirs. However, we do not maintain the priority queue, since the number of random walks of FORA depends linearly on the total sum of the residues instead of the maximum residue. When we finish the forward push, we can accurately estimate the running time of the random walk part, since (i) we know the total number of random walks; (ii) the average running time of one random walk depends only on α, which is dataset-independent. Therefore, we can easily estimate the cost of a random walk and use it for random walk cost estimation no matter what dataset we are running on. Therefore, we propose the adaptive approach to balance the forward push and random walk cost as shown in Algorithm 3.

Initially, we start the forward push by setting $r_{max} = 1$ and calculate the current accumulated forward push cost (Algorithm 3, Line 11). Then, it checks if the forward push cost is still lower than the estimated random walk cost (Algorithm 3, Line 4). If this is the case, the algorithm continues the forward push process, then updates the accumulated forward push cost and updates the estimated random walk cost (Algorithm 3, Lines 5–11). The forward push terminates as soon as the estimated random walk cost is larger than the forward push cost (Algorithm 3, Line 4). By this strategy, it guarantees that, when the forward push terminates, the cost will not differ from the random walk cost by a large margin. The random walk phase is similar to the one in Algorithm 2 except that here we prune the zero-hop random walks as mentioned in Section 4.1. In particular, we convert

ALGORITHM 3: *FORA* for Whole-graph SSPPR with optimization

Input: Graph G, source node s, probability α, relative error threshold ϵ
Output: Estimated PPR $\hat{\pi}(s, v)$for all $v \in V$

1 Let rc be the cost of a random walk, and FC be the running time of the forward push;
2 Let $r(s, v_i), \pi^{\circ}(s, v_i)$ be the residue and reserve of node v_i in forward push, and initially only $r(s, s) = 1$ while all other values are zero;
3 Let $FC = 0, r_{max} = 1, r_{sum} = 1, \omega = r_{sum} \cdot \frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2\cdot\delta}$;
4 **while** $\exists v \in V$ *such that* $r(s, v)/|N^{out}(v)| > r_{max}$ *and* $FC < \omega * rc$ **do**
5 **for** *each* $u \in \mathcal{N}^{out}(v)$ **do**
6 $r(s, u) \leftarrow r(s, u) + (1 - \alpha) \cdot \frac{r(s,v)}{|N^{out}(v)|}$
7 $\pi^{\circ}(s, v) \leftarrow \pi^{\circ}(s, v) + \alpha \cdot r(s, v)$;
8 $r_{sum} = r_{sum} - \alpha \cdot r(s, v)$;
9 $\omega = r_{sum} \cdot (1 - \alpha) \cdot \frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2\cdot\delta}$;
10 $r(s, v) \leftarrow 0$;
11 $FC \leftarrow$ current elapsed time;
12 Let $\hat{\pi}(s, v_i) = \pi^{\circ}(s, v_i) + \alpha \cdot r(s, v_i)$ for all $v_i \in V$;
13 **for** $v_i \in V$ *with* $r(s, v_i) > 0$ **do**
14 Let $r(s, v_i) = (1 - \alpha) \cdot r(s, v_i), \omega_i = \lceil r(s, v_i)(1 - \alpha) \cdot \omega/r_{sum} \rceil$;
15 Let $a_i = \frac{r(s,v_i)}{r_{sum}} \cdot \frac{\omega}{\omega_i}$;
16 **for** $i = 1$ *to* ω_i **do**
17 Randomly select a out-neighbor u of v_i and generate a random walk W from u;
18 Let t be the end point of W;
19 $\hat{\pi}(s, t) += \frac{a_i \cdot r_{sum}}{\omega}$;
20 return $\hat{\pi}(s, v_1), \ldots, \hat{\pi}(s, v_n)$;

α portion of the residue $r(s, v_i)$ to its reserve (Algorithm 3, Line 12), and the residue of node v_i is reduced to $(1 - \alpha) \cdot v_i$ (Algorithm 3, Line 14). When we sample a random walk from each node v_i, we first randomly select one of its out-neighbors and then do random walk from these nodes, thus avoiding the zero-hop random walks.

As we will see in our experimental evaluation, the balanced strategy can help reduce the average running time of whole-graph SSPPR queries by almost half, which demonstrates the effectiveness of the balancing strategy. For the indexing version of our FORA and the top-k algorithm, we tune r_{max} to evaluate the trade-off between the index size and the query performance in Section 8.

5 TOP-*K* SSPPR

In this section, we discuss how FORA handles approximate top-k SSPPR queries.

Rationale. A straightforward approach to answer a top-k SSPPR query with FORA is to first apply it to perform a whole-graph SSPPR query and then return the k nodes with the largest approximate PPR values. However, if we are to satisfy the accuracy requirement described in Definition 2.2, we would need to set the parameters of FORA according to the exact kth largest PPR value $\pi(s, v_k^*)$, which is unknown in advance. To address this, a naive solution is to conservatively set $\pi(s, v_k^*) = 1/n$, which, however, would lead to unnecessary overheads.

To avoid the aforementioned overheads, we propose a trial-and-error approach as follows: We first assume that $\pi(s, v_k^*)$ is a large value (e.g., $1/k$), and we set the parameters of FORA

ALGORITHM 4: Top-k FORA with bound refinement

Input: Graph G, source node s, probability α
Output: k nodes with the highest approximate PPR scores

1 **for** $\delta = \frac{1}{k}, \frac{1}{2k}, \frac{1}{4k}, \ldots, \frac{1}{n}$ **do**
2 Invoke Algorithm 2 with G, s, α, and r_{max} set by Equation (10) and fail probability $p'_f = \frac{p_f}{n \cdot \log n}$;
3 Let $C = \{v'_1, \ldots, v'_k\}$ be the set that contains the k nodes with the top-k largest lower bounds (from Theorem 3.1);
4 Let $LB(u)$ and $UB(u)$ be the lower and upper bounds of $\pi(s, u)$ (from Theorem 3.1);
5 **if** $UB(v'_i) < (1 + \epsilon) \cdot LB(v'_i)$ *for* $i \in [1, k]$ **and** $LB(v'_k) \geq \delta$ **then**
6 Let U be the set of nodes $u \in V \setminus C$ such that $UB(u) > (1 + \epsilon) \cdot LB(v'_k)$;
7 **if** $\nexists u \in U$ **such that** $UB(u) < (1 + \epsilon) \cdot LB(u)/(1 - \epsilon)$ **then**
8 return $v'_1, v'_2, \ldots, v'_k$ and their estimated PPR;

accordingly to perform a whole-graph SSPPR query. After that, we inspect the results obtained to check whether the estimated PPR values are indeed large. If they are not as large as we have assumed, then we re-run FORA with more conservative parameters and check the new results returned. This process is conducted iteratively, until we are confident that the results from FORA conform to the requirements in Definition 2.2. In this section, we first present a top-k algorithm by iteratively refining the upper and lower bounds of the top-k PPR results in Section 5.1. Nevertheless, it is expensive to calculate the upper and lower bounds for each node in each iteration, and it is unclear whether the algorithm will terminate with δ close to $\pi(s, v^*_k)$ or not. Therefore, the expected running time of Algorithm 4 might be identical to that of invoking Algorithm 2 with $\delta = 1/n$. Therefore, in Section 5.2, we further propose a new top-k query algorithm that provides guarantee on the expected running time, and that the algorithm has high probability to terminate with $\pi(s, v^*_k)/4 \leq \delta \leq \pi(s, v^*_k)$.

5.1 Top-k with Bound Refinement

Algorithm. Algorithm 4 shows the pseudo-code of the top-k extension of FORA with bound refinement. The algorithm consists of at most $\log n$ iterations. In the ith iteration, we invoke Algorithm 2 with δ set to $\frac{1}{2^{i-1} \cdot k}$ and the failure probability set to $p'_f = \frac{p_f}{n \log n}$ (Lines 1–2 in Algorithm 4). (The reason for this setting will be explained shortly.) After we obtain the results from *FORA*, we compute an upper bound and a lower bound of each node's PPR value and use them to decide whether the current top-k results are sufficiently accurate (Lines 3–8). If the top-k results are accurate, then we return them as the top-k answers (Line 8); otherwise, we proceed to the next iteration. In the following, we elaborate how the upper and lower bounds of each node's PPR value is derived.

Define $LB_0(v) = 0$ and $UB_0(v) = 1$ for any $v \in V$. We have the following theorem that establishes the lower bound $LB_j(v)$ and upper bound $UB_j(v)$ of $\pi(s, v)$ in the jth iteration of Algorithm 4:

Theorem 5.1. *In the jth iteration of Algorithm 4, let ω_j be the ω calculated by* FORA *(Algorithm 2, Line 2) in this iteration, and $\pi^\circ_j(s, v)$ and $\hat{\pi}_j(s, v)$ be the reserve and estimated PPR of v. Define*

$$\epsilon_j = \sqrt{\frac{3r_{sum} \cdot \log(2/p'_f)}{\omega_j \cdot \max\{\pi^\circ_j(s, v), LB_{j-1}(s, v)\}}}, \text{ and } \lambda_j = \frac{2/3 \log(2/pf')}{2\omega_j}$$

$$+\frac{\sqrt{\frac{4}{9}r_{sum}^2 \cdot \log^2(2/p'_f) + 8r_{sum} \cdot \omega_j \cdot \log(2/p'_f) \cdot UB_{j-1}(v)}}{2\omega_j}.$$

Then, with at least $1 - p'_f$ *probability, the following two inequalities hold simultaneously:*

$$\hat{\pi}_j(s,v)/(1+\epsilon_j) \le \pi(s,v) \le \hat{\pi}_j(s,v)/(1-\epsilon_j),$$
$$\hat{\pi}_j(s,v) - \lambda_i \le \pi(s,v) \le \hat{\pi}_j(s,v) + \lambda_i.$$

Proof. Ref. Appendix A. □

Theorem 5.1 enables us to derive tight lower and upper bounds of each node's PPR value in each iteration. In particular, we set

$$UB_j(v) = min\{1, \hat{\pi}_j(s,v)/(1-\epsilon), \hat{\pi}_j(s,v) + \lambda_i\},$$

$$LB_j(v) = \max\{\hat{\pi}_j(s,v)/(1+\epsilon), \hat{\pi}_j(s,v) - \lambda_i, 0\}.$$

With these upper and lower bounds, the following theorem shows that if Lines 5 and 7 in Algorithm 4 hold, then Algorithm 4 returns the answer for the approximate top-k SSPPR query.

Theorem 5.2 (Approximate Top-k). *Let* $v'_1, \ldots, v'_k$ *be the* k *nodes with the largest lower bounds in the* j*th iteration of Algorithm 4. Let* U *be the set of nodes* $u \in V \setminus C$ *such that* $UB_j(u) > (1+\epsilon) \cdot LB_j(v'_k)$*, If* $UB(v'_i) < (1+\epsilon) \cdot LB(v'_i)$ *for* $i \in [1,k]$*,* $LB_j(v'_k) \ge \delta$*, and there exists no* $u \in U$ *such that* $UB_j(u) < (1+\epsilon) \cdot LB_j(u)/(1-\epsilon)$*, then returning* $v'_1, \ldots, v'_k$ *and their estimated PPR values would satisfy the requirements in Definition 2.2 with at least* $1 - j \cdot n \cdot p'_f$ *probability.*

Proof. Ref. Appendix A. □

Now recall that the number of iterations in Algorithm 4 is $\log n$, and in each iteration, we assume that the upper and lower bounds are correct. Hence, by applying union bound, the failure probability will be at most $n \log n \cdot p'_f$. Note that $p'_f = \frac{p_f}{n \log n}$. The failure probability is hence no more than p_f, and we guarantee that the returned answer has approximation with at least $1 - p_f$ probability.

5.2 Top-k with Improved Time Complexity

Despite the fact that Algorithm 4 provides superb performance on top-k query processing as shown in Wang et al. [2017], there is no guarantee that the running time will depend on $\delta = \pi(s, v_k^*)$ instead of $\delta = 1/n$. Also, after each iteration, we need to re-calculate the lower- and upper-bounds for each node, which may take more than half of the query running time. This motivates us to propose our new top-k algorithm, which avoids the overheads of the bound-refinement and provides running time guarantees with respect to $\pi(s, v_k^*)$.

Algorithm. Algorithm 5 shows the pseudo-code of the top-k extension of FORA. The algorithm consists of at most $\log_2(n/k)$ iterations. In the ith iteration, we invoke Algorithm 2 with δ set to $\frac{1}{k \cdot 2^{i-1}}$, relative error threshold $\epsilon' = \epsilon/2$, and the failure probability set to $p'_f = \frac{p_f}{n \log_2(n/k)}$ (Lines 1–2, Algorithm 5). (The reason for this setting will be explained shortly.) After we obtain the results from *FORA*, we compute the estimated PPR scores for each node and get the kth largest estimated PPR value. We compare the kth estimated PPR score with $(1+\epsilon) \cdot \delta$ and use this as evidence to see whether the current top-k results are sufficiently accurate (Line 4). If the top-k results are accurate, then we return them as the top-k answers (Lines 5–6); otherwise, we halve the value of δ and proceed to the next iteration. In the following, we analyze the approximation guarantee and

time complexity of Algorithm 5. First, we have the following lemma about the value of δ when Algorithm 5 terminates:

Lemma 5.3. *Let v_k be the node that has the kth largest estimated PPR value and $\hat{\pi}(s, v_k)$ be the estimated PPR value for node v_k with respect to s. Let v_k^* be the node with the true kth largest PPR. Then, when Algorithm 5 terminates, it holds for δ that:*

- $\delta > \pi(s, v_k^*)$ *with at most* $n \cdot p_f'/8$ *probability;*
- $\delta \leq \pi(s, v_k^*)$ *with at least* $1 - \log_2 \frac{1}{\pi(s, v_k^*)} \cdot n \cdot p_f'/8$*;*
- $\delta \leq \pi(s, v_k^*)/2^{x+1}$ $(x = 1, 2, 3 \ldots)$ *with at most* $\frac{p_f'}{2^x}$ *probability.*

Proof. We first consider the case when Algorithm 5 terminates with $\delta > \pi(s, v_k^*)$.

$\boldsymbol{\delta > \pi(s, v_k^*)}$**.** If $\delta > \pi(s, v_k^*)$ when Algorithm 5 terminates, then we know that there exists at least $n - k + 1$ nodes such that $\pi(s, v) < \delta$. Denote X as the set of nodes such that $v \in X$ if $\pi(s, v) < \delta$. For any of these nodes, we consider the probability that their PPR values with respect to s is greater than $(1 + \epsilon) \cdot \delta$. Note that according to Line 2 of Algorithm 5, the number of random walks is set to $\frac{r_{sum} \cdot (\epsilon/2 \cdot 2/3 + 2) \cdot \log(2/p_f')}{(\epsilon/2)^2 \cdot \delta}$. Let t be a node in X, and $\lambda = \epsilon \cdot \delta$, according to Lemma 3.2, we have that:

$$
\begin{aligned}
\Pr[\hat{\pi}(s,t) - \pi(s,t) > \lambda] &\leq \exp\left(-\frac{\lambda^2}{r_{sum} \cdot (2\pi(s,t) + 2\lambda/3)} \cdot \frac{4r_{sum} \cdot (2 + \epsilon/3) \cdot \log(2/p_f')}{\epsilon^2 \cdot \delta}\right) \\
&\leq \exp\left(-\frac{4\delta \cdot (2 + \epsilon/3)}{2 \cdot \pi(s,t) + 2\epsilon \cdot \delta/3} \cdot \log(2/p_f')\right) \\
&\leq \exp\left(-\frac{4\delta \cdot (2 + \epsilon/3)}{2 \cdot \delta + 2\epsilon \cdot \delta/3} \cdot \log(2/p_f')\right) \qquad (\delta > \pi(s,t)) \\
&\leq \exp\left(-\frac{4 \cdot (2 + \epsilon/3)}{2 + 2\epsilon/3} \cdot \log(2/p_f')\right) \\
&< \exp\left(-3 \cdot \log(2/p_f')\right) < p_f'/8.
\end{aligned}
$$

So, by union bound, it is satisfied that $\hat{\pi}(s, t) < \pi(s, t) + \epsilon \cdot \delta$ holds for any node $t \in X$ with at least $1 - n \cdot p_f'/8$ probability. Also note that $\pi(s, t) < \delta$. This indicates that with at least probability $1 - n \cdot p_f'/8$, for all nodes $t \in X$, it is satisfied that $\hat{\pi}(s, t) < (1 + \epsilon) \cdot \delta$. Since there are at least $n - k + 1$ nodes in X, it indicates that the returned $\hat{\pi}(s, v_k)$ must be no larger than $max_{v \in X} \hat{\pi}(s, v)$, which is less than $(1 + \epsilon) \cdot \delta$. However, this contradicts the fact that Algorithm 5 terminates when $\hat{\pi}(s, v_k) \geq (1 + \epsilon) \cdot \delta$. As a result, with at most $n \cdot p_f'/8$ probability, the algorithm terminates when $\delta > \pi(s, v_k^*)$.

Next, we consider the probability when Algorithm 5 terminates with $\delta \leq \pi(s, v_k^*)$.

$\boldsymbol{\delta \leq \pi(s, v_k^*)}$**.** Since in Algorithm 5, there are at most $\log_2 \frac{1}{k \cdot \pi(s, v_k*)}$ iterations such that $\delta > \pi(s, v_k^*)$, the probability that the algorithm terminates when $\delta > \pi(s, v_k^*)$ is bounded by $\log_2 \frac{1}{k \cdot \pi(s, v_k^*)} n \cdot p_f'/8$. Hence, with at least probability $1 - \log_2 \frac{1}{k \cdot \pi(s, v_k^*)} n \cdot p_f'/8$, the algorithm terminates with $\delta \leq \pi(s, v_k^*)$.

Finally, we consider whether δ will be too small when Algorithm 5 terminates.

$\boldsymbol{\delta \leq \pi(s, v_k^*)/2^{x+1}}$**.** Consider the k nodes with the k largest PPR values. Denote these nodes as $v_1^*, v_2^*, v_3^*, \ldots, v_k^*$. Consider the estimated PPR of v_i^* with respect to s $(1 \leq i \leq k)$. According to Lemma 3.2, it is satisfied that:

ALGORITHM 5: Top-k FORA

Input: Graph G, source node s, probability α
Output: k nodes with the highest approximate PPR scores

```
for δ = 1/k, 1/(2k) ..., 1/n do
    Invoke Algorithm 2 with G, s, and α with ε' = ε/2, failure probability p'_f = p_f / (n log_2 (n/k)), and r_max
    set by Equation (10);
    Let π̂(s, v_k) be the kth largest estimated PPR score returned by Algorithm 2;
    if π̂(s, v_k) ≥ (1 + ε) · δ then
        Let v'_1, v'_2, ..., v'_k be the k nodes with the top-k largest PPR values;
        return v'_1, v'_2, ..., v'_k and their estimated PPR values;
```

$$
\begin{aligned}
\Pr[\hat{\pi}(s,v_i^*) \le (1+\epsilon)\cdot\delta] &\le \Pr[\hat{\pi}(s,v_i^*) \le (1+\epsilon)\cdot\pi(s,v_k^*)/2^{x+1}] \quad (\delta \le \pi(s,v_k^*)/2^{x+1}) \\
&\le \Pr[\hat{\pi}(s,v_i^*) \le (1+\epsilon)\cdot\pi(s,v_i^*)/2^{x+1}] \quad (\pi(s,v_k^*) \le \pi(s,v_i^*)) \\
&\le \Pr[\hat{\pi}(s,v_i^*) \le (1-(1-1/2^x)\cdot\epsilon)\cdot\pi(s,v_i^*)] \\
&\quad (1+\epsilon)/2^{x+1} \le 1-(1-1/2^x)\cdot\epsilon \text{ for } 0<\epsilon<1 \\
&\le \exp\left(-\frac{((1-1/2^x)\cdot\epsilon)^2\cdot\pi(s,v_i^*)}{r_{sum}\cdot(2+2(1-1/2^x)\cdot\epsilon/3)}\cdot\frac{4r_{sum}\cdot(2+\epsilon/3)\cdot\log(2/p_f')}{\epsilon^2\cdot\delta}\right) \\
&\le \exp\left(-2(1-1/2^x)^2\cdot 2^{x+1}\cdot\frac{2(2+\epsilon/3)}{2+2(1-1/2^x)\cdot\epsilon}\cdot\log(2/p_f')\right) \\
&\quad 2(1-1/2^x)^2 > 1, \quad \frac{2(2+\epsilon/3)}{2+(1-1/2^x)} > 1 \\
&\le \exp\left(-2^{x+1}\cdot\log(2/p_f')\right) < \frac{(p_f')^2}{2^{x+1}}.
\end{aligned}
$$

By union bound, the probability that $\hat{\pi}(s,v_i^*) \ge (1+\epsilon)\cdot\delta$ holds for any $1 \le i \le k$ simultaneously is at least $1-k\cdot\frac{(p_f')^2}{2^{x+1}} \ge 1-\frac{p_f'}{2^{x+1}}$. Since there are k estimations no smaller than $(1+\epsilon)\cdot\delta$, Algorithm 5 will terminate. Therefore, Algorithm 5 terminates when $\delta \le \pi(s,v_i^*)/2^{i+1}$ with at most $\frac{p_f'}{2^{x+1}}$ probability, which finishes the proof. □

Lemma 5.3 indicates several desired properties of our top-k algorithm. First, the algorithm stops with $\delta > \pi(s,v_k^*)$ with low probability. Besides, it terminates with $\delta \le \pi(s,v_k^*)$ with high probability, which is an important condition for providing approximate top-k answer. Third, when the algorithm terminates, δ will not deviate from $\pi(s,v_k^*)$ by a large margin. The larger the margin is between δ and $\pi(s,v_k^*)$, the lower the probability it is. By leveraging 5.3, we further have the following lemma on the time complexity of our top-k algorithm:

Lemma 5.4. *The expected running time of Algorithm 5 can be bounded by*

$$
O\left(\min\left\{\frac{\sqrt{m\cdot\log(2/p_f)}}{\epsilon\cdot\sqrt{\pi(s,v_k^*)}}, \frac{\log(2/p_f)}{\epsilon^2\cdot\pi(s,v_k^*)}\right\}\right).
$$

Proof. Let c be the constant factor of the time complexity of Algorithm 2 and δ^* be the value of δ when Algorithm 5 terminates. Suppose that the algorithm terminates after $i+1$ iterations, and

note that the time complexity of each iteration is bounded by $O(\frac{c}{\epsilon\cdot\sqrt{\delta}}\sqrt{m\cdot(2\epsilon/3+2)\cdot\log(2/p_f)})$. Then, the cost C of FORA can be bounded by

$$\begin{aligned}
C &= \sum_{\delta=1/k}^{1/(2^i\cdot k)} \frac{c}{\epsilon\cdot\sqrt{\delta}}\sqrt{m\cdot(2\epsilon/3+2)\cdot\log(2/p_f)} \\
&= \frac{c}{\epsilon}\sqrt{m\cdot(2\epsilon/3+2)\cdot\log(2/p_f)}\cdot\sum_{\delta=1/k}^{1/(2^i\cdot k)}\frac{1}{\sqrt{\delta}} \\
&\le \frac{c}{\epsilon}\sqrt{m\cdot(2\epsilon/3+2)\cdot\log(2/p_f)}\cdot\frac{1+1/\sqrt{\delta^*}}{1-1/\sqrt{2}} \\
&\le \frac{c}{\epsilon}\sqrt{m\cdot(2\epsilon/3+2)\cdot\log(2/p_f)}\cdot\frac{4}{\sqrt{\delta^*}}.
\end{aligned}$$

Denote

$$\phi = \frac{4c}{\epsilon}\sqrt{m\cdot(2\epsilon/3+2)\cdot\log(2/p_f)}.$$

Next, we further consider the expected cost of Algorithm 5.

$$\begin{aligned}
\mathbb{E}[C] &= \Pr[\delta > \pi(s,v_k^*)]\cdot C_{\delta>\pi(s,v_k^*)} + \Pr[\pi(s,v_k^*)/4 < \delta \le \pi(s,v_k^*)]\cdot C_{\pi(s,v_k^*)/4<\delta\le\pi(s,v_k^*)} \\
&\quad + \sum_{x=1}^{\log_2(n/\pi(s,v_k^*))} \Pr[\pi(s,v_k^*)/2^{i+2} < \delta \le \pi(s,v_k^*)/2^{i+1}]\cdot C_{\pi(s,v_k^*)/2^{i+2}<\delta\le\pi(s,v_k^*)/2^{i+1}} \\
&< 1\cdot\frac{\phi}{\sqrt{\pi(s,v_k^*)}} + 1\cdot\frac{\phi}{\sqrt{\pi(s,v_k^*)/4}} + \sum_{i=1}^{\log_2(n/\pi(s,v_k^*))}\frac{p_f'}{2^{i+1}}\frac{\phi}{\sqrt{\pi(s,v_k^*)/2^{i+1}}} \\
&< \frac{4\phi}{\sqrt{\pi(s,v_k^*)}} = \frac{16}{\epsilon\cdot\sqrt{\pi(s,v_k^*)}}\sqrt{m\cdot(2\epsilon/3+2)\cdot\log(2/p_f)}.
\end{aligned}$$

Besides, note that the time complexity of each iteration can also be bounded by $O(\frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\delta})$. Still let c denote the constant in the time complexity. Then, the cost C of FORA can be bounded by

$$C = c\cdot\frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2}\cdot\sum_{\delta=1/k}^{1/(2^i\cdot k)}\frac{1}{\delta} \le c\cdot\frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2}\cdot\frac{2}{\delta}.$$

Let $\phi = 2c\cdot\frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2}$, the expected cost of Algorithm 5 then can be further bounded by:

$$\begin{aligned}
\mathbb{E}[C] &= \Pr[\delta > \pi(s,v_k^*)]\cdot C_{\delta>\pi(s,v_k^*)} + \Pr[\pi(s,v_k^*)/4 < \delta \le \pi(s,v_k^*)]\cdot C_{\pi(s,v_k^*)/4<\delta\le\pi(s,v_k^*)} \\
&\quad + \sum_{x=1}^{\log_2(n/\pi(s,v_k^*))} \Pr[\pi(s,v_k^*)/2^{i+2} < \delta \le \pi(s,v_k^*)/2^{i+1}]\cdot C_{\pi(s,v_k^*)/2^{i+2}<\delta\le\pi(s,v_k^*)/2^{i+1}} \\
&< 1\cdot\frac{\phi}{\pi(s,v_k^*)} + 1\cdot\frac{\phi}{\pi(s,v_k^*)/4} + \sum_{i=1}^{\log_2(n/\pi(s,v_k^*))}\frac{p_f'}{2^{i+1}}\cdot\frac{\phi}{\pi(s,v_k^*)/2^{i+1}} \\
&< \frac{6\phi}{\pi(s,v_k^*)} = \frac{12c\cdot(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2\cdot\pi(s,v_k^*)}.
\end{aligned}$$

Therefore, the expected time complexity of Algorithm 5 can be bounded by

$$O\left(\min\left\{\frac{\sqrt{m\cdot\log(2/p_f)}}{\epsilon\cdot\sqrt{\pi(s,v_k^*)}},\frac{\log(2/p_f)}{\epsilon^2\cdot\pi(s,v_k^*)}\right\}\right),$$

which finishes the proof. □

It still remains to clarify whether Algorithm 5 returns approximate top-k answers. The following lemma shows that our algorithm returns approximate top-k answer with high probability:

Lemma 5.5. *Algorithm 5 returns an ϵ-approximate top-k answer with at least $1-p_f$ probability.*

Proof. Let $v_1, v_2, \ldots, v_k$ be the returned k nodes by Algorithm 5 and $R = \{v_1^*, v_2^*, \ldots, v_k^*\}$ be the k nodes with the real top-k largest PPR values. According to Lemma 5.3, Algorithm 5 terminates with $\delta \le \pi(s, v_k^*)$ (denoted as Condition $C1$) with at least $1-\log_2\frac{1}{\pi(s,v_k^*)}\cdot n\cdot p_f'/8$ probability. When $C1$ holds, we note that for v_i^*, it is satisfied that, with $1-p_f'/2$ probability, $\hat{\pi}(s,v_i^*)-\pi(s,v_i^*) > \frac{\epsilon}{2}\cdot\pi(s,v_i^*)$. As a result,

$$\hat{\pi}(s,v_i^*)-\pi(s,v_i^*) > \frac{\epsilon}{2}\cdot\pi(s,v_i^*) \text{ for any } 1\le i\le k \tag{13}$$

holds with at least $1-k\cdot p_f'/2$ probability. We denote this condition as $C2$.

We next consider when conditions $C1$ and $C2$ both hold, the probability that the single source FORA fails to provide an ϵ-approximate top-k answer. When $C1$ holds, we know that $\hat{\pi(s,v_i^*)} > (1-\epsilon/2)\cdot\pi(s,v_i^*)$. With this condition, $\hat{\pi}(s,v_i)$ must be larger than $(1-\epsilon/2)\cdot\pi(s,v_i^*)$, since its estimation is ith largest and we know that there are at least i nodes with estimated PPR larger than $(1-\epsilon/2)\cdot\pi(s,v_i^*)$, i.e., $\hat{\pi}(s,v_1^*),\hat{\pi}(s,v_2^*),\ldots,\hat{\pi}(s,v_i^*)$. We say a query fails if there exists a returned node v_i such that:

(1) $\hat{\pi}(s,v_i) > (1-\epsilon/2)\cdot\pi(s,v_i)$,
(2) $\pi(s,v_i) < (1-\epsilon)\cdot\pi(s,v_i^*)$.

Next, we prove that v_i fails with very low probability. Let $\epsilon' = \frac{(1-\epsilon/2)\cdot\pi(s,v_i^*)}{\pi(s,v_i)}-1$.

$$\Pr[\hat{\pi}(s,v_i) > (1-\epsilon/2)\cdot\pi(s,v_i^*)] = \Pr[\hat{\pi}(s,v_i) > (1+\epsilon')\cdot\pi(s,v_i)].$$

Since $\pi(s,v_i) < (1-\epsilon)\pi(s,v_i^*)$, we have that $\epsilon' > \frac{\epsilon/2}{1-\epsilon}$. Also note that

$$\epsilon'\cdot\frac{\pi(s,v_i)}{\pi(s,v_i^*)} = (1-\epsilon/2)-\frac{\pi(s,v_i)}{\pi(s,v_i^*)} > (1-\epsilon/2)-(1-\epsilon) = \epsilon/2.$$

Then, according to Lemma 3.2, it is satisfied that:

$$\begin{aligned}\Pr[\hat{\pi}(s,v_i) > (1-\epsilon/2)\cdot\pi(s,v_i^*)] &= \Pr[(1+\epsilon')\cdot\pi(s,v_i)]\\ &\le \exp\left(-\frac{(\epsilon')^2\cdot\pi(s,v_i)}{r_{sum}\cdot(2+2\epsilon'/3)}\cdot\frac{r_{sum}\cdot(2+\epsilon/3)\cdot\log(2/p_f')}{(\epsilon/2)^2\cdot\delta}\right)\end{aligned}$$

$$\leq \exp\left(-\frac{\epsilon'}{(2+2\epsilon'/3)} \cdot \frac{\epsilon' \cdot \pi(s,v_i)}{\pi(s,v_i^*)} \cdot \frac{\pi(s,v_i^*)}{\delta} \cdot \frac{(2+\epsilon/3)\cdot \log(2/p_f')}{(\epsilon/2)^2}\right)$$

$$\left(\frac{\epsilon'}{2+2\epsilon'/3} \text{ is monotonically increasing, } \frac{\epsilon' \cdot \pi(s,v_i)}{\pi(s,v_i^*)} > \epsilon/2, \frac{\pi(s,v_i^*)}{\delta} > 1\right)$$

$$\leq \exp\left(-\frac{\epsilon/(2-2\epsilon)}{2+\epsilon/(3-3\epsilon)} \cdot \epsilon/2 \cdot 1 \cdot \frac{(2+\epsilon/3)\cdot \log(2/p_f')}{(\epsilon/2)^2}\right)$$

$$\leq \exp\left(-\frac{\epsilon/3+2}{2-5\epsilon/3} \cdot \log(2/p_f')\right) \quad \left(\frac{\epsilon/3+2}{2-5\epsilon/3} > 1\right)$$

$$\leq \exp(-\log(2/p_f')) = p_f'/2.$$

As a result, when $C1$ holds, the probability that the query does not fail on any node is at least $1 - n \cdot p_f'$ by applying union bound on the events that no v_i fails and condition $C2$ holds. Since in the worst case, there exists $\log(n/\pi(s,v_k^*))$ iterations when $C1$ holds. Therefore, we further have that the query returns ϵ-approximate answer with at least

$$1 - \left(\log_2 \frac{1}{\pi(s,v_k^*)} \cdot n \cdot p_f'/8 + \log(n/\pi(s,v_k^*)) \cdot n \cdot p_f'\right) \geq 1 - p_f$$

probability. This finishes the proof. □

6 EXTENSIONS

6.1 Extending *BiPPR* to Whole-Graph SSPPR

Recall from Section 2.2 that in BiPPR, it includes both a forward phase and a backward phase. It is proved in Andersen et al. [2007] that the amortized time complexity for the backward phase is $O(\frac{m}{n \cdot r_{max}})$, and in Lofgren et al. [2016], it shows that the forward phase requires $O(\frac{r_{max} \cdot \log(1/p_f)}{\epsilon^2 \cdot \delta})$ time, given the backward phase threshold r_{max}. Afterwards, they choose $r_{max} = O(\epsilon \cdot \sqrt{\frac{m \cdot \delta}{n \cdot \log(1/p_f)}})$ to minimize the time complexity for the pairwise PPR query, which is $O(\frac{1}{\epsilon}\sqrt{\frac{m \cdot \log(1/p_f)}{n \cdot \delta}})$. To apply BiPPR for whole-graph SSPPR queries, a straightforward approach is to use it to answer n point-to-point PPR queries (i.e., from s to every other node). This, however, leads to a total time complexity of $O(\frac{1}{\epsilon}\sqrt{\frac{mn \cdot \log(1/p_f)}{\delta}})$, which is a factor of $\sqrt{n}$ larger than that of the whole-graph SSPPR FORA.

To improve this, we observe that the n point-to-point PPR queries share the same forward phase, and hence, we can conduct the forward phase once and then re-use its results for all n backward phases. In addition, to reduce the total cost of n backward phases, we can set r_{max} to a larger value; although it would require more random walks to be generated in the forward phase, the tradeoff is still favorable, as the overhead of the forward phase has been significantly reduced by the re-usage of results. Since the backward phase (for all target nodes) has a cost of $O(\frac{m}{r_{max}})$, it can be verified that, by setting $r_{max} = O(\epsilon \cdot \sqrt{\frac{m \cdot \delta}{\log(1/p_f)}})$, the expected time complexity of this optimized version of BiPPR (for SSPPR queries) is

$$O\left(\frac{1}{\epsilon \cdot \sqrt{\delta}}\sqrt{m \cdot \log(1/p_f)}\right),$$

which is identical to that of single-source FORA.

However, as we show in Section 8, the optimized *BiPPR* is significantly outperformed by Whole-Graph SSPPR FORA. The reason is that, even after the aforementioned optimization, BiPPR either degrades to MC when r_{max} is large or still requires performing a backward phase from each node v in G, even if $\pi(s, v)$ is extremely small and can be omitted. In contrast, single-source FORA does not suffer from these deficiencies and avoids examining nodes with very small PPR values. Instead, it performs a forward search phase, followed by a number of random walks from the nodes visited in the search; this process tends to avoid examining nodes with very small PPR values, since those nodes are unlikely to be visited by the forward push or the random walks.

6.2 Extending FORA to Source Distributions

In many real applications of the personalized PageRank, the source s can be a distribution (e.g., on a set of bookmark pages) instead of a single node. We show that our algorithms for single-source-node *FORA* can be extended to the case of arbitrary source distributions.

Let σ be the node distribution that the source node s is sampled from. For any target node t, its personalized PageRank with respect to σ is defined as [Haveliwala 2002; Lofgren et al. 2016]:

$$\pi(\sigma, t) = \sum_{v \in V} \sigma(v) \cdot \pi(v, t),$$

where $\sigma(v)$ is the probability that a sample from σ equals v. To apply our algorithms, we modify Line 1 of Algorithm 1 to set the initial residue of each node v as $\sigma(v)$. Let $\pi^{\circ}(\sigma, v)$ (respectively, $r(\sigma, v)$) denote the reserve (respectively, residue) of node v in the modified version of Forward Push. Then, it is easy to prove that the following invariant holds for the modified version of Forward Push:

$$\pi(\sigma, t) = \pi^{\circ}(\sigma, t) + \sum_{v \in V} r(\sigma, v) \cdot \pi(v, t).$$

In particular, the initial states satisfy the above invariant, and by induction, it can be proved that the invariant still holds after every push operation. Given the above invariant, our algorithms can be applied to compute $\pi(\sigma, t)$ without compromising their asymptotic guarantees. Besides, the indexing scheme presented in Section 3.3 is still applicable, since the maximum number of random walks required for each node is identical to that in the single-source-node algorithms.

6.3 Extending FORA to Global PageRank

Global PageRank can be regarded as the personalized PageRank with a source distribution of $(1/n, 1/n, \ldots, 1/n)$. According to our discussion in Section 6.3, FORA can further be used to calculate the global PageRank. The classic solution for PageRank is the Power-Method, which takes a running time of $O(m \cdot \log \frac{1}{\delta \cdot \epsilon})$ to provide ϵ-approximation for PageRank scores above the threshold δ. To apply the Monte-Carlo method, we can first randomly sample source node, record the number of random walks stops at a node v, and use the fraction of random walks stopped at v as the estimated PageRank. To derive ϵ-approximation, the running time will be $O(\frac{(2\epsilon/3+2)\cdot\log(2/p_f)}{\epsilon^2 \cdot \delta})$. When ϵ is moderate, and $m > \frac{3}{\epsilon^2 \cdot \delta}$, the Monte-Carlo approach achieves a better time complexity.

With the proposed FORA framework, the running time can be bounded by:

$$O\left(\min\left\{\frac{\sqrt{m \cdot \log(2/p_f)}}{\epsilon \cdot \sqrt{\delta}}, \frac{\log(2/p_f)}{\epsilon^2 \cdot \delta}\right\}\right),$$

which actually achieves the best asymptotic performance compared to the two existing solutions when ϵ is moderate and $m > \frac{3}{\epsilon^2 \cdot \delta}$.

Besides, our top-k algorithm can be further extended to output the top-k nodes with the highest PageRank scores with a running time of:

$$O\left(\min\left\{\frac{\sqrt{m\cdot\log(2/p_f)}}{\epsilon\cdot\sqrt{\pi(v_k^*)}},\frac{\log(2/p_f)}{\epsilon^2\cdot\pi(v_k^*)}\right\}\right),$$

where $\pi(v_k^*)$ is the node with the kth largest PageRank score.

7 OTHER RELATED WORK

Apart from the methods discussed in Section 2.2, there exists a plethora of techniques for whole-graph and top-k SSPPR queries. Those techniques, however, are either subsumed by *BiPPR* and *HubPPR* or unable to provide worst-case accuracy guarantees. In particular, a large number of techniques adopt the *matrix-based* approach, which formulates PPR values with the following equation:

$$\boldsymbol{\pi_s} = \alpha\cdot\boldsymbol{e_s} + (1-\alpha)\cdot\boldsymbol{\pi_s}\cdot D^{-1}A, \tag{14}$$

where $\boldsymbol{\pi_s}$ is a vector whose ith element equals $\pi(s,v_i)$, $A\in\{0,1\}^{n\times n}$ is the adjacency matrix of G, and $D\in R^{n\times n}$ is a diagonal matrix in which each ith element on its main diagonal equals the out-degree of v_i. Matrix-based methods typically start from an initial guess of $\boldsymbol{\pi_s}$ and then iteratively apply Equation (14) to refine the initial guess until converge is achieved. Recent work that adopts this approach [Fujiwara et al. 2012; Maehara et al. 2014; Shin et al. 2015; Zhu et al. 2013] propose to decompose the input graph into tree structures or sub-matrices and utilize the decomposition to speed up the PPR queries. The state-of-the-art approach for the single-source and top-k PPR queries in this line of research work is *BEAR*, proposed by Shin et al. [2015]. However, as shown in Wang et al. [2016], the best of these methods is still inferior to *HubPPR* [Wang et al. 2016] in terms of query efficiency and accuracy.

There also exist methods that follow similar approaches to the forward search method [Andersen et al. 2006] described in Section 2.2. Berkhin [2005] proposes to pre-compute the Forward Push results from several important nodes and then use these results to speed up the query performance. Ohsaka et al. [2015] and Zhang et al. [2016] further design algorithms to update the stored Forward Push results on dynamic graphs. Jeh and Widom [2003] propose the backward search algorithm, which (i) is the reverse variant of the Forward Push method, and (ii) can calculate the estimated PPRs from all nodes to a target node t. Zhang et al. [2016] also design the algorithms to update the stored backward push results on dynamic graphs. Nonetheless, none of these solutions in this category provide approximation guarantees for single-source or top-k PPR queries on directed graphs.

In addition, there are techniques based on the Monte-Carlo framework. Fogaras et al. [2005] propose techniques to pre-store the random walk results and use them to speed up the query processing. Nonetheless, the large space consumption of the technique renders it applicable only on small graphs. Backstrom and Leskovec [2011] and Sarma et al. [2013] investigate the acceleration of the Monte-Carlo approach in distributed environments. Lofgren et al. [2014] propose *FastPPR* , which significantly outperforms the Monte-Carlo method in terms of query time. However, *FastPPR* in turn is subsumed by *BiPPR* [Lofgren et al. 2016] in terms of query efficiency. In Lofgren [2015], Lofgren further proposes to combine a modified version of Forward Push, random walks, and the backward search algorithm to reduce the processing time of pairwise PPR queries. Nevertheless, the time complexity of the method remains unclear, since Lofgren does not provide any theoretical analysis on the asymptotic performance of the method in Lofgren [2015]. Wang and Tao [2018] also consider combine the forward random walks and the backward search to accelerate the heavy

hitter queries in personalized PageRank; Wei et al. [2019] find connections between SimRank and PPR, combine the forward random walks with backward search, and propose the PRSim algorithm that can answer SimRank queries with sublinear time on power-law graphs.

Finally, a plethora of research work [Avrachenkov et al. 2011; Bahmani et al. 2011; Fujiwara et al. 2012, 2013; Gupta et al. 2008; Lofgren et al. 2016] study how to efficiently process the top-k PPR queries. Gupta et al. [2008] propose to use Forward Push to return the top-k answers. However, their solutions do not provide any approximation guarantee. Avrachenkov et al. [2011] study how to use Monte-Carlo approach to find the top-k nodes. Nevertheless, the solution does not return estimated PPR values and does not provide any worst-case assurance. Fujiwara et al. [2012, 2013] and Shin et al. [2015] investigate how to speed up the top-k PPR queries with the matrix decomposition approach. These approaches either cannot scale to large graphs or do not provide approximation guarantees.

Most recently, Wei et al. propose the index-free TopPPR [Wei et al. 2018], which combines the Forward Push, random walk, and the backward propagation to answer top-k PPR queries with precision guarantees. However, as we will see in our experiments, our FORA+ actually achieves a better performance than TopPPR when we set $\rho = 0.99$ on large datasets and achieves a better trade-off among the space consumption, query efficiency, and query accuracy. The main reason is that FORA+ can benefit from the indexing scheme while TopPPR cannot. It is also difficult for TopPPR to benefit from indexing scheme, since (i) their random walk adopts the $\sqrt{\alpha}$-random walk, and all the nodes visited will be used to estimate the PPR scores; (ii) the sources of random walks in Monte-Carlo phase are generated randomly in TopPPR and it may result in poor cache performances. In contrast, FORA+ only needs to scan the index structures in order and only stores the destinations in index structure, making it lightweight and cache-friendly.

8 EXPERIMENTS

In this section, we experimentally evaluate our methods for whole-graph SSPPR queries and top-k SSPPR queries. For whole-graph (respectively, top-k) SSPPR queries, we include our index-free method *FORA*, which includes the optimizations as mentioned in Section 4 (respectively, Section 5.2) and their index-based variant, referred to as *FORA+*, against the state-of-the-art. All experiments are conducted on a Linux machine with an Intel 2.9GHz CPU and 200GB memory.

8.1 Experimental Settings

Datasets and query sets. We use six real graphs: *DBLP*, *Web-St*, *Pokec*, *LJ*, *Orkut*, and *Twitter*, which are public benchmark datasets used in recent work [Lofgren et al. 2016; Wang et al. 2016]. We further generate two synthetic datasets using the RMAT random graph generator [Chakrabarti et al. 2004], denoted as *RMAT-1* and *RMAT-2,* to examine the scalability of our proposed top-k algorithms. Note that *RMAT-2* includes 8.6B edges. Moreover, we test our methods on four different game social networks from Tencent Games. Table 3 summarizes the statistics of the datasets. For each dataset, we choose 50 source nodes uniformly at random, and we generate an SSPPR query from each chosen node. In addition, we also generate five top-k queries from each source node, with k varying in $\{100, 200, 300, 400, 500\}$. Note that the maximum k is set to 500 in accordance to Twitter's Who-To-Follow service [Gupta et al. 2013], whose first step requires deriving top-500 PPR results.

Methods. For whole-graph SSPPR queries, we compare our proposed *FORA* and *FORA+* against three methods: (i) the *Monte-Carlo* approach, dubbed as *MC*; (ii) the optimized *BiPPR* for SSPPR queries described in Section 6.1; (iii) *HubPPR*, which is the indexed version of *BiPPR*. We further compare our *FORA* and *FORA+* against the version without the optimization techniques (Ref.

Table 3. Datasets ($K = 10^3, M = 10^6, B = 10^9$)

Name	n	m	Type	Linking Site
DBLP	613.6K	2.0M	undirected	www.dblp.com
Web-St	281.9K	2.3M	directed	www.stanford.edu
Pokec	1.6M	30.6M	directed	pokec.azet.sk
LJ	4.8M	69.0M	directed	www.livejournal.com
Orkut	3.1M	117.2M	undirected	www.orkut.com
Twitter	41.7M	1.5B	directed	twitter.com
RMAT-1	41.7M	1.5B	directed	-
RMAT-2	128M	8.6B	directed	-
X0	26.1M	485.6M	undirected	tencent.com
X1	50.1M	792.0M	undirected	tencent.com
X2	58.2M	1.1B	undirected	tencent.com
X3	74.3M	1.5B	undirected	tencent.com

Section 4), dubbed as *FORA-Basic* and *FORA-Basic+* for the index-free and index-based methods, respectively.

For top-k SSPPR queries, we compare our algorithm with the existing approximate solutions: the single-source *BiPPR*, the top-k algorithm for *HubPPR* in Wang et al. [2016], and the *TopPPR* [Wei et al. 2018]. For *BiPPR* and *HubPPR*, we use the same ϵ, δ, and p_f as FORA. For *TopPPR*, we follow the settings in Wei et al. [2018] and set $\rho = 0.99$. We further extend our top-k algorithm (Ref. Algorithm 5) to the Monte-Carlo approach and denote this algorithm as *MC-Topk*. We also include the *Forward Push* [Andersen et al. 2006] and *TPA* [Yoon et al. 2018] as a baseline for top-k SSPPR queries, and we tune their accuracy control parameters on each dataset separately so their precisions for top-k PPR queries are the same as *FORA* on each dataset. Besides, we also compare our *FORA* and *FORA+* against the version without the top-k optimization techniques (Ref. Section 5.2), dubbed as *FORA-Basic* and *FORA-Basic+* for the index-free and index-based algorithms, respectively.

Parameter setting. Following previous work [Lofgren et al. 2014, 2016; Wang et al. 2016], we set $\delta = 1/n, p_f = 1/n$, and $\epsilon = 0.5$. For our *FORA* and *FORA+*, note that the performance and/or the index size depends on the choice of r_{max}. On the whole-graph queries, for *FORA-Basic* and *FORA-Basic+*, r_{max} is set according to Section 3.2; we then use the balanced strategy to auto-decide the choice of r_{max} for *FORA*; for *FORA+*, we include the optimization technique in Section 4.1 and tune r_{max} varying from r^*_{max} to $7r^*_{max}$ where r^*_{max} is the choice of r_{max} set according to Equation (10). We find that $r_{max} = 2r^*_{max}$ strikes the best trade-off, and therefore use this setting for *FORA+* on the whole-graph queries. For top-k PPR query, we also tune r_{max} to find the best index size for our top-k queries and vary r_{max} from r^*_{max} to $7r^*_{max}$. As we show in the experiment, when r_{max} is set to r^*_{max}, it achieves the best trade-off, and therefore we use this setting in our top-k evaluation. For fair comparison, the index size of *HubPPR* is set to be the same as that of *FORA+* for top-k SSPPR processing and also *FORA-Basic+* (for whole-graph SSPPR processing).

8.2 Whole-graph SSPPR Queries

In our first set of experiments, we evaluate the efficiency of each method for whole-graph SSPPR queries on the six public datasets. Table 4 reports the average query time of each method. Observe that both *FORA* and *BiPPR* achieve better query performance than *MC*, which is consistent with our analysis that the time complexity of *FORA* and *BiPPR* is better than that of *MC*. Moreover,

Table 4. Whole-graph SSPPR Performance (s) (i) ($K = 10^3$)

	MC	*BiPPR*	*HubPPR*	*FORA-Basic*	*FORA*	*FORA-Basic+*	*FORA+*
DBLP	14.2	3.8	2.8	0.8	0.6	0.09	0.05
Web-St	5.4	3.7	1.6	0.03	0.02	0.01	0.01
Pokec	69.1	24.9	19.6	11.26	6.3	0.9	0.4
LJ	163.5	61.4	50.8	15.5	9.9	1.2	0.6
Orkut	230.6	158.2	126.3	40.1	26.4	4.8	1.7
Twitter	4.3K	3.1K	2.4K	513.8	283.1	63.3	29.8

Table 5. Whole-graph SSPPR Performance (s) (ii) ($K = 10^3$)

	FORA-Basic	*FORA*	*FORA-Basic+*	*FORA+*
X0	527.47	259.9	66.7	31.2
X1	957.4	504.9	130.7	60.4
X2	1,072.5	530.4	152.4	63.7
X3	2,340.6	1,163.3	216.8	97.2

FORA is at least four times faster than *BiPPR* on most of the datasets. The reason, as we explain in Section 3.2, is that *BiPPR* either degrades to the *MC* approach when the backward threshold is large or requires conducting a backward search from each node v in G, even if $\pi(s, v)$ is extremely small. In contrast, *FORA* avoids degrading to *MC* and tends to omit nodes with small PPR values, which helps improve efficiency. In addition, *FORA+* achieves significant speedup over *FORA* and is around 10 times faster than the latter on most of the datasets. The *HubPPR* also improves over *BiPPR*, but the improvement is far less than what *FORA+* achieves over *FORA*. Moreover, even without any index, *FORA* is still more efficient than *HubPPR*.

As we can observe from Table 4, with our optimization techniques introduced in Section 4, the index-free method *FORA* improves over *FORA-Basic* by up to 1.8×. Apart from the six public datasets, we further test the effectiveness of our methods on the four social networks from company X. We omit the results for the baseline methods (MC, BiPPR, and HubPPR), since they incur prohibitive processing costs. As shown in Table 5, *FORA* still improves over *FORA-Basic* by more than 2× almost on all datasets, which demonstrates the effectiveness of our proposed optimization technique for online algorithms. For the index-based method, *FORA* improves over *FORA+* by at least twice on almost all datasets, and up to 2.8×. As shown in Table 8, the space consumption required by *FORA+* is twice as that of *FORA-Basic+*, which demonstrates that our optimization technique achieves a good trade-off between query time and space consumption.

8.3 Top-*k* SSPPR Queries

In our second set of experiments, we evaluate the efficiency and accuracy of each method for top-k SSPPR queries. For our methods, we only include our *FORA* and *FORA+* that include the optimizations presented in Section 5.2 to avoid the figures being too crowded. We will examine the effectivess of our optimization techniques in the next set of experiments.

*8.3.1 Top-**k** Query Efficiency.* Figure 1 reports the average query time of each method on four representative datasets: *DBLP*, *Pokec*, *Orkut*, and *Twitter*. (The results on the other two datasets are qualitatively similar and are omitted due to the space constraint.) Note that the y-axis is in log-scale. Recall that *Forward Push* provides no approximation guarantee, and we tune the r_{max} on each dataset separately so it provides the same precision for top-500 SSPPR queries as our *FORA*

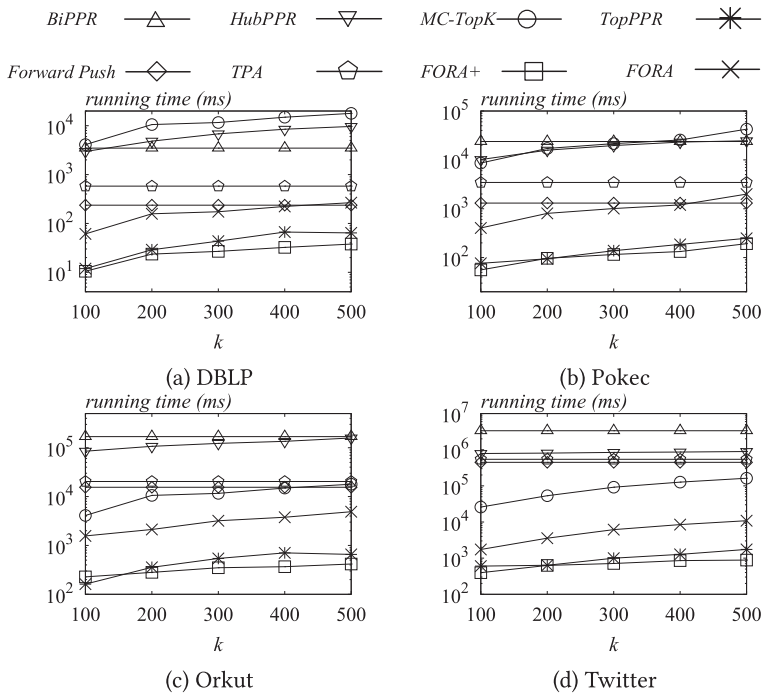


Fig. 1. Top-k SSPPR query efficiency: varying k.

algorithm does. Similarly, for TPA [Yoon et al. 2018], we tune the parameters so it provides the best possible precision.

The main observation is that our *FORA+* achieves the best performance on all the datasets. For instance, on Twitter dataset with $k = 500$, our *FORA+* provides similar or better accuracy and NDCG as competitors and runs 2× faster than TopPPR, two orders of magnitude faster than MC-Topk, and more than 500× faster than Forward Push, TPA, BiPPR, and HubPPR. This is expected, since our FORA+ applies an iterative approach to refine the top-k answers and terminates immediately whenever the answer could provide the desired approximation guarantee; it further uses the indexing scheme to reduce the expensive costs of random walks.

Our FORA is the fastest online algorithm except for TopPPR, since TopPPR uses a more advanced filter-refinement paradigm to answer the top-k queries. However, their proposed approach does not benefit from an indexing scheme and is outperformed by our FORA+. It is also difficult for TopPPR to benefit from an indexing scheme, since (i) their random walk adopts the $\sqrt{\alpha}$-random walk, and all the nodes visited will be used to estimate the PPR scores; (ii) the sources of random walks in Monte-Carlo phase are generated randomly in TopPPR and it may result in poor cache performances. In contrast, FORA+ only needs to scan the index structures in order and only store the destinations in index structure, making it lightweight and cache-friendly.

Another observation is that after extending the MC approach with our top-k algorithm, MC-Topk achieves more than 20× speedup on Twitter over its single-source alternative. This further demonstrates the effectiveness of our top-k algorithm proposed in Section 5.2, which improves the time complexity to $O(\frac{\log n}{\epsilon^2 \cdot \pi(s, v_k^*)})$. However, our FORA and FORA+ are still far more efficient than MC-Topk, since FORA and FORA+ explore the forward push to reduce the random walk costs.

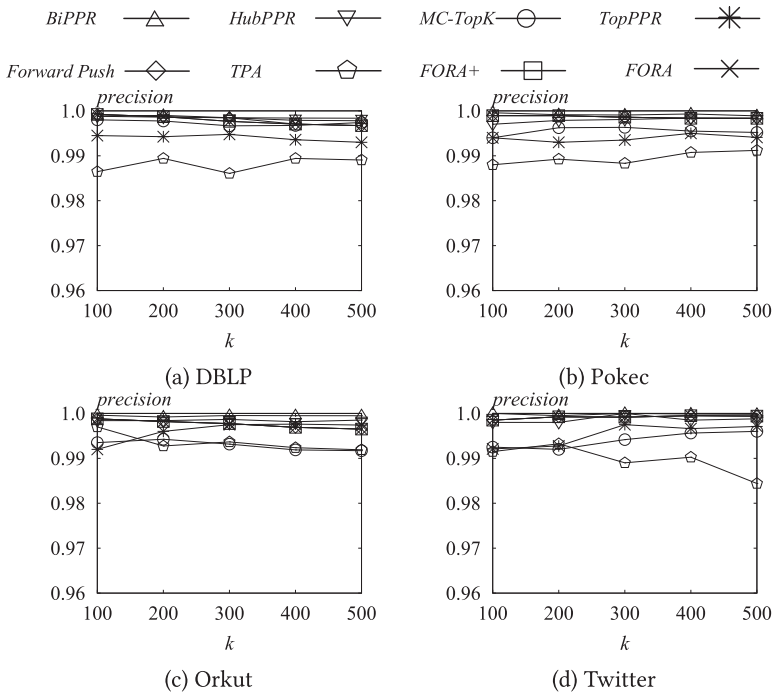


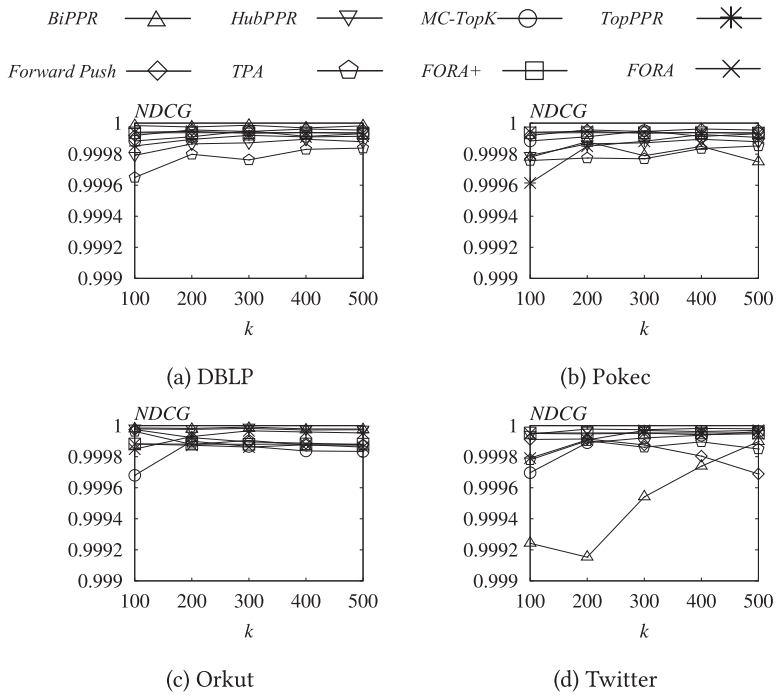
Fig. 2. Top-k SSPPR query accuracy: varying k.

Finally, with our index structure, *FORA+* further improves over *FORA* by an order of magnitude, which demonstrates the effectiveness of the index structure. Notably, on the Twitter social network with 1.5B edges, our *FORA+* can answer the top-500 query in 0.8s.

*8.3.2 Top-**k** Query Accuracy.* To compare the accuracy of the top-k results returned by each method, we first calculate the ground-truth answer of the top-k queries using the *Power Iteration* [Page et al. 1999] method with 100 iterations. Afterwards, we evaluate the top-k results of each algorithm by their precision and NDCG [Järvelin and Kekäläinen 2000] with respect to the ground truth. Note that the precision and recall are the same for the top-k SSPPR queries and is the fraction of nodes returned by the top-k algorithm that are real top-k nodes. For NDCG, let s be the query node, $v_1, v_2, \ldots, v_k$ be the k nodes returned by the top-k algorithm, and $v_1^*, v_2^*, \ldots, v_k^*$ be the true top-k nodes. Then, the NDCG is defined as $\frac{1}{Z_k}\sum_{i=1}^{k}\frac{2^{\pi(s,v_i)}-1}{\log(i+1)}$, where $Z_k = \sum_{i=1}^{k}\frac{2^{\pi(s,v_k^*)}-1}{\log(i+1)}$.

Figure 2 (respectively, Figure 3) shows the accuracy (respectively, NDCG) of the top-k query algorithms on four datasets: *DBLP*, *Pokec*, *Orkut*, and *Twitter*. Observe that all methods except TPA consistently provide high precision. In the meantime, notice that our FORA+ consistently provides similar precision as TopPPR and sometimes even slightly better precision than TopPPR on all the tested datasets. In terms of NDCG, all methods achieve very high NDCG scores, above 0.999 on all datasets.

*8.3.3 Scalability of Top-**k** Algorithms.* In this section, we examine the scalability of our FORA+ using synthetic datasets. We first generate a synthetic dataset with the same size as Twitter and then generate a graph with 8.6B edges. The results are reported in Table 6. As we can see, our FORA+ achieves 3× improvement over TopPPR on both datasets while providing the same accuracy and NDCG. As we will see in Section 8.5, the space consumption of FORA+ is only 2×



(a) DBLP (b) Pokec

(c) Orkut (d) Twitter

Fig. 3. Top-k SSPPR query NDCG: varying k.

Table 6. Scalability Test on Synthetic Datasets ($K = 500$)

	Query Time		**Precision**		**NDCG**	
	RMAT-1	RMAT-2	RMAT-1	RMAT-2	RMAT-1	RMAT-2
FORA+	17.4	149.5	0.993	0.995	0.9999	0.9999
TopPPR	51.1	481.0	0.993	0.995	0.9999	0.9999

and 1.5× that of TopPPR on RMAT-1 and RMAT-2, respectively. This demonstrates that FORA+ achieves a better trade-off between query efficiency and space consumption when providing the same accuracy for the top-k queries.

8.4 Effectiveness of the Top-k Optimization

In this set of experiments, we evaluate the effectiveness of the new top-k algorithm proposed in Section 5.2, which provides improved time complexity and reduces the time to calculate the bounds for each node. We present the results for six representative datasets: *DBLP*, *Pokec*, *Orkut*, *Twitter*, *X2*, and *X3*.

Figures 4(a)–(f) demonstrate the running time of *FORA* and *FORA+* against the versions without the optimization techniques in Section 5.2, referred to as *FORA-Basic* and *FORA-Basic+* for the index-free and index-based solution, respectively. As we can observe, with the new algorithm, *FORA* (respectively, *FORA+)* improves over *FORA-Basic* (respectively, *FORA-Basic+*) by a large margin. For instance, on Twitter dataset, *FORA* (respectively, *FORA+*) improves over *FORA-Basic* (respectively, *FORA-Basic+*) by around 4× (respectively, 6×). The main reason for the significant improvements are two-fold: (i) the time complexity of *FORA* and *FORA+* depend on $\frac{1}{\pi(s, v_k^*)}$ while

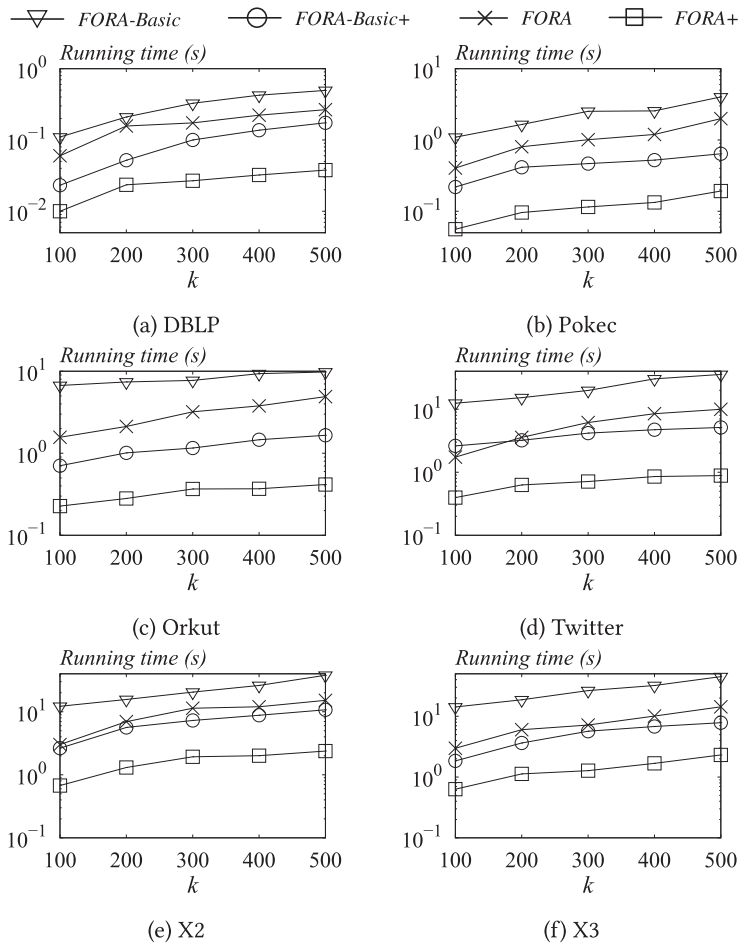


Fig. 4. Effectiveness of top-k optimization: query efficiency.

FORA-Basic and *FORA-Basic+* depend on $\frac{1}{n}$; (ii) *FORA* and *FORA+* avoid the expensive bound calculation part that are required by *FORA-Basic* and *FORA-Basic+*.

Next, we report the accuracy of the four methods on the six datasets as shown in Figures 5(a)–(f). As we can observe, all four methods provide similarly high accuracy for the top-k queries on all datasets. To explain, all four methods share the similar spirit by adaptively refining the top-k answer and returning the approximate answer with theoretical guarantees. Therefore, all the four methods provide similarly high accuracy.

8.5 Preprocessing Costs

Finally, we inspect the preprocessing costs of our methods against alternatives. We first examine the preprocessing time of the index-based methods: *FORA+*, *FORA-Origin+*, *HubPPR*, and *TPA*. Note that the r_{max} of *FORA+* for whole-graph and top-k queries are different, and therefore the preprocessing times are shown as separately in Table 7. The choice of tuned r_{max} of *FORA+* is the same as that of *FORA-Basic*, and therefore their preprocessing times are the same. As shown in Table 7, the preprocessing time of *FORA+*, *FORA-Basic+*, *HubPPR*, and *TPA* are all moderate. On the largest dataset *X3*, our *FORA+* for whole graph SSPPR queries (respectively, for top-k SSPPR

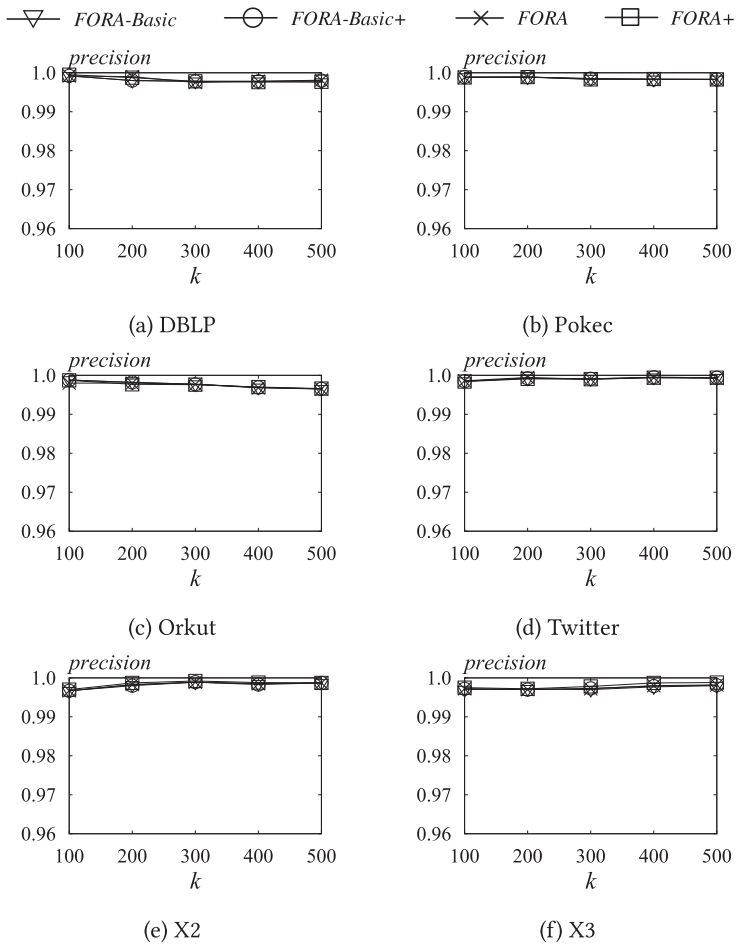


Fig. 5. Effectiveness of top-k Optimization: query accuracy.

queries) can still finish index construction in less than 3h (respectively, less than 1.5h), which is more than compensated by its high query performance as shown in Table 4. Besides, this preprocessing time can be further significantly reduced by parallelizing the index construction process.

Next, we examine the space consumption of all methods. Note that each method will need to store at least a copy of the graph. For *MC*, *FORA*, and *TPA*, they can store only a single copy of the input graph (each node stores the out-neighbor list). However, for *HubPPR*, *BiPPR*, and *TopPPR*, they need to store two copies of the graph (each node stores an out-neighbor and an in-neighbor list) for efficient random walks and backward propagation.

Table 8 reports the space consumption of all methods. As we can observe, the index size of our *FORA+* for whole-graph queries and top-k queries are no more than 7.5× and 4× of the original graph, respectively. The space consumption is more than compensated by the superb performance of *FORA+*, where we can answer a whole-graph query within 30s and top-k query within 0.9s on the 1.5B edge Twitter graph, which improves over the index-free *FORA* by more than an order of magnitude on almost all datasets. This demonstrates the effectiveness and efficiency of our indexing scheme.

Table 7. Preprocessing Time

Datasets	*HubPPR*	*TPA*	*FORA+ (for whole graph)*	*FORA-Basic+ & FORA+ (for top-k)*
DBLP	26.4	4.4	7.6	3.6
Web-St	9.3	1.3	3.2	1.5
Pokec	90.6	42.5	77.1	34.6
LJ	279.9	112.5	181.3	81.2
Orkut	530.6	177.3	383.5	165.9
Twitter	5,088.6	3,112.3	5,130.4	2,255.4
RMAT-1	-	-	-	2,695.2
RMAT-2	-	-	-	12,076.8
X0	-	-	2,031.2	967.14
X1	-	-	5,244.5	2,428.6
X2	-	-	5,721.7	2,602.9
X3	-	-	8,836.4	4,213.5

Table 8. Space Consumption

Datasets	*MC/FORA*	*BiPPR/TopPPR*	*HubPPR*	*TPA*	*FORA-Basic+ & FORA+* (for top-k)	*FORA+* (for whole-graph)
DBLP	18.4MB	36.8MB	127.6MB	23.3MB	109.2MB	200.1MB
Web-St	10.4MB	20.8MB	66.3MB	12.6MB	55.9MB	101.4MB
Pokec	130.8MB	261.5MB	673.5MB	143.6MB	542.7MB	954.6MB
LJ	295.2MB	590.5MB	1.7GB	333.6MB	1.4GB	2.5GB
Orkut	950.1MB	1.9GB	3.5GB	974.9MB	2.6GB	4.2GB
Twitter	6.2GB	12.5GB	25.1GB	6.5GB	18.8GB	31.4GB
RMAT-1	6.2GB	12.5GB	-	-	20.0GB	-
RMAT-2	33.3GB	66.5GB	-	-	96.3GB	-
X0	2.0GB	-	-	-	7.5GB	13.1GB
X1	3.2GB	-	-	-	13.2GB	23.2GB
X2	4.6GB	-	-	-	17.5GB	30.4GB
X3	6.0GB	-	-	-	22.6GB	39.2GB

Compared to TopPPR, FORA+ achieves 2×, 3×, and 3.4× improvement on Twitter, RMAT-1, and RMAT-2, with only 1.5×, 1.6×, and 1.4× space consumption, respectively. The experimental results demonstrate that our proposed methods achieve a good trade-off between the space consumption and query efficiency when providing query answers with identical accuracy.

8.6 Tuning r_{max}

Finally, we examine the trade-off between the space consumption and query performance of our *FORA+* algorithm by tuning r_{max} for whole graph queries and top-k queries. Let r^*_{max} be the r_{max} value set according to Section 3.2. We then vary r_{max} from r^*_{max} to $7r^*_{max}$. Notice that the index size is proportional to r_{max} and let y be the index size when $r_{max} = r^*_{max}$. Then, if $r_{max} = 3r^*_{max}$, then the index size will be $3y$. As we can observe from Figures 6 and 7, when we increase the index size for *FORA+*, the query performance also improves on almost all datasets for both whole-graph queries and top-k queries. To explain, by increasing the index size, *FORA+* avoids the expensive

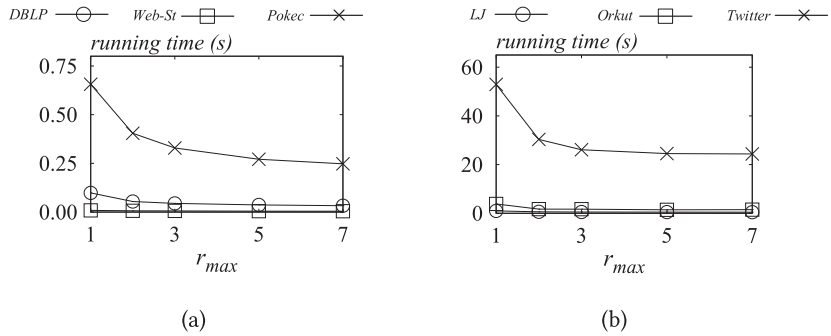


Fig. 6. Tuning r_{max} for whole-graph SSPPR queries ($\times r^*_{max}$).

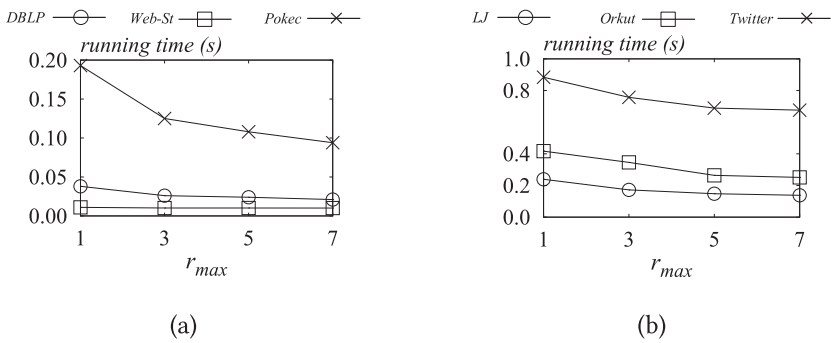


Fig. 7. Tuning r_{max} for top-k SSPPR queries ($\times r^*_{max}$).

random walks, thus saving the query time. However, from the experiment, we can observe that when $r_{max} = 2r^*_{max}$ (respectively, $rmax = r^*_{max}$), FORA+ actually achieves the best trade-off between the query performance and space consumption on whole-graph queries (respectively, top-k queries), and hence in our experiment, we set $r_{max} = 2r^*_{max}$ (respectively, $r_{max} = r^*_{max}$) for all the whole-graph queries (respectively, top-k queries).

9 CONCLUSION

We present FORA, a novel algorithm for approximate single-source personalized PageRank computation. The main ideas include (i) combining Monte-Carlo random walks with Forward Push in a non-trivial and optimized way; (ii) pre-computing and indexing random walk results; and (iii) additional pruning based on top-k selection. Compared to existing solutions, FORA involves a reduced number of random walks, avoids expensive backward searches, and provides rigorous guarantees on result quality. Extensive experiments demonstrate that FORA outperforms existing solutions by a large margin and enables fast responses for top-k SSPPR searches on very large graphs with little computational resources.

APPENDIX

A PROOF OF THEOREM

Proof of Theorem 5.1. Given ω_j, we can derive that:

$$\Pr[|\pi(s,t) - \hat{\pi}(s,t)| \geq \epsilon \cdot \pi(s,t)] \leq 2 \cdot \exp\left(-\frac{\epsilon^2 \cdot \omega_j \cdot \pi(s,t)}{2 + 2a \cdot \epsilon/3}\right).$$

Since $a \leq 1$ and $\pi(s,t) \geq \pi^{\circ}(s,t)$, and $\pi(s,t) \geq LB_{j-1}(t)$, we can derive that:

$$\Pr[|\pi(s,t) - \hat{\pi}(s,t)| \geq \epsilon \cdot \pi(s,t)] \leq 2\exp\left(-\frac{\epsilon^2 \cdot \omega_j \cdot \max\{\pi_j^{\circ}(s,v), LB_{j-1}(s,v)\}}{2 + 2\epsilon/3}\right).$$

Let p'_f equal the RHS of the above inequality. We have $\epsilon \geq \sqrt{\frac{3\log(2/p'_f)}{\omega_j \cdot \max\{\pi_j^{\circ}(s,v), LB_{j-1}(s,v)\}}}$.

By setting $\epsilon_j = \sqrt{\frac{3\log(2/p'_f)}{\omega_j \cdot \max\{\pi_j^{\circ}(s,v), LB_{j-1}(s,v)\}}}$, we can derive that $\hat{\pi}_j(s,v)/(1+\epsilon_j) \leq \pi(s,v) \leq \hat{\pi}_j(s,v)/(1-\epsilon_j)$ holds with $1 - p'_f$ probability. Similarly, we have

$$\Pr[|\pi(s,t) - \hat{\pi}(s,t)| \geq \epsilon \cdot \pi(s,t)] \leq 2\exp\left(-\frac{\lambda^2 \cdot \omega}{r_{sum} \cdot (2\pi(s,t) + 2\lambda/3)}\right).$$

Let p'_f equal the RHS of the above inequality and note that $\pi(s,t) \leq UB_{j-1}(t)$. This helps us derive the designed bound for λ_j.

Proof of Theorem 5.2. We apply a similar technique in Wang et al. [2016]. If $LB_j(v'_i) \cdot (1+\epsilon) > UB_j(v'_i)$. Then, it can be derived that

$$\hat{\pi}(v'_i) \leq UB_j(v'_i) \leq LB_j(v'_i) \cdot (1+\epsilon) \leq (1+\epsilon) \cdot \pi(s, v'_i).$$
$$\hat{\pi}(v'_i) \geq LB_j(v'_i) \geq UB_j(v'_i)/(1+\epsilon) \geq (1-\epsilon) \cdot \pi(s, v'_i).$$

Hence, $v'_1, \ldots, v'_k$ satisfy Equation (2). Let $v_1, \ldots, v_k$ be the k nodes that have the top-k exact PPR values. Assume that all bounds are correct, then as $LB(v'_l) \geq \delta$, it indicates that the top-k PPR values are no smaller than δ. In this case, all the top-k nodes should satisfy ϵ-approximation guarantee, i.e., they satisfy that $UB(v_i) < (1+\epsilon) \cdot LB(v_i)/(1-\epsilon)$.

Let $UB'_j(1), UB'_j(2), \ldots UB'_j(k)$ be the top-k largest PPR upper bounds in the jth iteration. Note that the ith largest PPR satisfies that $UB'_j(i) \geq \pi(s, v_i) \geq LB_j(s, v'_i)$.

Now assume that one of the upper bounds, say $UB'_j(i)$, is not from $UB(v_1), \ldots, UB(v_k)$. If it satisfies that $LB_j(v'_k) \cdot (1+\epsilon) \geq UB'_j(i)$, then it indicates that $LB_j(v'_i) \cdot (1+\epsilon) \geq UB'_j(i)$ for all nodes. Hence, we update the node whose upper bound is minimum among $UB_j(v'_1), \ldots, UB_j(v'_1)$, we can still guarantee that $LB_j(v'_i) \cdot (1+\epsilon) > UB_j(v'_i)$. We repeat this process until $UB_j(v'_1), \ldots, UB_j(v'_k)$ are the top-k upper bounds. However, let U be the set of nodes such that the node $u \in U$ satisfies that $UB_j(u) < LB_j(v'_k)$. Then it is still possible that these nodes are from the exact top−k answers. However, recall that if a node $u \in U$ is from the top-k, it should satisfy that $UB(u) < (1+\epsilon) \cdot LB(u)/(1-\epsilon)$. As a result, if there exists no node $u \in U$ such that $UB(u) < (1+\epsilon) \cdot LB(u)/(1-\epsilon)$, then no node $u \in U$ is from the top-k answers, in which case it will not affect the approximation guarantee.

Afterwards, we proceed a bubble sort on the top-k upper bounds in decreasing order. If we replace two upper bounds $UB_j(v'_x)$ and $UB_j(v'_y)$ with $x < y$, then $UB_j(v'_x) < UB_j(v'_y)$. Also $LB_j(v'_x) > LB_j(v'_y)$ from the definition. As

$$UB_j(v'_x)/LB_j(v'_y) < UB_j(v'_y)/LB_j(v'_y) \leq (1+\epsilon),$$
$$UB_j(v'_y)/LB_j(v'_x) < UB_j(v'_y)/LB_j(v'_y) \leq (1+\epsilon).$$

When the sort finishes, the inequations still hold. We then have

$$UB'_j(1) \leq (1+\epsilon) \cdot LB_j(v'_i) \ldots, UB'_j(k) \leq (1+\epsilon) \cdot LB_j(v'_k).$$

Also note that

$$\hat{\pi}(v_i') \geq LB_j(v_i') \geq \frac{1}{1+\epsilon} UB_j'(i) \geq (1-\epsilon) \cdot \pi(s, v_i),$$

for all $i \in [1, k]$. So, the answer provides approximation guarantee if the bounds from the first to the jth iteration are all correct. By applying the union bound, we can obtain that the approximation is guaranteed with probability at least $1 - n \cdot j \cdot p_f'$.

REFERENCES

Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. 2007. Local computation of PageRank contributions. In *Proceedings of the WAW*. 150–165.

Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. 2006. Local graph partitioning using PageRank vectors. In *Proceedings of the FOCS*. 475–486.

Konstantin Avrachenkov, Nelly Litvak, Danil Nemirovsky, Elena Smirnova, and Marina Sokol. 2011. Quick detection of Top-k personalized PageRank lists. In *Proceedings of the WAW*. 50–61.

Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: Predicting and recommending links in social networks. In *Proceedings of the WSDM*. 635–644.

Bahman Bahmani, Kaushik Chakrabarti, and Dong Xin. 2011. Fast personalized PageRank on MapReduce. In *Proceedings of the SIGMOD*. 973–984.

Pavel Berkhin. 2005. Survey: A survey on PageRank computing. *Int. Math.* 2, 1 (2005), 73–120.

Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. 2004. R-MAT: A recursive model for graph mining. In *Proceedings of the SDM*. 442–446.

Fan R. K. Chung and Lincoln Lu. 2006. Survey: Concentration inequalities and martingale inequalities: A survey. *Int. Math.* 3, 1 (2006), 79–127.

Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. 2005. Towards scaling fully personalized PageRank: Algorithms, lower bounds, and experiments. *Int. Math.* 2, 3 (2005), 333–358.

Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Takeshi Mishima, and Makoto Onizuka. 2013. Efficient ad hoc search for personalized PageRank. In *Proceedings of the SIGMOD*. 445–456.

Yasuhiro Fujiwara, Makoto Nakatsuji, Takeshi Yamamuro, Hiroaki Shiokawa, and Makoto Onizuka. 2012. Efficient personalized PageTank with accuracy assurance. In *Proceedings of the KDD*. 15–23.

Manish S. Gupta, Amit Pathak, and Soumen Chakrabarti. 2008. Fast algorithms for top-k personalized PageRank queries. In *Proceedings of the WWW*. 1225–1226.

Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. 2013. WTF: The Who To Follow service at Twitter. In *Proceedings of the WWW*. 505–514.

Taher H. Haveliwala. 2002. Topic-sensitive PageRank. In *Proceedings of the WWW*. 517–526.

Kalervo Järvelin and Jaana Kekäläinen. 2000. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the SIGIR*. 41–48.

Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *Proceedings of the WWW*. 271–279.

Peter Lofgren. 2015. Efficient algorithms for personalized PageRank. Retrieved from: *CoRR* abs/1512.04633 (2015).

Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2016. Personalized PageRank estimation and search: A bidirectional approach. In *Proceedings of the WSDM*. 163–172.

Peter A. Lofgren, Siddhartha Banerjee, Ashish Goel, and C. Seshadhri. 2014. FAST-PPR: Scaling personalized PageRank estimation for large graphs. In *Proceedings of the KDD*. 1436–1445.

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

Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. 2015. Efficient PageRank tracking in evolving networks. In *Proceedings of the SIGKDD*. 875–884.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report. Stanford University, Stanford, CA.

Atish Das Sarma, Anisur Rahaman Molla, Gopal Pandurangan, and Eli Upfal. 2013. Fast distributed PageRank computation. In *Proceedings of the ICDCN*. 11–26.

Kijung Shin, Jinhong Jung, Lee Sael, and U. Kang. 2015. BEAR: Block elimination approach for random walk with restart on large graphs. In *Proceedings of the SIGMOD*. 1571–1585.

Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. HubPPR: Effective indexing for approximate personalized PageRank. *PVLDB* 10, 3 (2016), 205–216. Retrieved from: http://www.vldb.org/pvldb/vol10/p205-wang.pdf.

Sibo Wang and Yufei Tao. 2018. Efficient algorithms for finding approximate heavy hitters in personalized PageRanks. In *Proceedings of the SIGMOD*. 1113–1127.

Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: Simple and effective approximate single-source personalized PageRank. In *Proceedings of the SIGKDD*. 505–514.

Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Yu Liu, Xiaoyong Du, and Ji-Rong Wen. 2019. PRSim: Sublinear time SimRank computation on large power-law graphs. In *Proceedings of the SIGMOD*. 1042–1059.

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 *Proceedings of the SIGMOD*. 441–456.

Minji Yoon, Jinhong Jung, and U. Kang. 2018. TPA: Fast, scalable, and accurate method for approximate random walk with restart on billion scale graphs. In *Proceedings of the ICDE*.

Hongyang Zhang, Peter Lofgren, and Ashish Goel. 2016. Approximate personalized PageRank on dynamic graphs. In *Proceedings of the KDD*. 1315–1324.

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

Received August 2018; revised March 2019; accepted August 2019