Efficient Personalized PageRank Computation: A Spanning Forests Sampling Based Approach

Meihao Liao
Beijing Institute of Technology
Beijing, China
mhliao@bit.edu.cn

Rong-Hua Li
Beijing Institute of Technology
Beijing, China
lironghuabit@126.com

Qiangqiang Dai
Beijing Institute of Technology
Beijing, China
qiangd66@gmail.com

Guoren Wang
Beijing Institute of Technology
Beijing, China
wanggrbit@126.com

ABSTRACT

Computing the personalized PageRank vector is a fundamental problem in graph analysis. In this paper, we propose several novel algorithms to efficiently compute the personalized PageRank vector with a decay factor α based on an interesting connection between the personalized PageRank values and the weights of random spanning forests of the graph. Such a connection is derived based on a newly-developed matrix forest theorem on graphs. Based on this, we present an efficient spanning forest sampling algorithm via simulating loop-erased α-random walks to estimate the personalized PageRank vector. Compared to all existing methods, a striking feature of our approach is that its performance is insensitive w.r.t. (with respect to) the parameter α. As a consequence, our algorithm is often much faster than the state-of-the-art algorithms when α is small, which is the demanding case for many graph analysis tasks. We show that our technique can significantly improve the efficiency of the state-of-the-art algorithms for answering two well-studied personalized PageRank queries, including single source query and single target query. Extensive experiments on seven large real-world graphs demonstrate the efficiency of the proposed method.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**.

KEYWORDS

Personalized PageRank; Spanning Forest

ACM Reference Format:
Meihao Liao, Rong-Hua Li, Qiangqiang Dai, and Guoren Wang. 2022. Efficient Personalized PageRank Computation: A Spanning Forests Sampling Based Approach. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3514221.3526140

1 INTRODUCTION

Given a graph $G = (V, E)$, two nodes $s, t \in V$, and a decay factor α, the Personalized PageRank (PPR) $\pi(s, t)$ is defined as the probability that a random surfer starts from s stops at t when applying an α-random walk, where a random surfer randomly stops at the current node with probability α or travels to a neighbor of the current node with probability $1 - \alpha$. Clearly, by this definition, the PPR value $\pi(s, t)$ naturally measures the importance of node t w.r.t. (with respect to) s. That is, after randomly surfing, if s stops at t with a high probability, then t is important to s. Based on such a nice property, PPR has been widely used in web search related applications [28].

However, a recent trend is to employ PPR for many graph analysis tasks, such as graph clustering [4, 41], graph embedding [51], and graph neural networks [13]. In particular, PPR values can express the graph structure information by considering all paths between two nodes, which can be viewed as a type of information propagation procedure across the graph [42]. This new trend brings significant efficiency and effectiveness improvement to graph analysis tasks, but also introduces new computational challenges.

The new applications of PPR in graph analysis often require a small decay factor α. For example, for local graph clustering application, the optimal parameter setting for the decay factor is $\alpha = 0.01$ as reported in [41]. The same optimal parameter setting of α can also be found in graph neural network application [13]. The reason could be that with a small α, the α-random walk can explore a large portion of the graph, thus can obtain more information compared to the case with a large α. But unfortunately, all existing algorithms for computing the PPR values with a small α (e.g., $\alpha = 0.01$) are not very efficient on large graphs, which motivates us to develop more efficient algorithms to handle the small α case.

Specifically, previous PPR computation methods can be divided into two categories, including deterministic methods [3, 4, 10, 28] and Monte Carlo algorithms [7, 33]. In our case, of particular interest is the Monte Carlo algorithm. To estimate the $\pi(s, t)$ value, a classic Monte Carlo algorithm first simulates α-random walks from s and then counts the fraction of walks that terminates at t as an estimation. However, the major drawback of this method is that it only cares about the end-node of each random walk, and other nodes in the random walk are totally ignored. Moreover, the efficiency of such a method is heavily dependent on the decay factor α. For a small α, such a classic Monte Carlo method is inefficient, because it often takes a long time to simulate an α-random walk in this case.

To tackle the issues, we propose a novel solution based on an interesting connection between PPR and random spanning forests of the graph. We first establish a novel PageRank matrix forest theorem, which gives a new combinatorial explanation of PPR value $\pi(s, t)$ as the probability that s is rooted in t in a rooted random

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3526140

spanning forest.[1] Such a combinatorial explanation motivates us to design efficient PPR computation algorithms by sampling spanning forests. To this end, we propose a new loop-erased α-random walk technique to generate random spanning forests via extending the classic Wilson algorithm [48]. Compared to the previous methods, our spanning forest sampling based technique has several appealing features: (1) each step in the α-random walk except loops provides valuable information for estimating PPR values, and (2) the expected running time of our technique will not grow rapidly as α decreases, thus it is insensitive w.r.t. α.

We apply the newly-developed technique to improve previous algorithms for answering two PPR queries: single source PPR query and single target PPR query. For the single source PPR query, the state-of-the-art algorithm is a two-stage algorithm [46, 49] that combines deterministic forward push and Monte Carlo. Although several optimizations are made to accelerate the deterministic forward push stage [32, 49], there is little work focusing on optimizing the Monte Carlo stage. In the Monte Carlo stage, all these methods just simply simulates α-random walk, which is inefficient when α is small. We show that our technique can improve the Monte Carlo stage by replacing the α-random walk sampling with the proposed random spanning forest sampling. Note that our random spanning forest sampling technique is orthogonal to those optimizations made on deterministic forward push [32, 49], thus they can also be applied to optimize our solutions. For the single target PPR query, the state-of-the-art algorithm is the backward push algorithm [3]. Similarly, the backward push algorithm is also slow when α is small especially for high-degree nodes. To overcome this issue, we also propose a two-stage algorithm combining backward push and sampling spanning forests. We show that our algorithm can achieve a relative error guarantee in theory. Finally, we conduct extensive experiments using 7 large real-world graphs to evaluate our algorithms. The results show that (1) for the single source PPR query, our best algorithm can achieve one order of magnitude speedup over the state-of-the-art algorithm [49] on large graphs when $\alpha = 0.01$, and (2) for the single target PPR query, our best algorithm is around 3× faster than the state-of-the-art algorithm. To summarize, the main contributions of this paper are as follows.

New theoretical results. We develop three novel matrix forest theorems, based on which the PPR value $\pi(s,t)$ between two nodes s and t can be explained as the probability that s is rooted in t in a random spanning forest. We show that the proposed matrix forest theorems can be applied to develop an efficient algorithm to estimate the PPR values by sampling spanning forests in a graph. We believe that such a novel combinatorial explanation for the PPR values could be of independent interest.

New algorithms for PPR queries. We first propose a new algorithm to sample spanning forests based on a loop-erased α-random walk technique. We show that such a loop-erased α-random walk technique is insensitive w.r.t. α. Based on this technique, we develop a new two-stage algorithm which combing forward push (backward push) and spanning forest sampling to efficiently answer the single source (target) PPR query.

[1] A rooted spanning forest partitions the graph into several connected components (each tree is a connected component). If a node s is located in a tree that has a node t as its root, we call that node s is rooted in t.

Extensive experiments. We conduct extensive experiments using 7 real-world graphs to evaluate the efficiency of the proposed algorithms. The results show that our algorithms significantly outperform the state-of-the-art algorithms on all datasets. For reproducibility purpose, the source code of this paper is released at https://github.com/mhLeon/RSFPPR.

2 PRELIMINARIES

Let $G = (V, E, W)$ be a weighted graph, where V ($n = |V|$) is a set of vertices, E ($m = |E|$) is a set of edges, and $w_{uv} \in W$ denotes the weight of an edge $e = (u, v)$. Denote by A, the adjacency matrix of G with $A_{uv} = w_{uv}$ if $(u, v) \in E$, $A_{uv} = 0$ otherwise. Let $L = D - A$ be the Laplacian matrix of G, where D is a diagonal matrix with each entry $D_{ii} = \sum_j A_{ij}$. For a node $u \in V$, the weighted degree of u, denoted by d_u, is equal to D_{uu}. If the graph is unweighted, the weighted degree is exactly equal to the number of neighbors. For easy understanding of our results, we assume that the graph G is undirected in the following sections. It is important to note that the main technique and all theoretical results presented in Section 3 and Section 4 still work for directed graphs.

Given a source node s, a target node t, and a decay factor α, the personalized PageRank (PPR) of t w.r.t. s, denote by $\pi(s,t)$, is the probability that an α-random walk starting from s terminates at t. Here an α-random walk is a random walk where in each step the random walk stops at the current node with probability α and travels to a random neighbor with probability $1 - \alpha$. With this definition, we mainly focus on two types of personalized PPR computation problem in this paper.

Given a source node s, the single source PPR query is to compute $\pi(s, v)$ for each node $v \in V$. The answer of this query is a row vector $p_s \in R^{1\times n}$. Similarly, given a target node t, the single target PPR query is to compute $\pi(v, t)$ for each $v \in V$, and the answer of this query is a column vector $p_t \in R^{n\times 1}$. Below, we focus mainly on describing the concepts of single source PPR query, and similar concepts can also be applied for the single target PPR query.

Let $P = D^{-1}A$ be the probability transition matrix where each row is normalized by the weighted degree. The PPR vector p_s, which is PPR value of all nodes w.r.t. the source node s, satisfying the following linear equation

$$p_s = \alpha e_s + (1 - \alpha) p_s \cdot P, \tag{1}$$

where $e_s \in R^{1\times n}$ is the unit vector with 1 on the s-th element and 0 on others. Then, we have

$$\pi(s, v) = p_s[v] = \alpha[I - (1 - \alpha)P]^{-1}_{sv}. \tag{2}$$

Note that Eq. (1) can be easily reformulated as an equivalent linear system:

$$\tilde{p_s}(L + \beta D) = \beta e_s, \tag{3}$$

where $\beta = \alpha/(1 - \alpha)$, $\tilde{p_s} = p_s D^{-1}$ and L is the Laplacian matrix. Based on Eq. (3), we can easily derive that

$$\pi(s, v) = p_s[v] = [(L + \beta D)^{-1} \beta D]_{sv}. \tag{4}$$

Based on Eq. (4), we define the β-Laplacian matrix as follows.

Definition 2.1. (β-Laplacian) Given a graph $G = (V, E)$ and its Laplacian matrix $L = D - A$. Let α be the decay factor of the α-random walk. The β-Laplacian of G with parameter α is defined as $L_\beta = (\beta D)^{-1}(L + \beta D)$, where $\beta = \frac{\alpha}{1-\alpha}$.

Clearly, by Eq. (4) and Definition 2.1, we have $\pi(s,t) = (L_\beta^{-1})_{st}$. Thus, answering the PPR queries is equivalent to computing the inverse of the β-Laplacian matrix. Clearly, the answer of the single source query is a row of L_β^{-1}, while the answer of the single target query is a column of L_β^{-1}. Computing both the single source and single target queries are often very costly for large graphs, thus many approximation algorithms with a relative error guarantee have been proposed in the literature [32, 46, 49]. Below, we formally define such two approximate query processing problems which will be served as two applications of the proposed technique.

Definition 2.2. (Approximate single source PPR query) Given a relative error threshold $\epsilon > 0$, PPR threshold μ and a source node s, an approximate single source PPR query problem aims to compute an estimation $\tilde{\pi}(s,v)$ for each node $v \in V$ with $\pi(s,v) \geq \mu$ such that $|\tilde{\pi}(s,v) - \pi(s,v)| \leq \epsilon\pi(s,v)$ with a low failure probability p_f.

Definition 2.3. (Approximate single target PPR query) Given a relative error threshold $\epsilon > 0$, PPR threshold μ and a target node t, an approximate single target PPR query problem aims to compute an estimation $\tilde{\pi}(v,t)$ for each node $v \in V$ with $\pi(v,t) \geq \mu$ such that $|\tilde{\pi}(v,t) - \pi(v,t)| \leq \epsilon\pi(v,t)$ with a low failure probability p_f.

The two parameters, μ and p_f, are used to bound the estimation quality. The parameter μ is a threshold such that we can achieve a relative error guarantee if the exact PPR value exceeds μ; p_f is the failure probability that the algorithm will produce a wrong result. Following previous studies [32, 46, 49], we set $\mu = \frac{1}{n}$ and $p_f = \frac{1}{n}$ to guarantee a relatively precise result. With these settings, μ and p_f are both small enough to achieve a high estimating precision.

3 PAGERANK MATRIX FOREST THEOREM

The classic matrix tree (or matrix forest) theorem connects the number of spanning trees (or forests) to the determinant of the Laplacian matrix of a graph which is perhaps the most well-known result in spectral graph theory [16]. In this section, we establish a novel matrix forest theorem, referred to as the PageRank matrix forest theorem, based on the β-Laplacian matrix defined in Definition 2.1.

3.1 New matrix forest theorems

For a forest F, the weight of F is defined as the product of all weights of edges in F, that is $w(F) = \prod_{e \in F} w_e$. For unweighted graphs, we simply have $w_e = 1$, and thus $w(F)$ is also equal to 1 for all F. A spanning forest of G is a forest including all nodes in G. Note that a forest may have several connected tree components. A rooted spanning forest is a spanning forest where we specify one node as a root in each connected component. For convenience, if a node s belongs to a tree $\mathcal{T}$ (each tree is a connected component in the forest) which has a node t as its root, we say that s is rooted in t in the spanning forest. We denote $\rho(F)$ as the set of roots of F. The following result shows a connection between the determinant of L_β and the weights of the rooted spanning forests.

Theorem 3.1. *(PageRank matrix forest theorem) Given a graph $G = (V, E, W)$, for $\beta \in (0, \infty)$, the determinant of L_β is related to the rooted spanning forests as follows:*

$$det(L_\beta) = \frac{1}{\beta^n \prod\limits_{u \in V} d_u} \sum_{F \in \mathcal{F}} w(F) \prod_{u \in \rho(F)} \beta d_u,$$

where $\mathcal{F}$ denotes the set of all rooted spanning forests in G.

Based on Theorem 3.1, we can further develop two matrix forest theorems based on the minors of the matrix L_β as follows. Due to the space limits, all the missing proofs can be found in the full version of this paper [31].

Theorem 3.2. *Given a graph $G = (V, E, W)$, for $\beta \in (0, \infty)$, the determinant of the principle minor $L_\beta^{(v)}$, obtained by deleting the v-th row and column is related to the rooted spanning forests as follows:*

$$det(L_\beta^{(v)}) = \frac{1}{\beta^n \prod\limits_{u \in V} d_u} \sum_{F \in \mathcal{F}_v} w(F) \prod_{u \in \rho(F)} \beta d_u,$$

where $\mathcal{F}_v$ denotes the set of all rooted spanning forests in G having v as a root.

Theorem 3.3. *Given a graph $G = (V, E, W)$, for $\beta \in (0, \infty)$, given two distinct vertices u,v, the determinant of the minor $L_\beta^{(u,v)}$, obtained by deleting the u-th row and v-th column is related to the rooted spanning forests as follows:*

$$det(L_\beta^{(u,v)}) = \frac{1}{\beta^n \prod\limits_{i \in V} d_i} \sum_{F \in \mathcal{F}_{v,u}} w(F) \prod_{u \in \rho(F)} \beta d_u,$$

where $\mathcal{F}_{v,u}$ denotes the set of all rooted spanning forests in G in which u and v are in the same connected component and u is a root.

Note that although we focus mainly on undirected graphs, all results presented in the above theorems can be easily extended to directed graphs by using the concept of diverging forests as used in the traditional matrix forest theorem for directed graphs [1, 37]. Moreover, the extended results can also be proved by applying the same arguments based on the Leibniz formula as we used in the above theorems.

3.2 PPR computation by spanning forests

Recall that $\pi(s,s) = (L_\beta^{-1})_{ss}$ and $\pi(s,t) = (L_\beta^{-1})_{st}$. By Cramer's rule, we can obtain $\pi(s,s) = det(L_\beta^{(s)})/det(L_\beta)$ for the diagonal term and $\pi(s,t) = det(L_\beta^{(t,s)})/det(L_\beta)$ for the non-diagonal term. Then, by the matrix forest theorem developed in Section 3, we can compute the PPR values by the weights of spanning forests. Formally, we have the following results.

Theorem 3.4. *Given a graph $G = (V, E)$, a source node s and a decay factor α, the PPR value satisfies*

$$\pi(s,s) = \frac{\sum_{F \in \mathcal{F}_s} w(F) \prod_{u \in \rho(F)} \beta d_u}{\sum_{F \in \mathcal{F}} w(F) \prod_{u \in \rho(F)} \beta d_u},$$

where $\beta = \alpha/(1-\alpha)$, $\mathcal{F}$ is the set of all rooted spanning forests in G, $\mathcal{F}_s$ denotes the set of all rooted spanning forests in G having s as a root.

Theorem 3.5. *Given a graph $G = (V, E)$, a source node s and a decay factor α, the PPR value satisfies*

$$\pi(s,v) = \frac{\sum\limits_{F \in \mathcal{F}_{v,s}} w(F) \prod\limits_{u \in \rho(F)} \beta d_u}{\sum\limits_{F \in \mathcal{F}} w(F) \prod\limits_{u \in \rho(F)} \beta d_u},$$

for every node $v \in V$, where $\beta = \alpha/(1-\alpha)$, $\mathcal{F}$ is the set of all rooted spanning forests in G, $\mathcal{F}_{v,s}$ denotes the set of all rooted spanning forests in G in which s and v are in the same connected component and v is a root.

Based on the above two theorems, $\pi(s,t)$ equals the proportion of weights of spanning forests in which s is rooted in t to weights of all spanning forests. Clearly, the weights of all spanning forests form a *weight distribution*. Let $\Pr(s \text{ rooted in } t)$ be the probability that a node s rooted in a node t in a spanning forest randomly sampled from such a weight distribution. Then, we have the following results.

THEOREM 3.6. $\pi(s,t) = \Pr(s \textit{ rooted in } t)$.

Note that a spanning forest may contain several connected components (each tree is a connected component), which forms a partition of the nodes in the graph. Once a spanning forest F is generated, its corresponding partition ϕ is determined. Note that for a fixed partition ϕ of G, there may be many spanning forests in G that can produce the partition ϕ. Interestedly, we find that if a partition ϕ is given, the conditional probability that a node s is rooted in a node t (in a random spanning forest) conditioned on ϕ, denoted by $P(\text{s rooted in t}|\phi)$, can be explicitly determined as follows.

THEOREM 3.7. *Given a graph $G = (V, E)$, a spanning forest F and its partition $\phi = (V_1, \cdots, V_k)$. Suppose, without loss of generality, that s, t are two distinct vertices and t belongs to V_t. Let d_v be the weighted degree of node v. Then, the conditional probability that s is rooted in t conditioned on the partition ϕ equals $\frac{d_t}{\sum_{v \in V_t} d_v}$ if $s \in V_t$, equals 0 otherwise.*

Armed with Theorem 3.7, we can further obtain a different method to compute the PPR values. Let X_{st} be an indicator random variable that equals 1 if s and t are contained in the same component in a random spanning forest, equals 0 otherwise. Then, we have the following results.

THEOREM 3.8. $\pi(s,t) = E[\frac{d_t}{\sum_{u \in V_t} d_u} X_{st}]$.

4 SAMPLING SPANNING FORESTS

Note that by Theorem 3.6, we can estimate the PPR values via sampling spanning forests. Specifically, if we can sample spanning forests according to its weights, that is, $P(F) \propto w(F) \prod_{u \in \rho(F)} \beta d_u$, then an unbiased estimator of $\pi(s,t)$ can be easily derived. For example, suppose that we have drawn N random spanning forests. If n of which has a component such that s is rooted in t, then we can estimate $\pi(s,t)$ as $\frac{n}{N}$.

The remaining question is how can we sample spanning forests from such a weight distribution $P(F)$? To solve this problem, we propose a loop-erased α-random walk approach to sample random spanning forests based on the weight distribution $P(F)$. Our technique is a nontrivial extension of the classic Wilson algorithm for sampling spanning trees on graphs [48].

4.1 The loop-erased α-random walk

The loop-erased α-random walk does the same thing as the traditional α-random walk, but erasing all loops in the random walk trajectory. Below, we first discuss the concept of the traditional loop-erased random walk as introduced in [48].

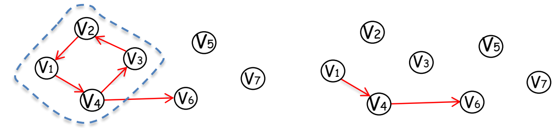


Figure 1: A random walk trajectory $\gamma = (v_1, v_4, v_3, v_2, v_1, v_4, v_6)$ and its loop-erased trajectory $LE(\gamma) = (v_1, v_4, v_6)$

Given a graph G and a random walk trajectory $\gamma = (v_1, \cdots, v_l)$ on G, we define the loop-erased trajectory as $LE(\gamma) = (v_{i_1}, \cdots, v_{i_j})$ by deleting all loops in γ. Formally, i_j is defined by the following inductive procedure: $i_1 = 1$ and $i_{j+1} = max\{i|v_i = v_{i_j}\} + 1$. Suppose that i_j is the max index by the above definition. Then, $LE(\gamma)$ contains i_j vertices and $i_j - 1$ directed edges. Loop-erased random walk is also self-avoiding; it will terminate when it hits the former trajectories. Initially, we set a node as a root and stop the first random walk when we hit the root. For example, in Fig. 1, suppose that v_6 is the root, and there is a random walk $\gamma = (v_1, v_4, v_3, v_2, v_1, v_4, v_6)$ stopping when it hits v_6. Then, its loop-erased trajectory is $LE(\gamma) = (v_1, v_4, v_6)$, by erasing the loop $(v_1, v_4, v_3, v_2, v_1)$. The process of erasing loops can be efficiently implemented by recording the next node in the random walk procedure. When a random walk stops, we retrace the trajectory, by starting from the first node, walking to the recorded next node until hitting the former trajectory. Note that the next node may be re-written many times, but after retracing it stores a unique next node in the final loop-erased trajectory.

The following results can be easily derived from [36].

THEOREM 4.1. *Let $\gamma = (v_{i_1}, \cdots, v_{i_j})$ be the final random walk trajectory after erasing loops. Denote by the former trajectory set Δ_0 and let $\Delta_k = \Delta_0 \cup \{v_1, \cdots, v_k\}$. We define $w(\gamma) = \prod_{i=1}^{k} w_{i_{k-1}, i_k}$. Then, the probability that γ is produced is*

$$\Pr(\Gamma = \gamma) = w(\gamma) \frac{det(L^{\Delta_k})}{det(L^{\Delta_0})},$$

where $L = D - A$ is the Laplacian matrix.

The loop-erased α-random walk is a nontrivial extension of the traditional loop-erased random walk. At each step, the loop-erased α-random walk has a probability α to stop. Suppose that it stops at a node u, then u is marked as a root. Note that for the loop-erased α-random walk, each loop-erased trajectory contains a root node when the α-random walk stops. We can derive the probability that a loop-erased trajectory $\gamma = (v_{i_1}, \cdots, v_{i_j})$ is generated when the loop-erased α-random walk stops at v_{i_j}.

THEOREM 4.2. *Let $\gamma = (v_{i_1}, \cdots, v_{i_j})$ be a loop-erased trajectory generated by a loop-erased α-random walk which stops at v_{i_j}. Denote by the former trajectory set Δ_0, and let $\Delta_k = \Delta_0 \cup \{v_1, \cdots, v_k\}$. We define $w(\gamma) = \prod_{i=1}^{k} w_{i_{k-1}, i_k}$. Then the probability that γ is produced is*

$$\Pr(\Gamma = \gamma) = \beta d_{v_j} \frac{det((L + \beta D)^{\Delta_k})}{det((L + \beta D)^{\Delta_0})} w(\gamma).$$

4.2 Algorithm for sampling spanning forests

Here we present our algorithm for sampling spanning forests. The intuition is that by iteratively performing loop-erased α-random

Algorithm 1: Loop-erased α-random walk sampling

Input: Graph $G = (V, E)$, a decay factor α
Output: $Root[u]$ for all $u \in V$

1 $InForest[u] \leftarrow false$, $Next[u] \leftarrow -1$, $Root[u] = -1$ for $u \in V$;
2 Fix an arbitrary ordering $(v_1, \cdots, v_n)$ of V;
3 **for** $i = 1 : n$ **do**
4 $\quad u = v_i$;
5 $\quad$ **while** $!InForest[u]$ **do**
6 $\quad\quad$ **if** $rand() < \alpha$ **then**
7 $\quad\quad\quad InForest[u] \leftarrow true$, $Root[u] \leftarrow u$;
8 $\quad\quad$ **else**
9 $\quad\quad\quad Next[u] \leftarrow RandomNeighbor(u)$;
10 $\quad\quad\quad u \leftarrow Next[u]$;
11 $\quad r \leftarrow Root[u]$, $u \leftarrow v_i$;
12 $\quad$ **while** $!InForest[u]$ **do**
13 $\quad\quad InForest[u] \leftarrow true$, $Root[u] \leftarrow r$;
14 $\quad\quad u \leftarrow Next[u]$;
15 **return** $Root[u]$ for all $u \in V$;

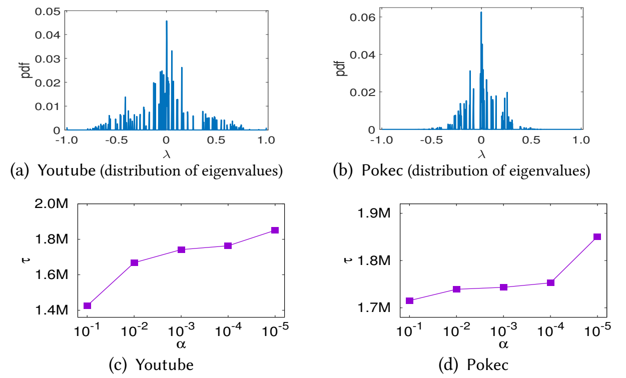
walks until all nodes in G are traveled, the trajectory exactly constructs a rooted spanning forest. We will see that the probability of each spanning forest F generated by our algorithm is exactly proportional to its weights, i.e., $\Pr(F) \propto w(F) \prod_{u \in \rho(F)} \beta d_u$.

The implementation details of the loop-erased α-random walk based sampling algorithm is outlined in Algorithm 1, which is an extension of the classic Wilson algorithm [48]. Specifically, Algorithm 1 starts by initializing an empty set F. We use a bool vector *InForest* to record whether node u has been added into F or not, a vector *Next* to record the next node in random walk step, and a vector *Root* to record the root of each node in the sampled spanning forest. The three vectors are initialized as $false$, -1 and -1 respectively (Line 1). Then, the loop-erased α-random walks are performed iteratively from a node u following a pre-fixed node ordering, and the resulting loop-erased trajectory is added into F until all nodes are covered (Line 3-14). Specifically, in each step, the random walk will stop in two cases, either (1) terminates at the current node with probability α (Line 6-7), or (2) terminates when hitting the former trajectories maintained by F (Line 5). If the loop-erased α-random walk stops with the first case, the vertex u is assigned as a root and added into F (Line 7). After the random walk stops, we retrace the walk by the *Next* array, and add the loop-erased trajectory into F (Line 13-14). The algorithm terminates when all nodes are processed (Line 3), and F is returned as a rooted spanning forest sampled from the weight distribution (Line 15). Note that since we only use the root information of the sampled spanning forest, it suffices to return the *Root* vector to represent a rooted spanning forest.

Theorem 4.3. *Let $\rho(F)$ be the root set of a rooted spanning forest F. Each F of G is sampled by Algorithm 1 with probability proportional to $w(F) \prod_{\rho(F)} \beta d_u$, that is*

$$\Pr(\gamma = F) = \prod_{u \in \rho(F)} \beta d_u \cdot \frac{w(F)}{det(L + \beta D)} \propto w(F) \prod_{u \in \rho(F)} \beta d_u.$$

Complexity anlaysis. The time complexity of Algorithm 1 can be derived by analyzing the number of operations on the *Next* array (Line 10). For the loop-erased random walk, when a loop is generated, the *Next* value of a node will be revised. The total number of random walk steps is mainly determined by the total number of revision of the *Next* array in Line 10, because the cost spent in the retrace process (Line 11-14) is dominated by the cost taken by the random walk process (Line 4-10). Note that the expected number of revision of the *Next* array in Line 10 equals the sum of the expected number of visits of each node in the entire α-loop erased walk process, which is denoted by τ. As a result, the time complexity of Algorithm 1 is $O(\tau)$. Below, we analyze the the expected number of visits for each node in the α-loop erased walk process.



(a) Youtube (distribution of eigenvalues) (b) Pokec (distribution of eigenvalues) (c) Youtube (d) Pokec

Figure 2: The distribution of eigenvalues of the matrix P ($P = D^{-1}A$) and the results of τ with varying α

Suppose that v_1 is the first node in the pre-fixed node ordering. Then, the expected number of visits to v_1 is given by $\frac{1}{\alpha}\pi(v_1, v_1)$, which can be derived by the power expansion $\sum_{k=0}^{\infty} (1-\alpha)^k {P^k}_{v_1, v_1}$ ($P = D^{-1}A$ is the probability transition matrix). In each step k, the expected probability mass that the α-random walk passes through v_1 is $(1-\alpha)^k P^k_{v_1, v_1}$, if the random walk does not stop. After that, the α-loop erased walk will no longer pass through v_1. Furthermore, the most important property of the loop-erased walk is that the node ordering is irrelevant to the final result [48]. That is, any node can be the first node and the final trajectory maintains the same distribution [48]. As a result, during the whole α-loop erased walk process, the expected number of visits to a node u is $\frac{1}{\alpha}\pi(u, u)$. By summing over all nodes, we can obtain $\tau = \frac{1}{\alpha}\sum_{u \in V} \pi(u, u)$. Note that since $\pi(u, u) = \sum_{k=0}^{\infty} \alpha(1-\alpha)^k P^k_{uu}$, we have $\tau = \sum_{u \in V}(\sum_{k=0}^{\infty}(1-\alpha)^k P^k_{uu})$. The following lemma show that τ is closely related to the spectrum of the probability transition matrix P ($P = D^{-1}A$).

Lemma 4.4. *Let $1 = \lambda_1 > \lambda_2 \geq \cdots \geq \lambda_n > -1$ be the probability eigenvalues of the transition matrix P ($P = D^{-1}A$). Then, we have*

$$\tau = \sum_{i=1}^{n} \frac{1}{1 - (1-\alpha)\lambda_i}. \tag{5}$$

Armed with Lemma 4.4, we can explain why τ is insensitive to α as follows. To estimate all nodes' personalized PageRank values, traditional α-random walk based methods need to simulate α-random walks from all nodes, which takes $O(\frac{n}{\alpha})$ per sample. For our loop-erased α-random walk, the total time complexity is $O(\tau)$, which is related to the spectrum of P. There are a large number of previous studies on the spectrum of transition probability matrix on real world graphs [18, 21, 22]. Note that τ is the sum of n terms, each term $\frac{1}{1-(1-\alpha)\lambda_i}$ falls in the range $(\frac{1}{2-\alpha}, \frac{1}{\alpha}]$, because $|\lambda_i| < 1$. Therefore, we have $\tau < \frac{n}{\alpha}$. Moreover, we can see that when λ_i is close to 1, $\frac{1}{1-(1-\alpha)\lambda_i}$ is close to $\frac{1}{\alpha}$. When λ_i is close to 0, $\frac{1}{1-(1-\alpha)\lambda_i}$ is close to 1, which is independent on α. As reported in [18], most of

Algorithm 2: The Forward Push Algorithm

```
Input: Graph G, source node s, decay factor α, threshold r_max
Output: Reserve q_s[v] and residual r[v] for all v ∈ V
1 for each u ∈ V do
2     r[u] = 0, q_s[u] = 0;
3 r[s] = 1;
4 while ∃u ∈ V such that r[u] ≥ d_u · r_max do
5     q_s[u] += αr[u];
6     for each z ∈ N(u) do
7         r[z] += (1 − α)w_zu r[u]/d_u;
8     r[u] = 0;
```

the eigenvalues of P in real-world graphs are concentrated around 0. That is to say, most terms in Eq. (5) are independent on α, thus τ is insensitive to α in real-world graphs. We also conduct experiments to compute the spectrum of P on 7 real world graphs (see Table 1) using the method proposed in [18]. The results on Youtube and Pokec are shown in Fig. 2(a-b). Similar results can also be observed on the other datasets. As can be seen, the distribution of eigenvalues are indeed concentrated around 0, implying that the time overhead of Algorithm 1 is insensitive w.r.t. α. We also study how the τ changes as α decreases. As shown in Fig. 2(c-d), τ increases smoothly w.r.t. α (note that in the horizontal axis, α decreases exponentially), which further confirms that our algorithm is insensitive to α.

5 SINGLE SOURCE PPR QUERY

In this section, we apply the proposed techniques to improve the performance of the existing algorithms for answering the single source PPR query. Below, we first briefly review the forward push algorithm and the state-of-the-art two-stage algorithms which combine deterministic forward push and α-random walk sampling. Then, we present our solutions by replacing the traditional α-random walk sampling with the proposed loop-erased α-random walk sampling.

5.1 Existing solutions

The forward push algorithm. The forward push algorithm is an efficient local method to compute single source PPR vector which was first proposed in [4]. As shown in Algorithm 2, the forward push algorithm maintains two vectors, including a reserve vector $q_s(v)$ and a residual vector $r(v)$ for all $v \in V$. Specifically, a push procedure (Line 4-8) is applied for each v with $r(v)$ larger than $d_v \cdot r_{max}$ until no such v exists. During the entire procedure, the following invariant is maintained for all $v \in V$ [4]:

$$\pi(s,v) = q_s(v) + \sum_{u \in V} r(u)\pi(u,v). \quad (6)$$

The algorithm runs in $O(\frac{1}{\alpha r_{max}})$ time. When r_{max} tends to 0, $q_s(v)$ converges to $\pi(s,v)$. However, a major limitation of the forward push algorithm is that there is no additive or relative error guarantee on $q_s(v)$ for a fixed r_{max}.

The α-random walk sampling algorithm. The single source PPR query can be efficiently estimated by simulating α-random walks. The algorithm generates a number of random walks from s, then counts the fraction of random walks that terminates at v as an estimation of $\pi(s,v)$. The major drawback of this algorithm is that to obtain a precise estimation, the number of samples can be very large. According to [7], to guarantee a relative error ϵ, it needs to generate $O(\frac{nlogn}{\epsilon^2})$ α-random walks. As the expected length of each α-random walk is $\frac{1}{\alpha}$, the algorithm takes $O(\frac{nlogn}{\alpha\epsilon^2})$ time.

Combining forward push and α-random walk sampling. To overcome the limitations of the forward push and the α-random walk sampling algorithms, Wang et al. [46] proposed a two-stage algorithm, called FORA, which combines a deterministic forward push stage and a Monte Carlo stage by sampling α-random walks. Let $W = \frac{nlogn}{\epsilon^2}$. To achieve a relative error ϵ, FORA first performs forward push with threshold r_{max}, and then runs $r(v)W$ random walks from each node v. The total α-random walks needed can be bounded by $nlogn \cdot r_{max}W$. Then, r_{max} is set to minimize the complexity. As a result, FORA reduces the time complexity of the α-random walk sampling algorithm from $O(\frac{nlogn}{\alpha\epsilon^2})$ to $O(\frac{nlogn}{\alpha\epsilon})$. Recently, ResAcc [32] and SPEEDPPR [49] improves FORA by accelerating the forward push algorithm. In particular, SPEEDPPR admits a time complexity $O(\frac{1}{\alpha}nlognlog\frac{1}{\epsilon}+\frac{nlogn}{\alpha})$ which is the state-of-the-art algorithm. However, for all the two-stage algorithms, no existing optimization technique has been done for the Monte Carlo stage.

5.2 Our solutions

In this subsection, we present our solutions based on the idea of replacing traditional α-random walks with loop-erased α-random walks in the state-of-the-art algorithms. We find that implementing such an idea is nontrivial, and there are two technical challenges needed to be tackled. Below, we first describe two challenges and the high-level ideas of our solutions to tackle these challenges.

Challenges and high-level ideas of our solutions. First, recall that in FORA, the number of α-random walks needed to sample from node u is $r(u)W$, which is different for each node. This is because the threshold used in the forward push algorithm for each node is different. The high-degree node may admit a very large residue, thus requires a large number of α-random walks. The total number of α-random walk in FORA can be bounded by $nr_{max}W$. However, in the context of sampling spanning forests using loop-erased α-random walks, the number of samples are the same for all nodes. Suppose that d_{max} is the largest degree over all nodes. Then, by applying the Chernoff bound, it requires $d_{max}r_{max}W$ loop-erased α-random walks, which makes the algorithm inefficient.

To circumvent this issue, we propose a new forward push algorithm called *balanced forward push*, which adapts the threshold for each node u from $d_u r_{max}$ to r_{max}. The detailed implementation of this algorithm can be found in our full version [31]. Although the balanced forward push algorithm only changes the threshold (compared to the traditional forward push algorithm), it is nontrivial to analyze its time complexity. Moreover, it is also very challenging to analyze the number of samples needed in our two-stage PPR computation algorithm when using such a balanced forward push as the *push stage*. We will tackle this by introducing an improved estimator together with carefully applying the Chernoff bound (see Theorem 5.3). Our result shows that it is sufficient to sample $r_{max}W$ random spanning forests without losing theoretical guarantee. Note that sampling a random spanning forest is roughly equivalent to draw n α-random walk samples. Since sampling a random spanning forest by loop-erased α-random walk is often much faster than sampling n α-random walks, we can achieve significantly speedup over FORA, as confirmed in our experiments.

Second, to estimate $\pi(s, t)$, a basic estimator only needs the information of the spanning forests in which s is rooted in t based on Theorem 3.6. Let X_i be a random variable that represents whether a node i is rooted in a target node t in a random spanning forest. It is easy to verify that random variables $X_1, \cdots, X_n$ are dependent, which violates the condition of applying Chernoff inequality to bound the sample size. To tackle this challenge, we propose an improved estimator based on Theorem 3.8. The key idea of the improved estimator is based on the so-called conditional Monte Carlo estimation technique [38], because our spanning forests sampling method can obtain the root probability conditioned on a fixed partition of the graph. By using the conditional probabilities, we can reduce the variance of the estimator based on the result of the total variance formula $Var[X] = Var[E[X|Y]] + E[Var[X|Y]]$, and $Var[X] > Var[E[X|Y]]$ since a variance is always non-negative. More intuitively, compared to the basic estimator, the improved estimator based on Theorem 3.8 can use much more additional information of a sampled spanning forest (i.e., the information of two nodes in the same connected component), instead of only using the root information as used in the basic estimator, thus can reduce the variance. Note that such a variance reduction trick can reduce the number of samples needed for a desired accuracy guarantee. Moreover, we will show that we are able to bound the sample size based on such a technique.

The proposed algorithm. Based on the above high-level ideas, we present our algorithms FORAL and FORALV in Algorithm 3, which corresponds to the algorithm with the basic estimator and the improved estimator respectively. The algorithm first invokes the *balanced forward push* to obtain the residual $r(s, u)$ and reserve $q_s[u]$ for all $u \in V$ (Lines 1-2). After that, the algorithm sets the parameters W and ω to guarantee the approximate accuracy (Lines 3-4). Then, ω random spanning forests are sampled independently by simulating loop-erased α-random walks to estimate the PPR values (Lines 5-14). Let F_i be the i-th sampled random spanning forest (Line 7). Then, the estimator is updated in two different ways, with or without applying the variance reduction technique. In particular, in FORAL (the algorithm with the basic estimator), a_v is computed as the sum over residuals on the subset of the connected component which v belongs to (Line 14). However, in FORALV (the algorithm with the improved estimator), a_v is computed by weighted averaging the residual in that subset according to the conditional probability (Line 11). Finally, the estimation $\hat{\pi}(s, v)$ is returned for each $v \in V$ as the query result (Line 15).

Note that in Line 1 of Algorithm 3, we can also use the improved forward push algorithm proposed in [49]. We refer to Algorithm 1 with the improved forward push algorithm as SPEEDL (with a basic estimator) and SPEEDLV (with an improved estimator) respectively. Below, we analyze the correctness and sample complexity of the proposed algorithms.

Analysis of the algorithm. First, we formally define the proposed estimators. Let $r(u)$ be the residual of u returned by forward push algorithm, V_v be the vertex set that is rooted in the same node as v. Then, to estimate $\sum_{u \in V} r(u)\pi(u, v)$ in Eq. (6), we can define two estimators $\bar{r}(v)$ and $\tilde{r}(v)$ for all $u \in V$, where $\bar{r}(v) \triangleq \sum_{u \in V_v} r(u)$ is a basic estimator and $\tilde{r}(v) \triangleq \frac{d_v \sum_{u \in V_v} r(u)}{\sum_{u \in V_v} d_u}$ is an improved estimator.

Algorithm 3: FORAL (FORALV)

Input: Graph $G = (V, E)$, source node s, decay factor α, push threshold r_{max}, relative error threshold ϵ, PPR value threshold μ
Output: Estimate PPR $\hat{\pi}(s, v)$ for all $v \in V$

```
1  Invoke balanced forward push with parameters G, s, α and r_max;
2  Let r(s, v_i), π̂(s, t) be the returned residue and reserve for all v_i ∈ V;
3  Let W = (2ε/3+2)·log(2/p_f) / (ε^2·μ);
4  Let ω = ⌈r_max · W⌉;
5  for i = 1 : ω do
6      Simulate loop-erased α-random walks on G;
7      Let F_i be the returned random spanning forest;
8      for each node t ∈ V do
9          Let V_v be subset of the partition Φ(F_i) which v belongs to;
10         if apply variance reduction then
11             a_v = d_v Σ_{u∈V_v} r(u) / Σ_{u∈V_v} d_u;
12         else
13             a_v = Σ_{u∈V_v} r(u);
14         π̂(s, v) = π̂(s, v) + a_v/ω;
15 return π̂(s, v) for all v ∈ V;
```

Let X_s be an indicator random variable that equals 1 if s is rooted in t in a spanning forest, equals 0 otherwise. Then, by Theorem 3.6, we have $E[X_s] = \pi(s, t)$. In other words, $\bar{r}(v)$ is an unbiased estimator of $\sum_{v \in V} r(v)\pi(v, t)$, thus the correctness of FORAL and SPEEDL can be guaranteed.

Let X_{st} be the co-occurrence random variable that equals 1 when s and t are in the same connected component of a spanning forest, equals 0 otherwise. By Theorem 3.8, we have $\pi(s, t) = E[\frac{d_t X_{st}}{\sum_{u \in V} d_v}]$. Let $Y = \sum_{v \in V} r(v) \frac{d_t X_{vt}}{\sum_{v \in V_s} d_t}$. Then, we can derive that $E[Y] = \sum_{v \in V} r(v)\pi(v, t)$ by the linearity of expectation. As a consequence, $\tilde{r}(v)$ is an unbiased estimator of $\sum_{v \in V} r(v)\pi(v, t)$, which guarantees the correctness of FORALV and SPEEDLV.

The following lemma shows that the improved estimator has a smaller variance compared to the basic estimator.

Lemma 5.1. *$Var[\tilde{r}(v)] \leq Var[\bar{r}(v)]$ for all $v \in V$.*

Note that the relative error of Algorithm 3 with the basic estimator is hard to bound due to the dependency of random variables X_s. However, the practical performance of our FORAL and SPEEDL algorithms are comparable to the state-of-the-art algorithms as confirmed in our experiments. Interestingly, unlike the basic estimator, we find that Algorithm 3 with the improved estimator can obtain a relative error guarantee. For our analysis, we need the following Chernoff bound [17].

Theorem 5.2. *(Chernoff bound) Let $X_i (1 \leq i \leq n)$ be independent random variables satisfying $X_i \leq E[X_i] + M$ for $1 \leq i \leq n$. Let $X = \frac{1}{n_r} \sum_{i=1} X_i$. Assume that $E[X]$ and $Var[X]$ be the expectation and variance of X. Then we have*

$$\Pr(|X - E[X]| \geq \lambda) \leq 2exp(-\frac{\lambda^2 n_r}{2Var[X] + 2M\lambda/3}).$$

By the Chernoff bound, we can derive the following theorem.

Theorem 5.3. *For any node t with $\pi(s, t) > \mu$, Algorithm 3 returns an approximate PPR value $\hat{\pi}(s, t)$ satisfying $|\pi(s, t) - \hat{\pi}(s, t)| \leq \epsilon d_t \pi(s, t)$ with probability at least $1 - p_f$.*

Similar to the time complexity analysis of the forward push algorithm [4], we can easily derive that the time complexity of our balanced forward push is $O(\frac{\bar{d}}{\alpha r_{max}})$, where $\bar{d}$ is the average degree. It can be further simplified as $O(\frac{\log n}{\alpha r_{max}})$ on scale-free graphs when $\bar{d} = \frac{\sum_{u \in V} d_u}{n} = \frac{2m}{n} = O(\log n)$.

Lemma 5.4. *Let $\bar{d}$ be the average degree. The time complexity of the balanced forward push can be bounded by $O(\frac{\bar{d}}{\alpha r_{max}})$.*

Based on Lemma 5.4, we can analyze the time complexity for all the proposed methods. In particular, the time costs of Algorithm 3 consist of two parts, the forward push stage and the Monte Carlo stage. For a fixed r_{max}, the time complexity of the deterministic forward push is bounded by $O(\frac{\log n}{\alpha r_{max}})$. The Monte Carlo stage samples $r_{max}W$ random spanning forests. The cost of sampling a spanning forest is τ; and the estimation process takes $O(n)$ in total, which is typically lower than τ. Therefore, the Monte Carlo stage takes $O(r_{max}W\tau)$ time. As a result, the total time complexity of Algorithm 3 is $O(\frac{\log n}{\alpha r_{max}} + r_{max}W\tau)$. This can be minimized by setting $r_{max} = \frac{\epsilon}{\sqrt{\alpha n \tau}}$, which results in an $O(\frac{1}{\epsilon}\sqrt{\frac{n \log n\tau}{\alpha}})$ complexity.

Similarly, for SPEEDL and SPEEDLV, the time costs include two parts: the time spent for forward push and the time taken for sampling spanning forests. By a similar analysis shown in [49], we can easily derive that the total time complexity of SPEEDL and SPEEDLV is $O(\frac{1}{\alpha} nlognlog\frac{1}{\epsilon} + logn\tau)$. As can be seen, the time complexity of our algorithms has a weak dependency on the parameter α, compared to the complexity of FORA [46] which is $O(\frac{nlogn}{\alpha\epsilon})$, and the complexity of SPEEDPPR [49] which is $O(\frac{1}{\alpha} nlognlog\frac{1}{\epsilon} + \frac{nlogn}{\alpha})$. Therefore, our algorithms can be much faster than the previous algorithms when α is small, which are also confirmed in our experiments.

5.3 Indexing spanning forests

Note that an optimization of FORA and SPEEDPPR is to pre-compute α-random walks, and then maintain the end-node for each α-random walk as an index. Such index-based methods are called FORA+ [46] and SPEEDPPR+ [49], respectively. To answer the single source PPR query, both FORA+ and SPEEDPPR+ can use the index to estimate PPR without simulating α-random walks online. For space overhead, FORA+ requires d_v/ϵ α-random walks for each node v. Thus, the total number of α-random walks is $\sum_{u \in V} d_u/\epsilon = 2m/\epsilon$. The index size of FORA+ can be further bounded by $O(\frac{n \log n}{\epsilon})$ with a relative error ϵ, given that $m = O(n \log n)$ on the scale free graphs. For SPEEDPPR+, it only requires d_v random walks for each v, thus its space overhead is $O(n \log n)$ [49].

Similar to FORA+ and SPEEDPPR+, we can also devise index-based variants of our online algorithms FORALV+ and SPEEDLV+. Specifically, we can first generate $O(\log n)$ random spanning forests. Note that similar to SPEEDPPR+, we can derive that $O(\log n)$ random spanning forests is sufficient to obtain a good estimation accuracy. Then, for each spanning forest, we maintain the root information for each node as the index. To implement the improved estimator in Algorithm 3 (Lines 10-11), we further maintains the total degree information in each connected component of a spanning forest. The total space overhead of our index is $O(n \log n)$. Note that to estimate PPR, sampling a spanning forest by loop-erased α-random walk is roughly equal to sampling n α-random walks

Algorithm 4: The Backward Push Algorithm

Input: Graph G, source node t, decay factor α, threshold r_{max}
Output: Reserve $q_t[v]$ and residual $r[v]$ for all $v \in V$

```
1 for each u ∈ V do
2     r[u] = 0, q_t[u] = 0
3 r[t] = 1;
4 while ∃u ∈ V such that r[u] ≥ r_max do
5     q_s[u] += αr[u];
6     for each z ∈ N(u) do
7         r[z] += (1 − α) w_uz r[u]/d_w;
8     r[u] = 0;
```

from each node. Because we can get n "samples of (i rooted in j)" for a spanning forest, while for an α-random walk we only get one sample, i.e., the end node of the random walk. Thus, the number of samples needed by our algorithm is around $1/n$ times FORA+ and SPEEDPPR+. Since sampling a spanning forest (τ) is much faster than sampling n α-random walks ($\frac{n}{\alpha}$) (especially for a small α), the index construction time of our algorithm is much lower than FORA+ and SPEEDPPR+, as confirmed in our experiments. As a result, compared to FORA+ and SPEEDPPR+, the key advantage of our index-based methods is that they can significantly save index-building time especially when α is small (e.g. $\alpha = 0.01$), which is also confirmed in our experiments.

6 SINGLE TARGET PPR QUERY

6.1 Existing solutions

Backward push. The backward push algorithm is an analogy of the forward push algorithm [3]. As shown in Algorithm 4, it also maintains two vectors reserve $q_t(v)$ and residual $r(v)$. Then, a slightly different push procedure is applied for each node with $r(v) > r_{max}$, until there is no node meeting $r(v) > r_{max}$ (Lines 4-8). The following invariant is maintained during the backward push process:

$$\pi(v,t) = q_t[v] + \sum_{u \in V} \pi(v,u) r(u). \tag{7}$$

Suppose that a push operation consumes time c_{push}, which is roughly $O(\log n)$ in an average case [3], the algorithm runs in $O(\frac{\pi(t) c_{push}}{\alpha r_{max}})$ time for a target node t. Unlike the forward push algorithm, the backward push algorithm can guarantee an additive error [3].

Randomized backward push. The randomized backward push algorithm introduces a sampling procedure into Algorithm 4 [43]. Specifically, in each push operation, a node sends residuals only to a small fraction of its randomly sampled neighbors, thus it can improve the efficiency. The time complexity of this algorithm is $(\frac{n\pi(t)}{\alpha\epsilon})$ for a relative error ϵ. The limitation of this algorithm is that it needs to take additional cost for sampling in each push operation. Moreover, its time complexity still depends on α.

6.2 The proposed algorithm

Here we develop two two-stage algorithms to answer the single target PPR query based on backward push and the proposed random forests sampling technique. Compared to the algorithms for processing single source PPR query, there are two differences in designing an algorithm for answering the single target PPR query.

First, the time complexity of the backward push algorithm depends on $\pi(t)$ of the target node t and it varies heavily over all nodes. For nodes with small $\pi(t)$ (the low-degree nodes often have a small $\pi(t)$), the backward push procedure terminates very fast. Consequently, there is no need to apply any sampling technique to speed up the algorithm for those nodes. For nodes with large $\pi(t)$ (the high-degree nodes often have a large $\pi(t)$), the backward push procedure often takes a long time especially when α is small. Therefore, in this case, we can devise two-stage algorithms based on backward push and sampling random spanning forests. Second, unlike the forward push algorithm used in the single source PPR query problem, an additive error r_{max} can be guaranteed by the backward push algorithm. To achieve a relative error, we can set r_{max} as $\frac{\epsilon}{n}$, resulting in that the time complexity of the backward push is $O(\frac{n\pi(t)c_{push}}{\alpha\epsilon})$.

Implementation details. The pseudo code of our algorithms is shown in Algorithm 5. Algorithm 5 includes two stages including deterministic backward push and sampling random spanning forests. First, Algorithm 5 performs backward push to compute the residual and reserve for each node (Lines 1-2). Then, Algorithm 5 simulates the loop-erased α-random walk technique to sample random spanning forests (Lines 5-14). Similar to Algorithm 3, Algorithm 5 can also use the basic estimating technique (Lines 10-11) and the improved estimating technique (Lines 12-13) to achieve unbiased estimations of PPR values. For convenience, Algorithm 5 with the basic estimator and the improved estimator are referred to as BACKL and BACKLV respectively.

Analysis of the algorithm. For a node v, let X_u be an indicator random variable that equals 1 if v is rooted in u, equals 0 otherwise. Let X_{st} be the co-occurrence random variable that equals 1 when s and t are in the same connected component of a spanning forest, equals 0 otherwise. $Y_1 = \sum_{u\in V} r(u)X_u$ for BACKL and $Y_2 = \sum_{u\in V} r(u)X_{uv}\frac{d_u}{\sum_{k\in V_v} d_k}$ for BACKLV. Similar to our previous analysis for the single source PPR query, we can easily derive that $E[Y_1] = E[Y_2] = \sum_{u\in V}\pi(v,u)r(u)$. Therefore, the variable a_v used in BACKL and BACKLV is an unbiased estimator of $\sum_{u\in V}\pi(v,u)r(u)$. Below, we apply the Chernoff bound to analyze the relative error guarantee of our algorithms.

Theorem 6.1. *For any node v with $\pi(v,t) > \mu$, both* BACKL *and* BACKLV *return an approximate PPR value $\hat{\pi}(v,t)$ satisfying $|\pi(v,t) - \hat{\pi}(v,t)| \le \epsilon\pi(v,t)$ with probability at least $1 - p_f$.*

Note that although both BACKL and BACKLV can guarantee the same relative error as shown in Theorem 6.1, BACKLV has a smaller variance based on the improved estimating technique. Therefore, we focus mainly on the BACKLV algorithm in the remaining of this paper. The time complexity of BACKLV for a target node t consists of two parts. In the backward push stage, BACKLV takes $O(\frac{c_{push}\pi(t)}{\alpha r_{max}})$ time, while in the Monte Carlo stage, BACKLV needs to sampling $r_{max}W$ spanning forests which consumes $O(r_{max}W\tau)$ time in total. Thus, the time complexity of BACKLV is $O(\frac{c_{push}\pi(t)}{\alpha r_{max}} + r_{max}W\tau)$, which can be minimized to $O(\frac{1}{\epsilon}\sqrt{\frac{c_{push}n\log n\tau}{\alpha}})$ by setting an appropriate r_{max}.

Algorithm 5: BACKL (BACKLV)

```
   Input: Graph G = (V, E), target node t, decay factor α, threshold r_max, relative error
          threshold ε, PPR value threshold μ
   Output: The estimated PPR π̂(v, t) for all v ∈ V
1  Invoke backward push with input parameter G, t, α and r_max;
2  Let r(v), π̂(v, t) be the returned residue and reserve for all v ∈ V;
3  Let W = ((2ε/3+2)·log(2/p_f)) / (ε²·μ);
4  Let ω = ⌈r_max · W⌉;
5  for i = 1 : ω do
6      Simulate a loop-erased walk on G;
7      Let F_i be the returned random spanning forests;
8      for each node v ∈ V do
9          Let V_t be subset of the partition Φ(F_i) which t belongs to and the root is u;
10         if apply variance reduction then
11             a_v = (Σ_{u∈V_t} r(u)d_u) / (Σ_{u∈V_t} d_u);
12         else
13             a_v = r(u);
14         π̂(v, t) = π̂(v, t) + a_v/ω;
15 return π̂(v, t) for all v ∈ V;
```

Table 1: Datasets

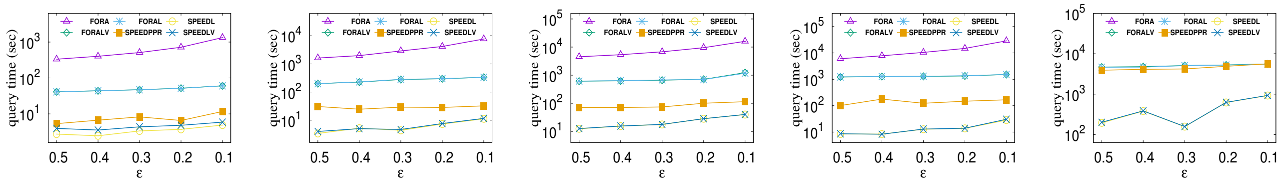
Type	Dataset	n	m	$\bar{d}$
unweighted graphs	Youtube	1,134,890	2,987,624	5.27
	Pokec	1,632,803	22,301,964	27.32
	LiveJournal	4,846,609	42,851,237	17.68
	Orkut	3,072,441	117,185,083	76.28
	Twitter	41,652,230	1,202,513,046	57.74
weighted graphs	DBLP	1,824,701	8,344,615	32.32
	StackOverflow	2,584,164	28,142,395	37.02

7 EXPERIMENTS

7.1 Experimental setup

Datasets and query sets. We use 5 real-life datasets including Youtube, Pokec, LiveJournal, Orkut and Twitter, which are widely used in previous studies [32, 43, 46, 49]. We also include 2 real-life general weighted graphs DBLP and StackOverflow. Specifically, DBLP is a collaboration network where each node represents an author, each edge represents collaboration relationship and the edge weight is the number of co-authored papers. StackOverflow is a user interaction network from the StackExchange site. Each node represents a user, each edge denotes an interaction relationship and the edge weight is the number of user interactions. The detailed statistics of these datasets are summarized in Table 1. All these datasets can be obtained from [30]. For the single source query problem, as used in [46], we perform queries using 50 source nodes generated uniformly at random for all competitors and take the average query time as the final result. For the single target query, the query time is highly dependent on the chosen target node. For the low-degree nodes, it terminates fast by only applying backward push, while for the high-degree nodes, it spends a long time for the backward push such that sampling technique is necessary in this case. We perform queries on 50 target nodes generated uniformly at random from the top 10% highest degree nodes and again take the average query time as the final result.

Different algorithms. For single source PPR query, we compare our algorithms with the state-of-the-art algorithms FORA [46] and SPEEDPPR [49]. We do not include other previous algorithms in the experiments because all of them are outperformed by SPEEDPPR [49]. For FORA and SPEEDPPR, we use their original implementations in [46] and [49] respectively. For our solutions, we implement 4 different algorithms which are FORAL, FORALV, SPEEDL, and



(a) Youtube, α=0.01 (b) Pokec, α=0.01 (c) LiveJournal, α=0.01 (d) Orkut, α=0.01 (e) Twitter, α=0.01

Figure 3: Runtime of different algorithms for answering the single source query on unweighted graphs

SPEEDLV. FORAL and FORALV denote Algorithm 3 with the basic estimator and the improved estimator respectively. Both of them use the balanced forward push in Line 1 of Algorithm 3. SPEEDL (SPEEDLV) is an improved algorithm for FORAL (FORALV) which is equipped with an improved forward push algorithm [49].

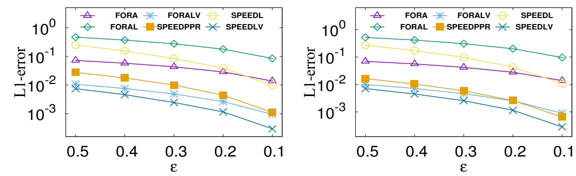
For single target PPR query, we compare our two-stage algorithm with the state-of-the-art algorithms BACK and RBACK. BACK is the backward push algorithm which can guarantee an additive error ϵ. To achieve a relative error, we only need to set the threshold as ϵ/n for BACK. RBACK [43] is the randomized backward push which can prune nodes with small residues in each push operation. However, RBACK needs to take additional time to preform random sampling. We implement BACK and RBACK by ourselves, as no available implementation of these algorithms are provided. For our algorithm, we implement BACKLV which is Algorithm 5 with an improved estimator. Note that since BACKL is clearly worse than BACKLV, we did not implement BACKL in our experiments.

Parameters. Since we focus mainly on small α, we set the parameter $\alpha = 0.01$ in our experiments. Moreover, many existing PPR-based graph mining algorithms often work very well when $\alpha = 0.01$ [13, 41, 50]. We will study the performance of our algorithms with varying α and also with very small α (e.g., $\alpha = 10^{-5}$). In addition, since the parameter α is typically set to 0.2 in most existing algorithms [43, 46, 49], we also consider this parameter setting for a fair comparison with those baseline algorithms, and the results can be found in our full version [31]. For the approximate single source/target query, there is a parameter ϵ which controls the relative error. We set ϵ as 0.5 by default; and vary ϵ from 0.1 to 0.5.

7.2 Single source query

In this experiment, we compare the performance of different algorithms for answering the single source query. The results on five unweighted datasets are reported in Fig. 3. For a better understanding of these results, we first focus on comparing the performance of FORA, FORAL, and FORALV. We observe that compared to FORA, both FORAL and FORALV obtain around 100× speedups on all datasets (with $\alpha = 0.01$). FORALV spends slightly more time than FORAL because it includes an additional computational cost of the sum over partitions. For large datasets, for example on Twitter, FORA runs out of 24 hours while both FORAL and FORALV take only thousands of seconds. In general, the runtime of all algorithms increase with decreasing ϵ, because all algorithms take more time to achieve a small error.

Second, we compare the runtime of SPEEDPPR, SPEEDL and SPEEDLV. Note that SPEEDPPR applies an optimized version of forward push which is often more efficient, but it cannot apply the theoretical bound to balance the time spent in the Monte Carlo phase and the forward push phase (as FORA does). Alternatively,



(a) LiveJournal, α = 0.01 (b) Orkut, α = 0.01

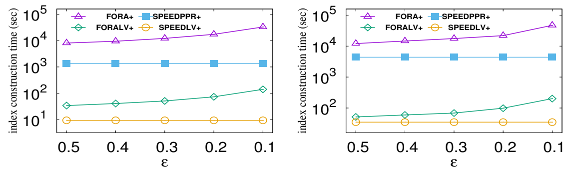
Figure 4: Comparison of the $L1$-error of different algorithms

SPEEDPPR balance the time spent in the two phases based on an estimation of the α-random walk time. SPEEDL and SPEEDLV also adopt a similar idea to balance the time. From Fig. 3, we can see that SPEEDL is the fastest algorithm among all the methods. On the largest dataset Twitter, both SPEEDL and SPEEDLV are an order of magnitude faster than SPEEDPPR. These results confirm our theoretical analysis that by replacing traditional α-random walks with loop-erased α-random walks can achieve a considerable speedup (especially for the cases of small α values).

To further evaluate the effectiveness of our loop-erased α-random walk based algorithms, we compare the L_1 errors of different algorithms. Here the L_1 error is deified as $\sum_{v \in V} |\tilde{\pi}(s,v) - \pi(s,v)|$ for all $v \in V$, where $\tilde{\pi}(s,v)$ is the personalized PageRank value estimated by different algorithms and $\pi(s,v)$ is the exact personalized PageRank value. Note that since both FORA and SPEEDPPR achieve the same relative error bounds, we focus mainly on comparing FORA, FORAL, and FORALV to see whether the loop-erased α-random walk based algorithms can also achieve a good estimating accuracy. Similar results can also be obtained by comparing SPEEDPPR, SPEEDL, SPEEDLV. Fig. 4 shows the results on LiveJournal and Orkut. The results on the other datasets are consistent. As can be seen, FORALV achieves the smallest L_1 errors, followed by FORA and FORAL. Note that FORAL is worse than FORA, because the the random variables used in FORAL is dependent. However, after using our variance reduction technique, the accuracy can be significantly improved. SPEEDPPR, SPEEDL, and SPEEDLV follow a similar trend; their $L1$-errors are slightly smaller than FORA, FORAL, and FORALV respectively, because they conduct more deterministic computing. These results suggest that our loop-erased α-random walk based algorithms can achieve a very good estimating accuracy.

7.3 Index-based method for single source query

In this experiment, we evaluate the performance of different index-based algorithms. The index-based variants for FORA and SPEEDPPR are denoted by FORA+ [46] and SPEEDPPR+ [49] respectively. We implement two index-based variants for our algorithms FORALV and SPEEDLV, denoted by FORALV+ and SPEEDLV+ respectively, because FORALV (SPEEDLV) is shown to be better than FORAL (SPEEDL).



(a) LiveJournal, $\alpha = 0.01$ (b) Orkut, $\alpha = 0.01$

Figure 5: Comparison of index construction time

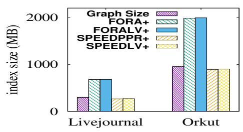
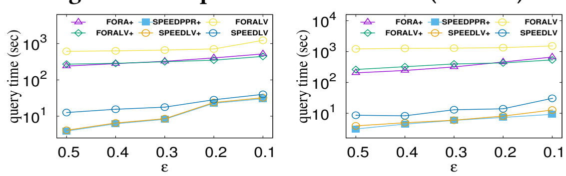


Figure 6: Comparison of index size ($\alpha = 0.01$)



(a) LiveJournal, $\alpha = 0.01$ (b) Orkut, $\alpha = 0.01$

Figure 7: Runtime of different index-based algorithms

We first compare the index construction time and index size of different algorithms. Note that both FORA+ and SPEEDPPR+ determine the index size based on theoretical results [46, 49]. FORA+ maintains $O(n \log n/\epsilon)$ α-random walks [46], while SPEEDPPR+ stores around $O(n \log n)$ α-random walks [49]. Since the performance of sampling n α-random walks (from n nodes) is similar to that of sampling a spanning forest, FORALV and SPEEDLV maintain $O(\log n/\epsilon)$ and $O(\log n)$ spanning forests respectively. Fig. 5 shows the index construction time on LiveJournal and Orkut. The results on other datasets are consistent. As can be seen, SPEEDLV+ achieves the lowest index construction time under all parameter settings, followed by FORALV+, SPEEDPPR+, and FORA+. Moreover, we can see that SPEEDLV+ is around an order of magnitude faster than SPEEDPPR+. These results demonstrate that our index-based algorithms are much more efficient than the state-of-the-art algorithms to construct the index, which also confirm our analysis in Section 5.3. Fig. 6 reports the index size of different algorithms on LiveJournal and Orkut. From Fig. 6, we can see that our index-based algorithms can achieve similar index size as the state-of-the-art algorithms. This is because for a spanning forest sample, we need to store the root for each node, while for a random walk sample, we only need to store the end node. Thus, although the number of samples needed by FORALV+ (SPEEDLV+) is around $1/n$ times FORA+ (SPEEDPPR+), the total space costs of them are nearly the same. Moreover, the space usages of all the index-based algorithms are comparable w.r.t. the graph size. These results further confirm that our index-based algorithms are space-efficient.

Second, we evaluate the query processing time of different index-based algorithms. The results are shown in Fig. 7. As can be seen, FORALV+ and SPEEDLV+ can achieve similar performance as FORA+ and SPEEDPPR+, respectively. Moreover, we also add the online algorithms FORALV and SPEEDLV for comparison. We can observe that all the index-based algorithms are faster than their online versions. Note that FORALV+ and SPEEDLV+ are slightly slower than FORA+ and SPEEDPPR+ respectively, because our algorithms takes additional cost to sum the results over the partitions of random spanning forests.

In summary, our index-based algorithms can achieve similar query processing time and similar index size over the state-of-the-art index-based algorithms. However, our index can be constructed within much lower time than those of the state-of-the-art algorithms.

7.4 Single target query

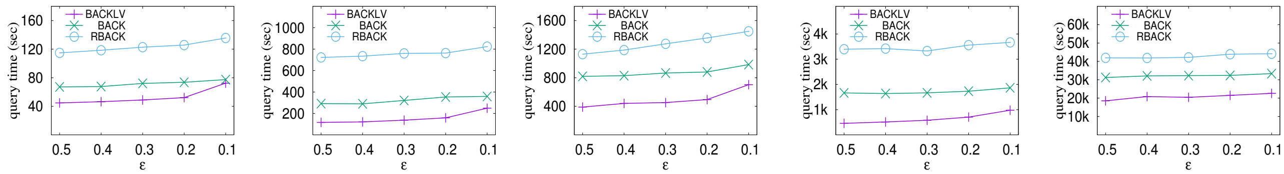
In this experiment, we compare the performance of BACK, RBACK, and BACKLV for answering the single target query. Fig. 8 shows the runtime of these three algorithms on five datasets. As shown in Fig. 8, BACKLV is significantly faster than BACK and RBACK (with $\alpha = 0.01$). In general, BACKLV can achieve $1\times \sim 3\times$ speedups over BACK on all datsets under most parameter settings. We also observe that RBACK is worse than BACK. The reason could be that (1) RBACK needs to use additional computational cost for sampling, and (2) to achieve a high precision, RBACK needs to set a small sampling threshold so that its performance is similar to that of the power method, which is often worse than the backward push algorithm. These results indicate that for the small α case, our loop-erased α-random walk based technique can also be useful for processing the single target query.

7.5 Results on real-life weighted graphs

In all previous experiments, we only consider unweighted graphs (a special case of weighted graph with all edge weights equaling 1) for a fair comparison with the state-of-the-art algorithms. In this experiment, we study the performance of different algorithms on general weighted graphs. To this end, we re-implement all the baseline methods as the available implementations in [43, 46, 49] cannot support general weighted graphs. The results of single source experiments on DBLP and StackOverflow are shown in Fig.9 and Fig.10. Similar results can also be observed on the other datasets. In general, the results on weighted graphs are consistent with our previous results on unweighted graphs. FORAL and FORALV (SPEEDL and SPEEDLV) have significantly less query time than FORA (SPEEDPPR). Our best algorithm SPEEDLV is at least one order of magnitude faster than the state-of-the-art algorithm (SPEEDPPR). The comparison of empirical error is also similar to that on unweighted graphs. Our SPEEDLV is clearly the winner with all parameter settings, and it is much more accurate than SPEEDPPR. Note that although there is a d_t term in the error bound of FORALV and SPEEDLV, the practical error performance is significantly better than that of FORA and SPEEDPPR as shown in Fig.10. The results of single target experiments are depicted in Fig.11. When $\alpha = 0.01$, BACKLV achieves a $2\times$ speed-up on both datasets, which is consistent with the previous results on unweighted graphs.

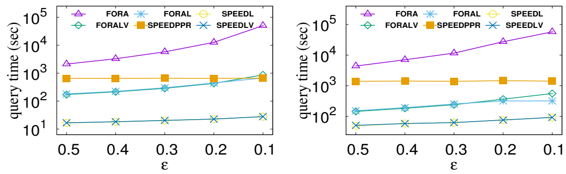
7.6 Results with various query node distributions

Here we study the effect of query node distributions. To this end, we consider three different node distributions to study how the degree of query node affects the query time. Specifically, we independently sample nodes uniformly from the whole node set, the top 10% high-degree node set and the top 10% low-degree node set respectively.



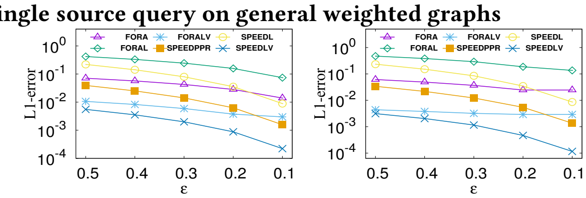
(a) Youtube, α=0.01 (b) Pokec, α=0.01 (c) LiveJournal, α=0.01 (d) Orkut, α=0.01 (e) Twitter, α=0.01

Figure 8: Runtime of different algorithms for answering the single target query on unweighted graphs



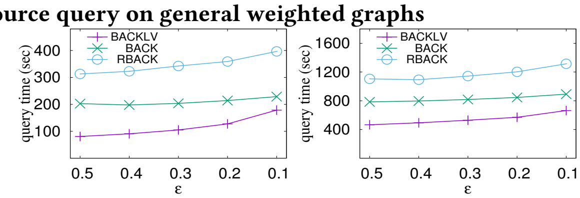
(a) DBLP, $\alpha = 0.01$ (b) StackOverflow, $\alpha = 0.01$

Figure 9: Runtime of different algorithms for answering the single source query on general weighted graphs



(a) DBLP, $\alpha = 0.01$ (b) StackOverflow, $\alpha = 0.01$

Figure 10: Comparison of $L1$-error for answering the single source query on general weighted graphs



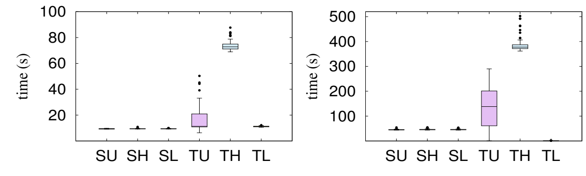
(a) DBLP, $\alpha = 0.01$ (b) StackOverflow, $\alpha = 0.01$

Figure 11: Runtime of different algorithms for answering the single target query on general weighted graphs

In this experiment, we choose the best methods, SPEEDLV and BACKLV, as two representative methods for single source query and single target query respectively; the results for other methods are consistent. To see the distribution of query time, we conduct experiments on 1000 nodes and use box-plot to show the results. The results are depicted in Fig.12. As can be seen, the query time of single source algorithms always have a very small variance. However, the query time of the single target algorithms are highly dependent on the node degree; and the query time of low-degree nodes is significantly lower than that of the high-degree nodes. For example, on Pokec, it takes around 400s for high-degree nodes, while all low-degree nodes take less than 1s to perform a query. These results indicate that for single target query, we only need to perform sampling to improve the query efficiency for high-degree query nodes, while for low-degree nodes, there is no need to do sampling.

7.7 Results with very small α

As discussed before, our algorithms are faster than existing methods especially when α is small. Previously, we only consider the case when α is relatively small ($\alpha = 0.01$). In this experiment, we study the case when α is very small. Note that if α tends to zero, the single

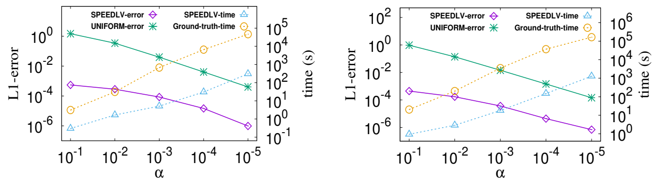


(a) Youtube, $\alpha = 0.01$ (b) Pokec, $\alpha = 0.01$

Figure 12: Query time distribution of single source query (SPEEDLV) and single target query (BACKLV). SU denotes single source query, choosing nodes uniformly. SH (SL) denotes single source query, selecting high (low) degree nodes uniformly. TU denotes single target query, picking nodes uniformly. TH (TL) denotes single target query, choosing high (low) degree nodes uniformly.

source PPR vector tends to a degree-weighted uniform distribution $\pi(s,u) = \frac{d_u}{2m}$ and the single target PPR vector tends to a constant distribution $\pi(u,t) = \frac{d_t}{2m}$ for all $u \in V$. Therefore, the degree-weighted uniform distribution vector is a very simple baseline for computing single source PPR vector when α is very small. Note that although the PPR vector is very close to a degree-weighted uniform distribution when α is very small, it can still provide more useful information than such a degree-weighted uniform distribution for node ranking and clustering due to the subtly difference between them [50]. For example, when we consider the degree normalized vector $\frac{\pi(s,u)}{d_u}$, the simple baseline will be degraded as a constant vector which is indistinguishable for all nodes ($1/2m$ for all nodes). However, as shown in [50], the degree-normalized PPR vector can still produce effective rankings and clusterings even when $\alpha = 10^{-6}$.

We vary α from 10^{-1} to 10^{-5}, and use the state-of-the-art deterministic method in [49] to compute the *ground-truth* of the single source PPR vector to an $L1$-error bound 10^{-9}. After that, we calculate the $L1$-error between SPEEDLV and the ground-truth PPR vector, and also compute the $L1$-error for the simple baseline. We randomly sample 50 nodes uniformly and take the average value the final result. The results on Youtube and Pokec are shown in Fig. 13. Similar results can also be observed on the other datasets. As can be seen, the $L1$-error of SPEEDLV is at least two orders of magnitude lower than that of the baseline method with varying α. These results indicate that even for a very small α, our algorithm can still produce much more accurate results than the baseline method. In addition, we can see that the $L1$-errors of both SPEEDLV and baseline decrease as α decreases. The reason could be that the results of SPEEDLV, the baseline method, and the ground truth PPR converge to the degree-weighted uniform distribution when α is very small, and thereby the $L1$-errors will be small. Fig. 13 also shows the time overheads for computing the ground truth and the time consumption by our SPEEDLV algorithm. We can see that the time costs by our algorithm are much lower than the time overheads



(a) Youtube (b) Pokec

Figure 13: Results with very small α (solid lines denote the L_1-error, and dashed lines represent the runtime)

for computing the ground truth. For a very small α ($\alpha \leq 10^{-4}$), SPEEDLV is at least two orders of magnitude faster than the ground truth computation algorithm [49]. These results indicate that our SPEEDLV algorithm can achieve a very good trade-off between accuracy and runtime.

8 RELATED WORK

PageRank computation. Methods for computing personalized Page-Rank can be divided into two categories: deterministic algorithms and randomized approximate algorithms. For deterministic methods, there are many studies that focus on matrix-based power methods [25, 53]. Based on the power method, many different optimization techniques were proposed. [19, 20] applied the Cheyshev polynomials to accelerate the convergence rate. BEAR [40] preprocessed the adjacency matrix so that there is a large and easy-to-invert submatrix and also pre-computed several submatrix required to form an index. BePI [29] improved BEAR by using the power method instead of matrix inversion. TPA [52] was also an index-based iterative method which used PageRank value to approximate the nodes that are far from the source node. [35] developed a core-tree decomposition technique to further improve the efficiency of the power method. Also, there are a large number of local methods for computing personalized PageRank, notable examples including the forward push method [4, 10] and the backward push method [3, 28, 34]. Although much progress has been made, deterministic methods are still slow for high-precision personalized PageRank computation.

For approximate methods, most of them are based on Monte Carlo simulation [7]. The idea of combining Monte Carlo and deterministic push method was first introduced in [33]. Much work follows this idea to improve different types of personalized PageRank queries. [44, 46] utilized the two-stage framework, which combines Monte Carlo and deterministic push, to answer the single source query. [32] and [49] further improved the single source query algorithm. [45] answered several new queries which aims to find heavy hitters in a graph based on the two-stage framework. However, in the Monte Carlo stage, all of the previous studies just simply simulate random walks. Unlike these studies, we propose an alternative method based on sampling of spanning forests which is shown to be more efficient than the random walk based sampling methods. Additionally, there also exist a number of algorithms to answer the top-k personalized PageRank query, which are also based on matrix operations [23, 25, 29], local methods [24, 27] and Monte Carlo techniques [8]. Specifically, matrix-based methods are based on the power method with a given absolute error bound ϵ_a; local methods conduct a local search from the source node while maintaining lower and upper bounds, and stops the search when the top-k results can be obtained by the lower and upper bounds; Monte Carlo techniques, including BiPPR [33], HubPPR [44] and FORA [46] are used for approximating the top-k PPR queries, which ensure a relative error ϵ_r for any PPR value larger than $1/n$, with probability at least $1 - 1/n$. TopPPR [47] is the state-of-the-art algorithm which combines forward push, backward push and Monte Carlo together to answer the top-k query.

Matrix forest theorem and spanning forest sampling. The Kirchhoff matrix tree theorem is perhaps the most classic result in spectral graph theory. Such a theorem has been generalized to spanning forest in early years [1, 11, 12, 39]. Most previous studies on the matrix forest theorem are based on the matrix $L + qI$ where q is a constant [1]. Unlike the previous studies, we establish a new PageRank matrix forest theorem based on the β-Laplacian matrix ($L_\beta = (\beta D)^{-1}(L + \beta D)$). We note that Chung and Zhao also introduced a PageRank matrix forest theorem on undirected graphs [14, 15]. Their results are mainly based on the classic Cauchy-Binet formula which are hard to extend to directed graphs. Moreover, their matrix forest theorem is based on the lazy random walk model, instead of the β-Laplacian matrix.

The algorithms for sampling spanning trees also have been heavily investigated [2, 26, 48]. The most well-known algorithms include (1) the Aldous-Broder algorithm [2] which simulates a random walk until the whole graph is covered; and (2) the Wilson algorithm [48] which simulates loop-erased random walks. Note that the concept of loop-erased random walk was also studied from the probability point of view [5, 6, 36]; and it was applied to generate spanning forests [5, 6] with an extended Wilson algorithm. Such an extended Wilson algorithm was also used for graph signal processing applications. [9, 38]. Unlike these work, we develop a loop-erased α-random walk algorithm to sample spanning forests for personalized PageRank computation.

9 CONCLUSION

In this work, we develop several novel personalized PageRank matrix-forest theorems which connects the personalized PageRank values to the weights of spanning forests. Based on this connection, we propose a new personalized PageRank computation algorithm that samples spanning forests via simulating loop-erased α-random walks on a graph. Compared to the previous algorithms, the proposed algorithm is shown to be more robust w.r.t. the parameter α. This enable us to improve the efficiency of the state-of-the-art algorithms when α is small. Specifically, by using our technique, we can significantly improve the efficiency of the state-of-the-art algorithms for answering two types of personalized PageRank queries, including single source and single target queries. Extensive experiments on 5 large real-life graphs demonstrate the efficiency of the proposed algorithms.

ACKNOWLEDGMENTS

This work was partially supported by (i) National Key Research and Development Program of China 2020AAA0108503, (ii) NSFC Grants 62072034 and U1809206. Rong-Hua Li is the corresponding author of this paper.

REFERENCES

[1] Rafig Agaev and Pavel Chebotarev. 2006. Spanning Forests of a Digraph and Their Applications. *CoRR* abs/math/0602061 (2006). arXiv:math/0602061

[2] David J. Aldous. 1990. The Random Walk Construction of Uniform Spanning Trees and Uniform Labelled Trees. *SIAM J. Discret. Math.* 3, 4 (1990), 450–465.

[3] Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. 2007. Local Computation of PageRank Contributions. In *WAW*. 150–165.

[4] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. 2006. Local Graph Partitioning using PageRank Vectors. In *FOCS*. 475–486.

[5] Luca Avena, Fabienne Castell, Alexandre Gaudillière, and Clothilde Mélot. 2018. Random forests and networks analysis. *Journal of Statistical Physics* 173, 3 (2018), 985–1027.

[6] Luca Avena and Alexandre Gaudillière. 2018. Two applications of random spanning forests. *Journal of Theoretical Probability* 31, 4 (2018), 1975–2004.

[7] Konstantin Avrachenkov, Nelly Litvak, Danil Nemirovsky, and Natalia Osipova. 2007. Monte Carlo Methods in PageRank Computation: When One Iteration is Sufficient. *SIAM J. Numer. Anal.* 45, 2 (2007), 890–904.

[8] Konstantin Avrachenkov, Nelly Litvak, Danil Nemirovsky, Elena Smirnova, and Marina Sokol. 2011. Quick Detection of Top-k Personalized PageRank Lists. In *WAW*. Springer, 50–61.

[9] Simon Barthelmé, Nicolas Tremblay, Alexandre Gaudilliere, Luca Avena, and Pierre-Olivier Amblard. 2019. Estimating the inverse trace using random forests on graphs. *arXiv preprint arXiv:1905.02086* (2019).

[10] Pavel Berkhin. 2006. Bookmark-Coloring Approach to Personalized PageRank Computing. *Internet Math.* 3, 1 (2006), 41–62.

[11] Seth Chaiken. 1982. A combinatorial proof of the all minors matrix tree theorem. *SIAM Journal on Algebraic Discrete Methods* 3, 3 (1982), 319–329.

[12] Pavel Chebotarev. 2008. Spanning forests and the golden ratio. *Discret. Appl. Math.* 156, 5 (2008), 813–821.

[13] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable Graph Neural Networks via Bidirectional Propagation. In *NIPS*.

[14] Fan Chung. 2010. PageRank as a discrete Green's function. *Geometry and Analysis I ALM* 17 (2010), 285–302.

[15] Fan Chung and Wenbo Zhao. 2010. PageRank and random walks on graphs. In *Fete of combinatorics and computer science*. Springer, 43–62.

[16] Fan R. K. Chung. 1996. *Spectral Graph Theory*. Vol. 92. American Mathematical Society.

[17] Fan R. K. Chung and Lincoln Lu. 2006. Survey: Concentration Inequalities and Martingale Inequalities: A Survey. *Internet Math.* 3, 1 (2006), 79–127.

[18] David Cohen-Steiner, Weihao Kong, Christian Sohler, and Gregory Valiant. 2018. Approximating the Spectrum of a Graph. In *KDD*.

[19] Mustafa Coskun, Ananth Grama, and Mehmet Koyutürk. 2016. Efficient Processing of Network Proximity Queries via Chebyshev Acceleration. In *KDD*. 1515–1524.

[20] Mustafa Coskun, Ananth Grama, and Mehmet Koyutürk. 2018. Indexed Fast Network Proximity Querying. *VLDB* 11, 8 (2018), 840–852.

[21] Kun Dong, Austin R. Benson, and David Bindel. 2019. Network Density of States. In *KDD*.

[22] Nicole Eikmeier and David F. Gleich. 2017. Revisiting Power-law Distributions in Spectra of Real World Networks. In *KDD*. 817–826.

[23] Yasuhiro Fujiwara, Makoto Nakatsuji, Makoto Onizuka, and Masaru Kitsuregawa. 2012. Fast and Exact Top-k Search for Random Walk with Restart. *VLDB* (2012), 442–453.

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

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

[26] Heng Guo, Mark Jerrum, and Jingcheng Liu. 2017. Uniform sampling through the Lovasz local lemma. In *STOC*. 342–355.

[27] Manish S. Gupta, Amit Pathak, and Soumen Chakrabarti. 2008. Fast algorithms for topk personalized pagerank queries. In *WWW*. 1225–1226.

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

[29] Jinhong Jung, Namyong Park, Lee Sael, and U Kang. 2017. BePI: Fast and Memory-Efficient Method for Billion-Scale Random Walk with Restart. In *SIGMOD*. 789–804.

[30] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[31] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, and Guoren Wang. 2022. Efficient Personalized PageRank Computation: A Spanning Forests Sampling Based Approach. *Full version: https://github.com/mhLeon/RSFPPR* (2022).

[32] Dandan Lin, Raymond Chi-Wing Wong, Min Xie, and Victor Junqiu Wei. 2020. Index-Free Approach with Theoretical Guarantee for Efficient Random Walk with Restart Query. In *ICDE*. 913–924.

[33] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2016. Personalized PageRank Estimation and Search: A Bidirectional Approach. In *WSDM*. 163–172.

[34] Peter Lofgren and Ashish Goel. 2013. Personalized PageRank to a Target Node. *CoRR* abs/1304.4658 (2013). arXiv:1304.4658 http://arxiv.org/abs/1304.4658

[35] Takanori Maehara, Takuya Akiba, Yoichi Iwata, and Ken-ichi Kawarabayashi. 2014. Computing Personalized PageRank Quickly by Exploiting Graph Structures. *VLDB* 7, 12 (2014), 1023–1034.

[36] Philippe Marchal. 2000. Loop-erased random walks, spanning trees and Hamiltonian cycles. *Electronic Communications in Probability* 5 (2000), 39–50.

[37] PavelChebotarev. 2002. Spanning Forests of Digraphs and Limiting Probabilities of Markov Chains. *Electronic Notes in Discrete Mathematics* 11 (2002), 108–116.

[38] Yusuf Yigit Pilavci, Pierre-Olivier Amblard, Simon Barthelmé, and Nicolas Tremblay. 2021. Graph Tikhonov Regularization and Interpolation Via Random Spanning Forests. *IEEE Trans. Signal Inf. Process. over Networks* (2021), 359–374.

[39] Elena Shamis. 1994. Graph-theoretic interpretation of the generalized row sum method. *Mathematical Social Sciences* (1994), 321–333.

[40] Kijung Shin, Jinhong Jung, Lee Sael, and U Kang. 2015. BEAR: Block Elimination Approach for Random Walk with Restart on Large Graphs. In *SIGMOD*. 1571–1585.

[41] Julian Shun, Farbod Roosta-Khorasani, Kimon Fountoulakis, and Michael W. Mahoney. 2016. Parallel Local Graph Clustering. *VLDB* (2016), 1041–1052.

[42] Hanzhi Wang, Mingguo He, Zhewei Wei, Sibo Wang, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2021. Approximate Graph Propagation. In *KDD*. 1686–1696.

[43] Hanzhi Wang, Zhewei Wei, Junhao Gan, Sibo Wang, and Zengfeng Huang. 2020. Personalized PageRank to a Target Node, Revisited. In *KDD*. 657–667.

[44] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. HubPPR: Effective Indexing for Approximate Personalized PageRank. *VLDB* 10, 3 (2016), 205–216.

[45] Sibo Wang and Yufei Tao. 2018. Efficient Algorithms for Finding Approximate Heavy Hitters in Personalized PageRanks. In *SIGMOD*. 1113–1127.

[46] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *KDD*. 505–514.

[47] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. 2018. TopPPR: Top-k Personalized PageRank Queries with Precision Guarantees on Large Graphs. In *SIGMOD*. 441–456.

[48] David Bruce Wilson. 1996. Generating Random Spanning Trees More Quickly than the Cover Time. In *STOC*. 296–303.

[49] Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. 2021. Unifying the Global and Local Approaches: An Efficient Power Iteration with Forward Push. In *SIGMOD*. 1996–2008.

[50] Xiao-Ming Wu, Zhenguo Li, Anthony Man-Cho So, John Wright, and Shih-Fu Chang. 2012. Learning with Partially Absorbing Random Walks. In *NIPS*. 3086–3094.

[51] Yuan Yin and Zhewei Wei. 2019. Scalable Graph Embeddings via Sparse Transpose Proximities. In *KDD*.

[52] 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 *ICDE*. 1132–1143.

[53] Fanwei Zhu, Yuan Fang, Kevin Chen-Chuan Chang, and Jing Ying. 2013. Incremental and Accuracy-Aware Personalized PageRank through Scheduled Approximation. *VLDB* 6, 6 (2013), 481–492.