Efficient Personalized PageRank Computation: The Power of Variance-Reduced Monte Carlo Approaches

ABSTRACT

Personalized PageRank (PPR) computation is a fundamental problem in graph analysis. The state-of-the-art algorithms for PPR computation are based on a bidirectional framework which include a deterministic forward push and a Monte Carlo sampling procedure. The Monte Carlo sampling procedure, however, often has a relatively-large variance, thus reducing the performance of the PPR computation algorithms. To overcome this issue, we develop two novel variance-reduced Monte Carlo techniques for PPR computation. Our first technique is to apply power iterations to reduce the variance of the Monte Carlo sampling procedure. We prove that conducting few power iterations can significantly reduce the variance of existing Monte Carlo estimators, only with few additional costs. Moreover, we show that such a simple and novel variance-reduced Monte Carlo technique can achieve comparable estimation accuracy and the same time complexity as the state-of-the-art bidirectional algorithms. Our second technique is a novel progressive sampling method which uses the historical information of former samples to reduce the variance of the Monte Carlo estimator. We develop several novel PPR computation algorithms by integrating both of these variance reduction techniques with two existing Monte Carlo sampling approaches, including random walk sampling and spanning forests sampling. Finally, we conduct extensive experiments on 5 real-life large graphs to evaluate our solutions. The results show that our algorithms can achieve much higher PPR estimation accuracy by using much less time, compared to the state-of-the-art bidirectional algorithms.

1 INTRODUCTION

Personalized PageRank is an important and well-known concept in network analysis. Given a directed and unweighted graph $G = (V, E)$ with n nodes, a decay parameter α, and a source distribution $\boldsymbol{\sigma}$, the personalized PageRank (PPR) vector $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$ is defined as the probability that an α-random walk starts from a source node s, which is sampled from the distribution $\boldsymbol{\sigma}$, and stops at each node $u \in V$. Here an α-random walk is a random surfer on graph which stops at the current node with probability α, and walks to a random outgoing neighbor of the current node with probability $1 - \alpha$.

Intuitively, by the above definition, the PPR value $\boldsymbol{\pi}_{\boldsymbol{\sigma}}(t)$ measures the *importance* of a node t with respect to (w.r.t.) the source distribution $\boldsymbol{\sigma}$. When the source distribution $\boldsymbol{\sigma}$ is a one-hot vector $\mathbf{e}_s$ (only s-th element is 1 and all the other elements are 0), the resulting PPR vector is called a single-source PPR vector w.r.t. the source node s [19, 21, 42]. Such a single-source PPR vector can measure the similarities between the source node s and the other nodes in the graph, thus it is widely used in many graph analysis applications, such as web search [6, 21], link prediction [4], community detection [1, 35], recommendation [11, 23], and machine learning [7, 24, 46]. Additionally, when the source distribution is defined as $\boldsymbol{\pi} = \sum_{u \in V} \frac{1}{n} \boldsymbol{\pi}_u$, the PPR vector is the classic PageRank centrality vector, which is a fundamental metric to measure the importance of the nodes in a graph [10, 15]. Among these applications, graph clustering [27, 35] and graph learning [7, 24, 46] often require a small α (e.g., $\alpha = 0.01$), while node similarity measure and node ranking [6, 15, 21, 25] typically need a relatively-large α (e.g., $\alpha = 0.2$).

Due to such a wide range of practical applications, there exist many algorithms to efficiently compute the PPR vector of a graph. All of these algorithms can be roughly classified into two categories: deterministic algorithms and approximate algorithms. The deterministic PPR algorithms are mainly based on the power iteration [8, 34, 47] or the forward push techniques [1, 5, 21]. To achieve a high accuracy, such deterministic PPR algorithms are often inefficient on large graphs as they typically requires a large number of iterations. To address this issue, many approximate algorithms based on Monte Carlo sampling are proposed, including both the α-random walk sampling [2, 31] and the spanning forests sampling [27]. Recently, such Monte Carlo based approximate algorithms are further improved by the state-of-the-art bidirectional algorithms [27, 28, 31, 42, 45], which integrate both forward push and Monte Carlo sampling techniques. Despite many efforts have been made, the practical performance of these bidirectional algorithms is still unsatisfactory to achieve a high estimation accuracy [27], especially when the decay parameter α is small (e.g., $\alpha = 0.01$) which is often the demanding case for machine learning related applications [7, 35, 46]. The main reason behind this may be that the Monte Carlo sampling procedure in these bidirectional algorithms often have a large variance, thus it needs to draw a large number of samples to achieve a high accuracy.

To overcome this problem, we propose two novel variance reduction techniques to reduce the variances of two existing Monte Carlo sampling procedures, including both the α-random walk (α-RW) sampling [2, 31] and spanning forests (SF) sampling [27]. Specifically, our first variance reduction technique is to apply power iterations on the existing Monte Carlo estimators to reduce their variances. We prove that with only few additional power iterations, the variance of the existing Monte Carlo estimators can be substantially reduced (the variance is reduced by $(1 - \alpha)^{2K}$ times by only performing K power iterations). Note that compared to the state-of-the-art bidirectional algorithm SpeedPPR [45], the implementation of our variance-reduced Monte Carlo technique is much simpler. Moreover, we show that such a simple and novel variance-reduced Monte Carlo technique can achieve comparable accuracy and the same time complexity as the SpeedPPR algorithm.

Our second technique is a novel progressive sampling strategy which utilizes the historical information of former samples to reduce the variance of the existing Monte Carlo estimators. We show that by using the estimator constructed by the former samples, we can obtain useful information to improve the variance of the sampling procedure. Furthermore, such a progressive sampling technique can be easily integrated with our first power-iteration based technique to further reduce the variance. We develop several novel

SIGMOD '23, June 03–05, 2022, Woodstock, NY
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

PPR computation algorithms by integrating our variance reduction techniques with two existing Monte Carlo sampling approaches, including the α-random walk sampling and spanning forests sampling. Finally, the results of comprehensive experiments on 5 large real-world graphs demonstrate the efficiency and effectiveness of the proposed solutions. To summarize, the main contributions of this work are as follows.

New theoretical results. First, we present theoretical analyses for the variances of two existing Monte Carlo sampling techniques. A formal comparison of the variances of these existing estimators is also given. Second, we develop two novel and powerful variance reduction techniques, including a power-iteration and a progressive sampling based techniques, to reduce the variances of the existing estimators. Theoretical analyses for the variance reduction of our techniques are also presented.

Novel PPR computation algorithms. We develop several novel PPR computation algorithms by integrating our variance reduction techniques with two existing Monte Carlo sampling methods. Compared to the state-of-the-art SpeedPPR algorithm, our algorithms are extremely simple and easy to implement. Moreover, unlike all existing bidirectional algorithms, our algorithms do not use the forward push procedure. Instead, we use few power iterations as well as a progressive sampling technique to reduce the variance. To our knowledge, this is first work that can outperform existing bidirectional algorithms without using the forward push technique.

Extensive experiments. We conduct extensive experiments on 5 large real-life graphs to evaluate our algorithms. The results show that our algorithms substantially outperform the state-of-the-art algorithms on most datasets, in terms of both accuracy and running time. For example, on a graph with more than 3M nodes and 117M edges, when $\alpha = 0.01$, our best algorithm can compute the single-source PPR vector with $L1$ error 4.6×10^{-9} using only 114 seconds, while SpeedPPR can only obtain the $L1$ error 1.8×10^{-4} using 160 seconds. For reproducibility purpose, the source code of this paper is released at an anonymous link https://anonymous.4open.science/r/pvr-0C72.

2 PRELIMINARIES

Given a directed and unweighted graph $G = (V, E)$. Denote by $\boldsymbol{A}$ the adjacency matrix of G, $\boldsymbol{D}_{out}$ be the diagonal matrix with each element $(\boldsymbol{D}_{out})_{ii} = d_{out}(i)$, the out degree of node i. $\mathbf{P} = \boldsymbol{A}\boldsymbol{D}_{out}^{-1}$ is the probability transition matrix. PageRank [25] can be modeled as an α-random walk process. Given a source distribution $\boldsymbol{\sigma}$, we first sample a node s from the distribution $\boldsymbol{\sigma}$. Then, an α-random walk starts from s; and the personalized PageRank (PPR) value of node u is defined as the probability that the walk stops at u. Let $\mathbf{e}_s$ be a *one-hot* vector with the s-th element equals 1, and the other elements are 0. Let $\vec{1}$ be an all-1 vector. When $\boldsymbol{\sigma} = \mathbf{e}_s$, we use $\boldsymbol{\pi}_s$ to denote the personalized PageRank vector with respect to (w.r.t.) the source s. When $\boldsymbol{\sigma} = \frac{\vec{1}}{n}$, we use $\boldsymbol{\pi}$ to denote the PageRank centrality vector. We can also represent PageRank values in a matrix form. Let $\boldsymbol{\Pi}$ be the personalized PageRank matrix, where $(\boldsymbol{\Pi})_{st}$ denotes the personalized PageRank value $\pi(s,t)$ w.r.t. to the source s. We have $\boldsymbol{\Pi} = \frac{1}{\alpha}(\mathbf{I} - (1-\alpha)\mathbf{P})^{-1}$. It follows that $\boldsymbol{\pi}_s$ is the solution of a linear system $\mathbf{A}_\alpha \mathbf{x} = \mathbf{b}$, where $\mathbf{A}_\alpha = \mathbf{I} - (1-\alpha)\mathbf{P}$ and $\mathbf{b} = \alpha \mathbf{e}_s$. Also, when $\mathbf{b} = \alpha\frac{\vec{1}}{n}$, the solution vector of $\mathbf{A}_\alpha \mathbf{x} = \mathbf{b}$ is the PageRank centrality vector $\boldsymbol{\pi}$.

Given a graph G, a rooted spanning forest is a subgraph of G without cycle. A rooted spanning forest may have several connected components, each connected component has a unique node called "root" where all nodes in that component has a unique path towards the root. As a result, each node in G belongs to one of such connected components. We say a node s is rooted in t when s belongs to the connected component in which t is the root. For convenience, we denote $\rho[s] = t$.

For an n-dimensional vector $\mathbf{x}$, we define the $L1$-norm of $\mathbf{x}$ as $\|\mathbf{x}\|_1 = \sum_{u\in V} |\mathbf{x}(u)|$, and the $L2$-norm of $\mathbf{x}$ as $\|\mathbf{x}\|_2 = \sqrt{\sum_{u\in V} (\mathbf{x}(u))^2}$. For a random vector $\mathbf{x}$, we evaluate the variance of $\mathbf{x}$ by $Var[\mathbf{x}] = E[\|\mathbf{x} - E[\mathbf{x}]\|_2^2]$. Also, it can be written as the sum of the variance of all its elements, $Var[\mathbf{x}] = \sum_{u\in V} Var[\mathbf{x}(u)]$.

In this paper, we focus on the following problem.

Definition 2.1. (Approximate personalized PageRank (PPR) computation) Given a relative error threshold $\epsilon > 0$, a PPR threshold μ and a source distribution $\boldsymbol{\sigma}$, the approximate PPR computation problem aims to calculate an estimation $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(u)$ for each node $u \in V$ with $\boldsymbol{\pi}_{\boldsymbol{\sigma}}(u) \geq \mu$ such that $|\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(u) - \boldsymbol{\pi}_{\boldsymbol{\sigma}}(u)| \leq \epsilon \boldsymbol{\pi}_{\boldsymbol{\sigma}}(u)$ with a small failure probability p_f.

When $\boldsymbol{\sigma}$ is a one-hot vector, i.e. $\boldsymbol{\sigma} = \mathbf{e}_s$, the problem is identical to a single-source personalized PageRank query [42]. When $\boldsymbol{\sigma} = \frac{\vec{1}}{n}$, the problem is to compute a PageRank centrality vector. In practice, the threshold μ is often set to $\frac{1}{n}$ so that it can guarantee a relative error for most relevant nodes [42, 45]. The failure probabilityp_f is also set to $\frac{1}{n}$ to ensure a vary small failure probability [42, 45].

2.1 Basic PPR Computation Techniques

Existing personalized PageRank estimators. There mainly exist three estimators for estimating single-source personalized PageRank vector $\boldsymbol{\pi}_s$. One is based on the α-random walk sampling, and the other two are based on spanning forests sampling.

Lemma 2.2. *([31, 42]) Let $\bar{\mathbf{x}}$ be a vector of random variables. If an α-random walk starts from s and stops at node t, we set $\bar{\mathbf{x}} = \mathbf{e}_t$. Then, $\bar{\mathbf{x}}$ is an unbiased estimator of $\boldsymbol{\pi}_s$, i.e., $E[\bar{\mathbf{x}}] = \boldsymbol{\pi}_s$.*

Lemma 2.3. *([27]) Let $F \in \mathcal{F}$ be a random rooted spanning forest and $\rho(F)$ be the root set of F. Suppose that F is sampled with probability $P(F) \propto \prod_{u\in\rho(F)} \frac{\alpha}{1-\alpha} d_{out}(u)$, and s is rooted in t in F. Denote by $\tilde{\mathbf{x}}$ a vector of random variables with $\tilde{\mathbf{x}} = \mathbf{e}_t$. Then, $\tilde{\mathbf{x}}$ is an unbiased estimator of $\boldsymbol{\pi}_s$, i.e., $E[\tilde{\mathbf{x}}] = \boldsymbol{\pi}_s$.*

If the graph is undirected, we have $d_{in}(u) = d_{out}(u) = d(u)$. Then, there is an improved spanning forest based estimator which considers the partition information of the spanning forest [27].

Lemma 2.4. *([27]) Let $F \in \mathcal{F}$ be a random rooted spanning forest and $\rho(F)$ be the root set of F. Suppose that F is sampled with probability $P(F) \propto \prod_{u\in\rho(F)} \frac{\alpha}{1-\alpha} d(u)$, and $V_{\rho[s]}$ is the node set of the connected component of F that s belongs to. Let $\dot{\mathbf{x}}$ be a a vector of random variables with $\dot{\mathbf{x}} = \sum_{u\in V_{\rho[s]}} \frac{d(u)}{\sum_{v\in V_{\rho[s]}} d(v)} \mathbf{e}_u$. Then, $\dot{\mathbf{x}}$ is an unbiased estimator of $\boldsymbol{\pi}_s$, i.e., $E[\dot{\mathbf{x}}] = \boldsymbol{\pi}_s$.*

Note that all the above three estimators can be easily generalized for arbitrary source distribution $\boldsymbol{\sigma}$, which we will discuss in Section 3 and Section 4.

Loop-erased α-random walk. In order to sample spanning forests from the desired probability distribution, a loop-erased α-random walk sampling algorithm was proposed in [27] which is a generalization of the classic Wilson algorithm [44]. The algorithm is outlined in Algorithm 1. First, the algorithm fixes an ordering of V (Line 2). Then, it traverses all the nodes following this ordering (Line 3-14). It makes use of an array *InForest* to record whether a

Algorithm 1: Loop-erased α-random walk [27]

```
Input: A graph $G = (V, E)$ and a decay parameter $\alpha$
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      $u = v_i$;
5      while $!InForest[u]$ do
6          if $rand() < \alpha$ then
7              $InForest[u] \leftarrow true, Root[u] \leftarrow u$;
8          else
9              $Next[u] \leftarrow RandomNeighbor(u)$;
10             $u \leftarrow Next[u]$;
11     $r \leftarrow Root[u], u \leftarrow v_i$;
12     while $!InForest[u]$ do
13         $InForest[u] \leftarrow true, Root[u] \leftarrow r$;
14         $u \leftarrow Next[u]$;
15 return $Root[u]$ for all $u \in V$;
```

node has been added into the spanning forest, uses an array $Next$ to maintain the next node in the spanning forest, and utilizes an array $Root$ to store the root information. An α-random walk is simulated until it stops or hits the former trajectories (Line 4-10), and the loops in the walk are erased by retracing the trajectories (Line 11-14). The time complexity of Algorithm 1 is $\frac{1}{\alpha}\sum_{u\in V}\pi(u,u)$, which is smaller than sampling n α-random walks (i.e., $\frac{n}{\alpha}$) [27]. When the algorithm terminates, a spanning forest F is maintained in $Root$ with probability $P(F) \propto \prod_{u\in\rho(F)}\frac{\alpha}{1-\alpha}d_{out}(u)$.

Forward search. As shown in Algorithm 2, the forward search method can be deemed as a deterministic version of α-random walk sampling [1]. It maintains the residues $\hat{\mathbf{r}}(u)$ and reserves $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(u)$ for all $u \in V$, where $\hat{\mathbf{r}}$ and $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$ are initialized as $\boldsymbol{\sigma}$ and $\mathbf{0}$ respectively (Lines 1). The algorithm performs a *deterministic* traversal on the graph and updates $\hat{\mathbf{r}}$ and $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$ accordingly (Lines 2-6). Specifically, for each node u with residual larger than $d_{out}(u)\cdot r_{max}$, it converts α fraction of u's residual into its reserve (Line 3), and equally distributes the other $1-\alpha$ fraction of u's residual to its out-neighbors (Line 5-6). During this procedure, an invariant is maintained for all $u \in V$ [1]:

$$\boldsymbol{\pi}_{\boldsymbol{\sigma}}(u) = \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(u) + \sum_{v\in V}\hat{\mathbf{r}}(v)\boldsymbol{\pi}_v(u). \tag{1}$$

It was shown that the algorithm runs in $O(1/r_{max})$ time [1]. When r_{max} approaches 0, $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(u)$ converges to $\boldsymbol{\pi}_{\boldsymbol{\sigma}}(u)$. Moreover, it is easy to see that Algorithm 2 can also be used for any source distribution.

2.2 The State-of-the-art Solutions

For single-source PPR query, Wang et al. proposed FORA [42] which combines forward search and α-random walk sampling. Specifically, to answer a query for $\boldsymbol{\pi}_s$, it first performs a forward search, then generates α-random walks from those nodes with non-zero residues. ResAcc [28] and SpeedPPR [45] further improved FORA by accumulating residues and combining power iterations with forward search. SpeedPPR outperforms all former approximate PPR computation algorithms on directed graphs. However, all these algorithms perform poorly when α is small (e.g., $\alpha = 0.01$). To address the small α case, Liao et al. [27] proposed SpeedL and SpeedLV to further improve SpeedPPR by combining forward push with spanning forests sampling. When the graph is undirected, SpeedL and SpeedLV can achieves the state-of-the-art performance for the small α case.

Unlike single-source personalized PageRank computation, algorithms for PageRank centrality computation are less well studied.

Algorithm 2: Forward Search [1]

```
Input: A graph $G$, source distribution $\boldsymbol{\sigma}$, decay parameter $\alpha$, threshold $r_{\max}$
Output: Residual vector $\hat{\mathbf{r}}$ and estimation vector $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$
1  $\hat{\mathbf{r}} \leftarrow \boldsymbol{\sigma}, \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} \leftarrow \mathbf{0}$;
2  while $\exists u \in V$ such that $\hat{\mathbf{r}}(u) \geq d_{out}(u)\cdot r_{max}$ do
3      $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(u) \leftarrow \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(u) + \alpha\hat{\mathbf{r}}(u)$;
4      for each $w \in N^{out}(u)$ do
5          $\hat{\mathbf{r}}(w) \leftarrow \hat{\mathbf{r}}(w) + (1-\alpha)\frac{\hat{\mathbf{r}}(u)}{d_{out}(u)}$;
6      $\hat{\mathbf{r}}(u) \leftarrow 0$;
7  return $\hat{\mathbf{r}}, \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$;
```

Note that $\boldsymbol{\pi} = \sum_{s\in V}\frac{1}{n}\boldsymbol{\pi}_s$, $\boldsymbol{\pi}$ can be simply estimated by first uniformly sampling a node $u \in V$, and then running an α-random walk from u. Suppose that the walk stops at t, then $\mathbf{e}_t$ is an unbiased estimator of $\boldsymbol{\pi}$. To our knowledge, one of best existing approximate algorithms that is tailored to PageRank centrality computation is such a Monte Carlo solution proposed in [2]. Note that FORA can also be extended to PageRank centrality computation [41], which we will add for comparison in our experiments (see Section 5.4).

3 NEW α-RANDOM WALK ESTIMATORS

In this section, we first conduct a detailed analysis for the variance of the traditional α-random walk (α-RW) estimator $\bar{\mathbf{x}}$. Then, we develop two novel variance reduction techniques to reduce the variance of $\bar{\mathbf{x}}$. Our first technique is to apply power iterations to reduce the variance, while the second technique is to utilize the historical information of former samples to reduce the variance.

3.1 Variance Analysis of α-RW Estimator

Although the α-random walk sampling is widely used to estimate a single-source personalized PageRank vector $\boldsymbol{\pi}_s$, it is easy to extend it to estimate PPR vector with any source distribution $\boldsymbol{\sigma}$. Specifically, similar to Lemma 2.2, we have the following results.

Lemma 3.1. *Let $\bar{\mathbf{x}}$ be a vector of random variables. If an α-random walk starts from a node sampled from $\boldsymbol{\sigma}$ and stops at node t, we set $\bar{\mathbf{x}} = \mathbf{e}_t$. Then, $\bar{\mathbf{x}}$ is an unbiased estimator of $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$, i.e., $E[\bar{\mathbf{x}}] = \boldsymbol{\pi}_{\boldsymbol{\sigma}}$.*

Proof. By the linearity of expectation, since $\boldsymbol{\pi}_{\boldsymbol{\sigma}} = \sum_{u\in V}\boldsymbol{\sigma}(u)\boldsymbol{\pi}_u$, we have $E[\bar{\mathbf{x}}] = \sum_{u\in V}\boldsymbol{\sigma}(u)\boldsymbol{\pi}_u = \boldsymbol{\pi}_{\boldsymbol{\sigma}}$. □

Based on Lemma 3.1, we can easily obtain a basic Monte Carlo algorithm [2], namely MCW, which is shown in Algorithm 3. If we have built an alias table [36] from $\boldsymbol{\sigma}$, the process of sampling a node from a certain distribution (Line 3) can be implemented via an alias sampling process using $O(1)$ time. Thus, each walk can be simulated within $O(\frac{1}{\alpha})$ time in expectation. We derive the variance of $\bar{\mathbf{x}}$ as follows.

Lemma 3.2. *Suppose that we simulate α-random walks from a node sampled from $\boldsymbol{\sigma}$, and use $\bar{\mathbf{x}}$ as an unbiased estimator of $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$. Then, the variance of the estimator $\bar{\mathbf{x}}$ is: $Var[\bar{\mathbf{x}}] = 1 - \|\boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_2^2$.*

Proof. Let $\bar{\mathbf{x}}(t)$ be the random variable such that if an α-random walk starts from a node sampled from $\boldsymbol{\sigma}$ and stops at t, $\bar{\mathbf{x}}(t) = 1$, and $\bar{\mathbf{x}}(t) = 0$ otherwise. Then, we have $E[(\bar{\mathbf{x}}(t))^2] = E[\bar{\mathbf{x}}(t)] = \boldsymbol{\pi}_{\boldsymbol{\sigma}}(t)$. Thus, $Var[\bar{\mathbf{x}}(t)] = E[(\bar{\mathbf{x}}(t))^2] - E[\bar{\mathbf{x}}(t)]^2 = \boldsymbol{\pi}_{\boldsymbol{\sigma}}(t) - (\boldsymbol{\pi}_{\boldsymbol{\sigma}}(t))^2$. So the variance of $\bar{\mathbf{x}}$ can be written as $Var[\bar{\mathbf{x}}] = \sum_{t\in V} Var[\bar{\mathbf{x}}(t)] = 1 - \sum_{t\in V}(\boldsymbol{\pi}_{\boldsymbol{\sigma}}(t))^2 = 1 - \|\boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_2^2$. □

According to Lemma 3.2, the variance of $\bar{\mathbf{x}}$ is closely related to the distribution $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$. Since $\|\boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_1 = 1$, we have $\frac{1}{n} \leq \|\boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_2^2 \leq 1$. When $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$ is a *one-hot* distribution (i.e., $\boldsymbol{\sigma} = \mathbf{e}_s$ and $\boldsymbol{\pi}_{\boldsymbol{\sigma}} = \mathbf{e}_s$, which is the case that there is no outgoing-edge from s), we have $Var[\bar{\mathbf{x}}] =$

Algorithm 3: MCW [2]

```
Input: A graph $G$, source distribution $\boldsymbol{\sigma}$, decay parameter $\alpha$, sample size $T$
Output: The estimated PageRank vector $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$
1 $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} \leftarrow \mathbf{0}$;
2 for $i = 1 : T$ do
3     Sample a node $s'$ from $\boldsymbol{\sigma}$;
4     Run an $\alpha$-random walk from $s'$; suppose that it stops at $t$;
5     $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(t) \leftarrow \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(t) + \frac{1}{T}$;
6 return $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$;
```

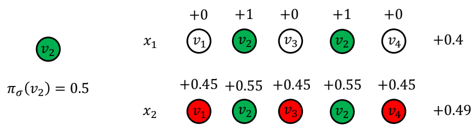


Figure 1: Illustration of the variances of different estimators. Here both x_1 and x_2 are unbiased estimators of $\boldsymbol{\pi}_{\boldsymbol{\sigma}}(v_2)$ and $Var[x_2] < Var[\boldsymbol{x}_1]$.

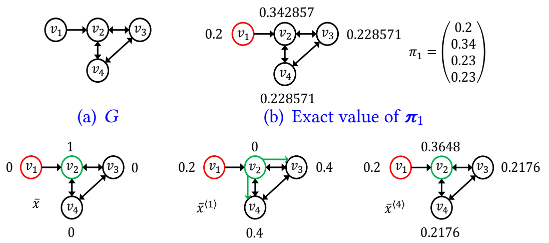
$1 - 1 = 0$. In this case, each α-random walk starts from s can only stop at s. Thus, the estimation of such a PPR vector is easy. However, when $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$ is a "*balanced*" distribution (i.e., $\frac{\vec{1}}{n}$, the all-one distribution), the variance becomes $Var[\bar{\mathbf{x}}] = 1 - \sum_{t \in V}(\boldsymbol{\pi}_{\boldsymbol{\sigma}}(t))^2 = 1 - \frac{1}{n}$. Clearly, such a case is the *hardest* case to estimate the PPR vector using α-random walk sampling.

Note that the variance plays an important role in sampling-based approximate algorithms. To illustrate this, we give an example in Fig. 1. Suppose that both $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are unbiased estimators of $\boldsymbol{\pi}_{\boldsymbol{\sigma}}(v_2) = 0.5$, i.e., $E[\boldsymbol{x}_1] = E[\boldsymbol{x}_2] = \boldsymbol{\pi}_{\boldsymbol{\sigma}}(v_2)$. In practice, to achieve a high accuracy, approximate algorithms always draw a number of samples and take the average value over all samples as an estimator. As shown in Fig. 1, the sample value of $\boldsymbol{x}_2$ is close to each other, thus $Var[\boldsymbol{x}_2] < Var[\boldsymbol{x}_1]$. Although they have the same expectation value, with 5 samples, the estimation value of $\boldsymbol{x}_1$ is 0.4 while the estimation value of $\boldsymbol{x}_2$ is 0.49, which is closer to the exact value. Estimator with smaller variance requires less samples to achieve a desired accuracy, thus reducing the running time for drawing samples. The worst-case time complexity of Algorithm 3 to guarantee an (ϵ, δ)-error is $O(\frac{nlogn}{\epsilon^2})$, by the standard Chernoff bound [42]. Below, we will propose two novel ideas to reduce the variance of $\bar{\mathbf{x}}$. We show that the variance reduction of our technique is substantial with only few additional time costs.

3.2 Variance Reduction by Power Iterations

The intuition of our first technique is that the closer an estimator is concentrated around $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$, the smaller the variance of the estimator is. According to the property of the power iterations, applying more power iterations on any vector $\mathbf{x}$ (i.e., $\mathbf{x}^{(t+1)} = \alpha\boldsymbol{\sigma} + (1-\alpha)\mathbf{P}\mathbf{x}^{(t)}$) will make the result converge towards $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$. This motivates us to use power iterations to reduce the variance of the estimator $\bar{\mathbf{x}}$, since performing power iterations can make the estimator $\bar{\mathbf{x}}$ close to $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$. Note that a power iteration only requires a traversal of all edges once which takes $O(m)$ time. On real-life scale-free graphs, we often have $O(m) = O(n \log n)$, thus it is negligible compared to the $O(\frac{n \log n}{\epsilon^2})$ time complexity of Algorithm 3.

Warm up: using one power iteration. We first consider the simple case that we apply only one power iteration to reduce the variance of the estimator $\bar{\mathbf{x}}$. Recall that according to the definition of the PageRank vector, $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$ satisfies the recursive formula: $\boldsymbol{\pi}_{\boldsymbol{\sigma}} = \alpha\boldsymbol{\sigma} + (1-\alpha)\mathbf{P}\boldsymbol{\pi}_{\boldsymbol{\sigma}}$. Then, we have the following result.



v_1 v_2 v_3 v_4
(a) G
0.342857
0.2
0.228571
0.228571
$\pi_1 = \begin{pmatrix} 0.2 \\ 0.34 \\ 0.23 \\ 0.23 \end{pmatrix}$
(b) Exact value of $\boldsymbol{\pi}_1$
1
0
0
0
$\bar{x}$
0
0.2
0.4
0.4
$\bar{x}^{\langle 1 \rangle}$
0.3648
0.2
0.2176
0.2176
$\bar{x}^{\langle 4 \rangle}$
(c) The estimation value of $\bar{\mathbf{x}}$, $\bar{\mathbf{x}}^{\langle 1 \rangle}$ and $\bar{\mathbf{x}}^{\langle 4 \rangle}$

Figure 2: A running example of the estimator $\bar{\mathbf{x}}^{\langle K \rangle}$. (a) An example graph G; (b) The exact value of the PPR vector $\boldsymbol{\pi}_1$ ($\alpha = 0.2$); (c) Suppose that an α-random walk starts from v_1 and stops at v_2, the estimation values of $\bar{\mathbf{x}}$, $\bar{\mathbf{x}}^{\langle 1 \rangle}$ and $\bar{\mathbf{x}}^{\langle 4 \rangle}$ are depicted. Compared to $\bar{\mathbf{x}}$, the estimation values of $\bar{\mathbf{x}}^{\langle 1 \rangle}$ and $\bar{\mathbf{x}}^{\langle 4 \rangle}$ are closer to the exact value, which results in smaller variances.

LEMMA 3.3. *Let $\bar{\mathbf{x}}$ be an unbiased estimator of $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$. Then, $\bar{\mathbf{x}}^{\langle 1 \rangle} = \alpha\boldsymbol{\sigma} + (1-\alpha)\mathbf{P}\bar{\mathbf{x}}$ is also an unbiased estimator of $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$.*

PROOF. By the linearity of expectation, we have $E[\bar{\mathbf{x}}^{\langle 1 \rangle}] = E[\alpha\boldsymbol{\sigma} + (1-\alpha)\mathbf{P}\bar{\mathbf{x}}] = \alpha\boldsymbol{\sigma} + (1-\alpha)\mathbf{P}E[\bar{\mathbf{x}}] = \alpha\boldsymbol{\sigma} + (1-\alpha)\mathbf{P}\boldsymbol{\pi}_{\boldsymbol{\sigma}} = \boldsymbol{\pi}_{\boldsymbol{\sigma}}$. □

By Lemma 3.3, we can construct a new estimator $\bar{\mathbf{x}}^{\langle 1 \rangle}$ based on the estimator $\bar{\mathbf{x}}$. Recall that to derive $\bar{\mathbf{x}}$, we need to add a mass of 1 on t if an α-random walk stops at the node t. To construct $\bar{\mathbf{x}}^{\langle 1 \rangle}$, however, we add α times the source distribution back to the corresponding nodes, and update t's out-neighbors by adding $\frac{1-\alpha}{d_{out}(t)}$ on their estimations. In other words, the new estimator uniformly distributes $1-\alpha$ times of the probability mass to t's out-neighbors. This operation can reduce the variance of the estimator $\bar{\mathbf{x}}$. Fig. 2 shows a running example for $\bar{\mathbf{x}}^{\langle 1 \rangle}$. To estimate the PPR vector $\boldsymbol{\pi}_1$ on the example graph G ($\alpha = 0.2$), when an α-random walk starts from v_1 and stops at v_2, the estimator $\bar{\mathbf{x}}$ adds 1 on the termination node v_2. For comparison, $\bar{\mathbf{x}}^{\langle 1 \rangle}$ first adds the source node v_1 by $\alpha = 0.2$, then instead of adding amount on the termination node v_2, it adds $(1-\alpha)/d_{out}(v_2)$ on the out-neighbors (v_3, v_4) of the termination node v_2. The resulting estimation value of $\bar{\mathbf{x}}^{\langle 1 \rangle}$ is closer to the exact value compared to $\bar{\mathbf{x}}$, which results in a smaller variance. The following lemma shows that given an arbitrary unbiased estimator $\mathbf{x}$, applying one power iteration on it will obtain a new estimator that has variance $(1-\alpha)^2$ times smaller than the original estimator $\mathbf{x}$.

LEMMA 3.4. *Let $\mathbf{x}$ be an unbiased estimator of $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$ and $\mathbf{x}^{\langle 1 \rangle} = \alpha\boldsymbol{\sigma} + (1-\alpha)\mathbf{P}\mathbf{x}$. Then, we have $Var[\mathbf{x}^{\langle 1 \rangle}] \le (1-\alpha)^2 Var[\mathbf{x}]$.*

PROOF. By definition, we have $Var[\mathbf{x}^{\langle 1 \rangle}] = E[\|\mathbf{x}^{\langle 1 \rangle} - \boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_2^2]$ and $Var[\mathbf{x}] = E[\|\mathbf{x} - \boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_2^2]$. Then, we have $\|\mathbf{x}^{\langle 1 \rangle} - \boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_2^2 = \|\alpha\boldsymbol{\sigma} + (1-\alpha)\mathbf{P}\mathbf{x} - \boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_2^2 = \|\alpha\boldsymbol{\sigma} + (1-\alpha)\mathbf{P}\mathbf{x} - (\alpha\boldsymbol{\sigma} + (1-\alpha)\mathbf{P}\boldsymbol{\pi}_{\boldsymbol{\sigma}})\|_2^2 = \|(1-\alpha)\mathbf{P}(\mathbf{x} - \boldsymbol{\pi}_{\boldsymbol{\sigma}})\|_2^2 = (1-\alpha)^2\|\mathbf{P}(\mathbf{x} - \boldsymbol{\pi}_{\boldsymbol{\sigma}})\|_2^2$. Note that $\mathbf{P}$ is a stochastic matrix. Since all the eigenvalues of a stochastic matrix have absolute values less than or equal to one (by the classic Gershgorin circle theorem), we have $\|\mathbf{P}(\mathbf{x} - \boldsymbol{\pi}_{\boldsymbol{\sigma}})\|_2^2 \le \|\mathbf{x} - \boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_2^2$. As a result, for any possible value of $\mathbf{x}$, we have $\|\mathbf{x}^{\langle 1 \rangle} - \boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_2^2 \le (1-\alpha)^2\|\mathbf{x} - \boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_2^2$. Thus, we can obtain that $Var[\mathbf{x}^{\langle 1 \rangle}] \le (1-\alpha)^2 Var[\mathbf{x}]$. □

Reducing variance by K-power iterations. Given that applying one power iteration can reduce variance, a natural idea is to extend it by using K power iterations to further reduce variance. First, we prove that there is a similar invariant by iteratively applying

Algorithm 4: PW

```
Input: graph $G$, source distribution $\boldsymbol{\sigma}$, decay factor $\alpha$, power iteration number $K$, sample
    size $T$
Output: Estimated PageRank vector $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$
1 $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} \leftarrow$ MCW $(G, \boldsymbol{\sigma}, \alpha, T)$;
2 $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} \leftarrow \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} + (1-\alpha)^K \mathbf{P}^K \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$;
3 return $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$;
```

$\boldsymbol{\pi}_{\boldsymbol{\sigma}} = \alpha\boldsymbol{\sigma} + (1-\alpha)\mathbf{P}\boldsymbol{\pi}_{\boldsymbol{\sigma}}$ for K times. Then, a new unbiased estimator can be derived by such an invariant formula.

Lemma 3.5. *For any $K > 0$, we have*

$$\boldsymbol{\pi}_{\boldsymbol{\sigma}} = \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} + (1-\alpha)^K \mathbf{P}^K \boldsymbol{\pi}_{\boldsymbol{\sigma}}. \tag{2}$$

Proof. By the definition of $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$, it can be expanded as $\boldsymbol{\pi}_{\boldsymbol{\sigma}} = \sum_{k=0}^{\infty} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma}$. We have:

$$\begin{aligned}
\boldsymbol{\pi}_{\boldsymbol{\sigma}} &= \sum_{k=0}^{\infty} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} \\
&= \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} + \sum_{k=K}^{\infty} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} \\
&= \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} + (1-\alpha)^K \mathbf{P}^K \sum_{k=0}^{\infty} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} \\
&= \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} + (1-\alpha)^K \mathbf{P}^K \boldsymbol{\pi}_{\boldsymbol{\sigma}}.
\end{aligned}$$

□

Lemma 3.6. *Suppose that an α-random walk starts from a node s sampled from distribution $\boldsymbol{\sigma}$ and stops at t. Then, for any $K > 0$, $\bar{\mathbf{x}}^{\langle K \rangle} = \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} + (1-\alpha)^K \mathbf{P}^K \mathbf{e}_t$ is an unbiased estimator of $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$.*

Proof. According to Lemma 3.1, $E[\bar{\mathbf{x}}] = \boldsymbol{\pi}_{\boldsymbol{\sigma}}$. Following the linearity of expectation, we can derive that

$$\begin{aligned}
E[\bar{\mathbf{x}}^{\langle K \rangle}] &= E[\sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} + (1-\alpha)^K \mathbf{P}^K \mathbf{e}_t] \\
&= \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} + (1-\alpha)^K \mathbf{P}^K E[\bar{\mathbf{x}}] \\
&= \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} + (1-\alpha)^K \mathbf{P}^K \boldsymbol{\pi}_{\boldsymbol{\sigma}} \\
&= \boldsymbol{\pi}_{\boldsymbol{\sigma}}.
\end{aligned}$$

□

Furthermore, we can easily derive an upper bound of $Var[\bar{\mathbf{x}}^{\langle K \rangle}]$ by iteratively applying Lemma 3.4 for K times.

Lemma 3.7. $Var[\bar{\mathbf{x}}^{\langle K \rangle}] \le (1-\alpha)^{2K}(1 - \|\boldsymbol{\pi}_{\sigma}\|_2^2)$.

For example, in Fig. 2, by iteratively applying the power iteration for four times, the resulting estimation value of $\bar{\mathbf{x}}^{\langle 4 \rangle}$ is much closer to the exact PPR vector $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$ compared to $\bar{\mathbf{x}}^{\langle 1 \rangle}$. This indicates that the estimator $\bar{\mathbf{x}}^{\langle 4 \rangle}$ has a smaller variance compared to $\bar{\mathbf{x}}^{\langle 1 \rangle}$.

Note that the first term in the right hand side of Eq. (2) can be seen as an estimation of $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$. And we have $\|\boldsymbol{\pi}_{\boldsymbol{\sigma}} - \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma}\|_1 = (1-\alpha)^K$. Clearly, when K becomes large, the $L1$-error of the first term is very small. For $\bar{\mathbf{x}}^{\langle K \rangle}$, we only need to approximate the small error part (the second term of Eq. (2)), since the first term can be computed deterministically. For an α-random walk sample, we assume that it stops at the node t, then the estimator deterministically propagates $\mathbf{e}_t$ on the graph for K steps to estimate the second term. Clearly, if there are T α-random walk samples and T is large, the total time costs of this procedure may be high. A nice trick to tackle this problem is that we can first sample T α-random walk samples, and then deterministically propagate the values of the α-random walk estimator on the graph for K steps (i.e., performing K power iterations). By the linearity of the PPR vector, it can be shown that such a procedure can exactly obtain the same estimator as that based on $\bar{\mathbf{x}}^{\langle k \rangle}$.

Lemma 3.8. *Let $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}1}$ be the estimation vector obtained by applying $\bar{\mathbf{x}}$ for T times and take the average, and then applying K-power iterations on the results. Let $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}2}$ be the estimation vector obtained by applying $\bar{\mathbf{x}}^{\langle K \rangle}$ for T times and take the average. Then, we have $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}1} = \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}2}$.*

Proof. Suppose that the i-th α-random walk stops at t_i. After obtaining T random walk samples, we have $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} = \sum_{i=1}^{T} \frac{\mathbf{e}_{t_i}}{T}$. Then, $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}1} = \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} + (1-\alpha)^K \mathbf{P}^K (\sum_{i=1}^{T} \frac{\mathbf{e}_{t_i}}{T})$. On the other hand, if we use $\bar{\mathbf{x}}^{\langle K \rangle}$ as an estimator, for the i-th random walk, the algorithm will add $\mathbf{z}_i = \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \frac{\boldsymbol{\sigma}}{T} + (1-\alpha)^K \mathbf{P}^K \frac{\mathbf{e}_{t_i}}{T}$. As a result, $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}2} = \sum_{i=1}^{T} \mathbf{z}_i = \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} + \sum_{i=1}^{T} (1-\alpha)^K \mathbf{P}^K \frac{\mathbf{e}_{t_i}}{T} = \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} + (1-\alpha)^K \mathbf{P}^K (\sum_{i=1}^{T} \frac{\mathbf{e}_{t_i}}{T}) = \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}1}$. □

A Monte Carlo algorithm PW based on the estimator $\bar{\mathbf{x}}^{\langle K \rangle}$ is outlined in Algorithm 4. The algorithm follows a "first walk, and then propagate" framework. Note that this framework is fundamentally different from the state-of-the-art bidirectional approaches [31, 42, 45] which are based on the "first propagate, and then walk" framework. Specifically, Algorithm 4 first simulates T α-random walks to derive an estimation $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$ of $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$ (Line 1). Then, K power iterations are applied on $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$ to construct the final estimator. Since each power iteration takes $O(m)$ time, the time complexity of PW can be easily derived.

Lemma 3.9. *The time complexity of Algorithm 4 is $O(\frac{T}{\alpha} + Km)$.*

Note that PW is extremely simple. However, the performance of PW is competitive with the state-of-the-art bidirectional algorithms as shown in our experiments. We can utilize the following Chernoff inequality to bound the sample size of Algorithm 4 to guarantee an (ϵ, δ)-error.

Lemma 3.10. *(Chernoff bound) Let $X_i (1 \le i \le n_r)$ be independent random variables satisfying $X_i \le M$ for $1 \le i \le T$. Let $X = \frac{1}{T}\sum_{i=1} X_i$. Assume that $E[X]$ be the expectation of X, $\|X\|^2 = \frac{1}{T}\sum_{i=1}^{T} E[X_i^2]$. Then we have*

$$\Pr(|X - E[X]| \ge \lambda) \le exp(-\frac{\lambda^2 T}{2(\|X\|^2 + M\lambda/3)}).$$

Lemma 3.11. *Let $W = \frac{(2\epsilon/3+2)\log(2/p_f)}{\epsilon^2 \cdot \mu}$. For any node t with $\boldsymbol{\pi}_{\boldsymbol{\sigma}}(t) > \mu$, when $K > log_{1-\alpha}\mu$ and $T > (1-\alpha)^K W$, Algorithm 4 returns an approximate PPR value $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(t)$ satisfying $|\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(t) - \boldsymbol{\pi}_{\boldsymbol{\sigma}}(t)| \le \epsilon\boldsymbol{\pi}_{\boldsymbol{\sigma}}(t)$ with probability at least $1 - p_f$.*

Proof. Let X be the random variable that represents the amount added on node t in each α-random walk sample (the second term of estimator $\bar{\mathbf{x}}^{\langle K\rangle}$). Suppose that the random walk stops at node t'. Let $X = (1-\alpha)^K \mathbf{P}^K \mathbf{e}_{t'}(t)$. Then, we have $X \le (1-\alpha)^K$. Given that $1-\alpha < 1$, for any i, we have $X_i \le (1-\alpha)^K$. Thus, we have $\|X\|^2 \le ((1-\alpha)^K)^2 < (1-\alpha)^K \mu < (1-\alpha)^K \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(t)$. By substituting $M = (1-\alpha)^K$ in Lemma 3.10, we have

$$\Pr(|X - E[X]| \ge \lambda) \le exp(-\frac{\lambda^2 T}{2(1-\alpha)^K(\boldsymbol{\pi}_{\boldsymbol{\sigma}}(t) + \lambda/3)}).$$

Let $\lambda = \epsilon \boldsymbol{\pi}_{\boldsymbol{\sigma}}(t)$. Then, by $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(t) - \boldsymbol{\pi}_{\boldsymbol{\sigma}}(t) = X - E[X]$, we have

$$\Pr(|\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(t) - \boldsymbol{\pi}_{\boldsymbol{\sigma}}(t)| \ge \epsilon \boldsymbol{\pi}_{\boldsymbol{\sigma}}(t)) \le exp(-\frac{\epsilon^2 \cdot T \cdot \boldsymbol{\pi}_{\boldsymbol{\sigma}}(t)}{(1-\alpha)^K(2+2\epsilon/3)}) \le p_f.$$

The last inequality follows by substituting $T > (1-\alpha)^K W$ and $\boldsymbol{\pi}_{\boldsymbol{\sigma}}(t) > \mu$. □

The analysis of Lemma 3.9 and Lemma 3.11 is general. For example, if the graph has $m = O(n^{\frac{3}{2}})$ edges, the time complexity of PW is $O(\frac{T}{\alpha} + Kn^{\frac{3}{2}})$. In practice, real-life scale-free graphs with $m = O(n \log n)$ is of special interest. Since most previous studies [42, 45] consider scale-free graphs, we also focus on such graphs for a fair comparison. Also, we set $p_f = \frac{1}{n}, \mu = \frac{1}{n}$ to ensure a relatively-accurate estimation result. This parameter setting is also widely adopted in previous work [42, 45]. With these parameter setting, we can easily derive the following Corollary from Lemma 3.11.

Corollary 1. *Suppose that $m = O(n \log n)$, $p_f = \frac{1}{n}$, $\mu = \frac{1}{n}$, Algorithm 4 can achieve an $(\sqrt{\frac{n \log n (1-\alpha)^K}{T}}, \frac{1}{n})$-error with probability larger than $1 - \frac{1}{n}$.*

Proof. According to Lemma 3.11, $W = O(\frac{n \log n}{\epsilon^2})$, by setting $\epsilon = \sqrt{\frac{n \log n (1-\alpha)^K}{T}}$, T and K satisfy $T = (1-\alpha)^K W$, thus the algorithm can satisfy an (ϵ, μ)-error. □

According to Corollary 1, we can vary T and K; as long as $\epsilon = \sqrt{\frac{n \log n (1-\alpha)^K}{T}}$ is a constant, the same error guarantee holds. The time complexity is $O(\frac{T}{\alpha} + Km) = O((\frac{(1-\alpha)^K}{\alpha\epsilon^2} + K) n \log n)$, which is minimized when $K = \log_{1-\alpha} \frac{\alpha}{ln \frac{1}{1-\alpha}} \epsilon^2 = O(\log_{1-\alpha} \epsilon^2)$. In this case, $T = O(n \log n)$, the overall time complextiy is $O(n \log n \log \frac{1}{\epsilon})$. For example, when $\alpha = 0.2$, suppose that we set $T = n \log n$, $K \approx 6$, then we can guarantee a relative error $\epsilon = 0.5$ with probability $1 - \frac{1}{n}$. Moreover, from an engineering point of view, we can balance the running time of random walks and power iterations to achieve a better empirical performance. Similar balance strategy was also adopted in previous work [42, 45]. For example, if we have set $T = n \log n$, $K = \log_{1-\alpha} \epsilon^2$ according to the theoretical analysis. However, in practice, the running time of simulating random walks may be longer than running power iterations with this parameter setting. Then, we can adaptively reset T as $\frac{n \log n}{10}$ and $K = \log_{1-\alpha} \frac{\epsilon^2}{10}$. According to Corollary 1, the same error guarantee still holds, while the empirical running time of the algorithm to achieve the same relative error ϵ can be reduced.

The $O(n \log n \log \frac{1}{\epsilon})$ time complexity of PW achieves the best time complexity as the state-of-the-art SpeedPPR algorithm for approximating single-source personalized PageRank query [45]. To our knowledge, only our algorithms and SpeedPPR can achieve the $O(\log \frac{1}{\epsilon})$ complexity. When α becomes smaller, the time of random walk phase grows with $O(\frac{1}{\alpha})$, which is also the same as SpeedPPR. However, in their implementations, the SpeedPPR algorithm involves a very complicated procedure which combines forward push, power iterations and α-random walks. Specifically, it first invokes local push, and uses a queue to maintain active nodes; when the size of the queue exceeds a threshold, it turns into sequential scan (power iteration), and uses dynamic threshold to decide a node to push. The queue threshold and the dynamic threshold need carefully design. Their implementation is also highly optimized with some smart data structures. For comparison, our algorithm is extremely simple, we only need to simulate a number of α-random walks, and then conduct a few basic power iterations on the obtained estimation. The implementation code is within 20 lines. As indicated in our experiments, the performance of PW is better than SpeedPPR with the same relative error guarantee (see Fig. 8).

3.3 Variance Reduction Using Former Samples

Since applying one power iteration can reduce the variance of an estimator by a factor of $(1-\alpha)^2$, it is efficient when α is relatively large (e.g., $\alpha = 0.2$). However, when α is relatively small (e.g., $\alpha = 0.01$), it may require a number of power iterations to achieve a significant variance reduction. To tackle this problem, we propose a novel technique to further reduce the variance by using the historical information of the former samples.

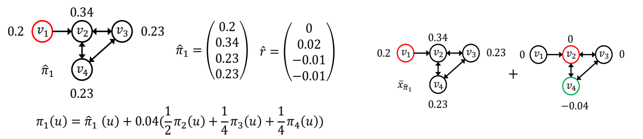
The traditional α-random walk sampling algorithm does not consider the historical information of former samples. That is, we do the same thing when we draw the 1-st sample as when we draw the 1000-th sample. A question is that, can we utilize the former samples to reduce the variance of the estimator? First, we show that a similar invariant equation as Eq. (1) in forward search also holds for any estimated vector $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$.

Lemma 3.12. *Suppose that $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$ is an estimated PageRank vector, and $\hat{\mathbf{r}}$ is the corresponding residual vector, where $\hat{\mathbf{r}} = \boldsymbol{\sigma} + \frac{1-\alpha}{\alpha}\mathbf{P}\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} - \frac{1}{\alpha}\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$. Then, for all $t \in V$, $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(t)$ satisfies the following equation:*

$$\boldsymbol{\pi}_{\boldsymbol{\sigma}}(t) = \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(t) + \sum_{u \in V} \hat{\mathbf{r}}(u) \boldsymbol{\pi}_u(t). \tag{3}$$

Proof. By the definition of PageRank vector $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$, $\boldsymbol{\pi}_{\boldsymbol{\sigma}} = \alpha\boldsymbol{\sigma} + (1-\alpha)P\boldsymbol{\pi}_{\boldsymbol{\sigma}}$. Let $\mathbf{A}_\alpha = \mathbf{I} - (1-\alpha)\mathbf{P}$, $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$ be the solution of the linear system $\mathbf{A}_\alpha \mathbf{x} = \alpha\boldsymbol{\sigma}$. Let $\boldsymbol{\Pi} = (\frac{1}{\alpha}(\mathbf{I} - (1-\alpha)\mathbf{P}))^{-1} = \alpha \mathbf{A}_\alpha^{-1}$, we have $\boldsymbol{\pi}_s = \boldsymbol{\Pi}\boldsymbol{\sigma}$. The t, s-th element (the t-th row, s-th column element) of $\boldsymbol{\Pi}$ equals $\boldsymbol{\pi}_s(t)$. Suppose $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$ is an estimated PageRank vector, since we define the residual vector $\hat{\mathbf{r}}$ as $\hat{\mathbf{r}} = \boldsymbol{\sigma} + \frac{1-\alpha}{\alpha}\mathbf{P}\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} - \frac{1}{\alpha}\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} = \boldsymbol{\sigma} - \frac{1}{\alpha}(\mathbf{I} - (1-\alpha)\mathbf{P})\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} = \boldsymbol{\sigma} - \frac{1}{\alpha}\mathbf{A}_\alpha\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$, we directly have $\boldsymbol{\sigma} = \frac{1}{\alpha}\mathbf{A}_\alpha\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} + \hat{\mathbf{r}}$. Multiplied both sides by $\boldsymbol{\Pi}$, we have $\boldsymbol{\pi}_{\boldsymbol{\sigma}} = \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} + \boldsymbol{\Pi}\hat{\mathbf{r}}$. By expanding the matrix multiplication, we can obtain Eq. (3) for all $t \in V$. □

Note that a key difference between Eq. (1) and Eq. (3) is that the residual values in Eq. (3) can be either positive or negative. Given an estimated PageRank vector $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$, computing the corresponding residual vector is not hard (the time cost is the same as that of performing one power iteration, i.e., $O(m)$). Based on such an invariant formula, we can build a novel estimator conditioned on the residual vector. Given a vector $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$ as the current estimation, let $\hat{\mathbf{e}}$ denote the error vector which is defined as $\hat{\mathbf{e}} = \boldsymbol{\pi}_{\boldsymbol{\sigma}} - \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$. According to Lemma 3.12, $\hat{\mathbf{e}} = \boldsymbol{\Pi}\hat{\mathbf{r}}$ is a linear combination of the residual vector, where the linear coefficient is the personalized PageRank values. Let $|\hat{\mathbf{r}}|$ be a vector where each element $|\hat{\mathbf{r}}|(u) = |\hat{\mathbf{r}}(u)|$ (the absolute value). Note that $\boldsymbol{\Pi}\hat{\mathbf{r}}$ can be estimated using α-random


0.2
v_1
v_2
v_3
v_4
0.34
0.23
0.23
$\hat{\pi}_1$
$\hat{\pi}_1 = (0.2, 0.34, 0.23, 0.23)$
$\hat{r} = (0, 0.02, -0.01, -0.01)$
$\pi_1(u) = \hat{\pi}_1(u) + 0.04(\frac{1}{2}\pi_2(u) + \frac{1}{4}\pi_3(u) + \frac{1}{4}\pi_4(u))$
$\bar{x}_{\hat{\pi}_1}$
+
0
−0.04

(a) Estimation $\hat{\boldsymbol{\pi}}_1$ and the residual vector $\hat{\boldsymbol{r}}$ (b) $\bar{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_1}$

Figure 3: A running example of the estimator $\bar{\mathbf{x}}_{\boldsymbol{\pi}_\sigma}$ on G. (a) Given a current estimation $\hat{\boldsymbol{\pi}}_1 = [0.2, 0.34, 0.23, 0.23]^T$, the corresponding residual vector $\hat{r}$ is $[0, 0.02, -0.01, -0.01]^T$. The invariant $\boldsymbol{\pi}_1(u) = \hat{\boldsymbol{\pi}}_1(u) + 0.04(\frac{1}{2}\boldsymbol{\pi}_2(u) - \frac{1}{4}\boldsymbol{\pi}_3(u) - \frac{1}{4}\boldsymbol{\pi}_4(u))$ holds for $u = v_1, v_2, v_3, v_4$; (b) $\bar{\mathbf{x}}_{\boldsymbol{\pi}_\sigma}$ selects v_2, v_3 and v_4 with probability $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{4}$ to start α-random walks respectively. Suppose that v_2 is chosen, the α-random walk starts from v_2 and stops at v_4, the estimation value of $\bar{\mathbf{x}}_{\hat{\pi}_1}$ is depicted.

walk sampling with a source distribution $\frac{|\hat{\mathbf{r}}|}{\|\hat{\mathbf{r}}\|_1}$. Thus, the error vector $\hat{\mathbf{e}}$ can also be estimated by such an α-random walk sampling procedure. We also define the operator $sgn(x)$ as the sign operator which equals 1 when $x > 0$ and equals -1 when $x \le 0$. Formally, we have the following results.

Lemma 3.13. *Denote by $\hat{\boldsymbol{\pi}}_\sigma$ an estimated PageRank vector, $\hat{\mathbf{r}} = \boldsymbol{\sigma} + \frac{1-\alpha}{\alpha}\mathbf{P}\hat{\boldsymbol{\pi}}_\sigma - \frac{1}{\alpha}\hat{\boldsymbol{\pi}}_\sigma$ is the corresponding residual vector. Suppose that we sample a source node s' from the distribution $\frac{|\hat{\mathbf{r}}|}{\|\hat{\mathbf{r}}\|_1}$, and an α-random walk starts from s' and stops at t. Then, $\bar{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_\sigma} = \hat{\boldsymbol{\pi}}_\sigma + sgn(\hat{\mathbf{r}}(s'))\|\hat{\mathbf{r}}\|_1\mathbf{e}_t$ is an unbiased estimator of $\boldsymbol{\pi}_\sigma$.*

Proof. Let $\tilde{\mathbf{x}}_{s'}$ denote the random vector that if an α-random walk starts from s' and stops at a node t, i.e., $\tilde{\mathbf{x}}_{s'} = sgn(\hat{\mathbf{r}}(s'))\mathbf{e}_t$. We have $E[\tilde{\mathbf{x}}_{s'}] = sgn(\hat{\mathbf{r}}(s'))\boldsymbol{\pi}_{s'}$. By the linearity of expectation, we can derive that $E[\bar{\mathbf{x}}] = \hat{\boldsymbol{\pi}}_\sigma + \sum_{u\in V} Pr[s' = u]\|\hat{\mathbf{r}}\|_1 E[\tilde{\mathbf{x}}_u] = \hat{\boldsymbol{\pi}}_\sigma + \|\hat{\mathbf{r}}\|_1 \boldsymbol{\Pi}(sgn(\hat{\mathbf{r}}(u))\frac{|\hat{\mathbf{r}}|}{\|\hat{\mathbf{r}}\|_1}) = \hat{\boldsymbol{\pi}}_\sigma + \boldsymbol{\Pi}\hat{\mathbf{r}} = \boldsymbol{\pi}_\sigma$. □

Fig. 3 shows a running example of $\bar{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_\sigma}$ on the example graph G in Fig. 2. Suppose that we have obtained a current estimation $\hat{\boldsymbol{\pi}}_1 = [0.2, 0.34, 0.23, 0.23]^T$, which is already near to the exact value of $\boldsymbol{\pi}_1$. We can compute the corresponding residual vector $\hat{\boldsymbol{r}} = [0, 0.02, -0.01, -0.01]^T$. Then, the invariant $\boldsymbol{\pi}_1(u) = \hat{\boldsymbol{\pi}}_1(u) + 0.04(\frac{1}{2}\boldsymbol{\pi}_2(u) - \frac{1}{4}\boldsymbol{\pi}_3(u) - \frac{1}{4}\boldsymbol{\pi}_4(u))$ holds for $u = v_1, v_2, v_3, v_4$. $\bar{\mathbf{x}}_{\boldsymbol{\pi}_1}$ first selects v_2, v_3 and v_4 with probability $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{4}$ respectively, then starts α-random walks from the selected node. Suppose that v_2 is chosen and the α-random walk starts from v_2 and stops at v_4. Then, the estimation value $\bar{\mathbf{x}}_{\boldsymbol{\pi}_1} = \hat{\boldsymbol{\pi}}_1 + [0, 0, 0, -\|\hat{\boldsymbol{r}}\|_1]^T$ ($\|\hat{\boldsymbol{r}}\|_1 = 0.04$). Since $\hat{\boldsymbol{\pi}}_1$ is already close to the exact value, $\|\hat{\boldsymbol{r}}\|_1$ can be very small, which results in a small variance. The novel estimator $\bar{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_\sigma}$ inspires us to design a new algorithm for estimating $\boldsymbol{\pi}_\sigma$. We can first simulate some *burn-up* random walks to obtain a PPR estimation $\hat{\boldsymbol{\pi}}_\sigma$. Then, we compute the corresponding residual vector $\hat{\mathbf{r}}$, and run more random walks by sampling nodes from the normalized residual distribution and starts α-random walks from those nodes. Intuitively, if the estimated PageRank $\hat{\boldsymbol{\pi}}_\sigma$ is close to $\boldsymbol{\pi}_\sigma$, the norm of the correspond residual vector $\hat{\mathbf{r}}$ will be small, which results in a new estimator with a smaller variance. Formally, we have

Lemma 3.14. *Let $\hat{\boldsymbol{\pi}}_\sigma$ be a PPR estimation, $\hat{\mathbf{r}}$ be its corresponding residual vector, then $Var[\bar{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_\sigma}] = \|\hat{\mathbf{r}}\|_1^2(1 - \|\boldsymbol{\Pi}\frac{\hat{\mathbf{r}}}{\|\hat{\mathbf{r}}\|_1}\|_2)$.*

Proof. Let $\tilde{\mathbf{e}}$ be a random vector $\tilde{\mathbf{e}} = \bar{\mathbf{x}} - \hat{\boldsymbol{\pi}}_\sigma$. Then, we have $E[\tilde{\mathbf{e}}] = E[\bar{\mathbf{x}}] - \hat{\boldsymbol{\pi}}_\sigma = \boldsymbol{\Pi}\hat{\mathbf{r}}$. Suppose that we sample a source node s' from the distribution $\frac{|\hat{\mathbf{r}}|}{\|\hat{\mathbf{r}}\|_1}$, and an α-random walk starts from s' and stops at t. Then, $\tilde{\mathbf{e}} = sgn(\hat{\mathbf{r}}(s'))\|\hat{\mathbf{r}}\|_1\mathbf{e}_t$. Let $\mathbf{z} = \boldsymbol{\Pi}\frac{|\hat{\mathbf{r}}|}{\|\hat{\mathbf{r}}\|_1}$, where $\mathbf{z}(t)$ is a random variable that equals 1 if the random walk stops at the node t, and equals 0 otherwise. Then, we have $E[(\mathbf{z}(t))^2] = E[\mathbf{z}(t)] = (\boldsymbol{\Pi}\frac{|\hat{\mathbf{r}}|}{\|\hat{\mathbf{r}}\|_1})(t)$, $E[(\tilde{\mathbf{e}}(t))^2] = E[|\tilde{\mathbf{e}}(t)|^2] = \|\hat{\mathbf{r}}\|_1^2 E[|\tilde{\mathbf{z}}(t)|^2] = \|\hat{\mathbf{r}}\|_1(\boldsymbol{\Pi}|\hat{\mathbf{r}}|)(t)$. And we have the following result: $Var[\tilde{\mathbf{x}}(u)] = Var[\tilde{\mathbf{e}}(u)] = E[(\tilde{\mathbf{e}}(u))^2] - E[\tilde{\mathbf{e}}(u)]^2$. Thus, $Var[\tilde{\mathbf{x}}] = \sum_{u\in V}\tilde{\mathbf{x}}(u) = \sum_{u\in V} E[(\tilde{\mathbf{e}}(u))^2] - \sum_{u\in V} E[\tilde{\mathbf{e}}(u)]^2 = \|\hat{\mathbf{r}}\|_1 \vec{1}^T\boldsymbol{\Pi}|\hat{\mathbf{r}}| - (\boldsymbol{\Pi}\hat{\mathbf{r}})^T(\boldsymbol{\Pi}\hat{\mathbf{r}}) = \|\hat{\mathbf{r}}\|_1^2 - \|\boldsymbol{\Pi}\hat{\mathbf{r}}\|_2^2 = \|\hat{\mathbf{r}}\|_1^2(1 - \|\boldsymbol{\Pi}\frac{\hat{\mathbf{r}}}{\|\hat{\mathbf{r}}\|_1}\|_2^2) \ge 0$. □

Corollary 2. *Let $\boldsymbol{\Pi}$ denote the personalized PageRank matrix. Let $\hat{\boldsymbol{\pi}}_{\sigma 1}$, $\hat{\boldsymbol{\pi}}_{\sigma 2}$ be two PPR estimations with the corresponding residual vector $\hat{\mathbf{r}}_1$, $\hat{\mathbf{r}}_2$. If $\|\hat{\mathbf{r}}_1\|_1^2(1 - \|\boldsymbol{\Pi}\frac{\hat{\mathbf{r}}_1}{\|\hat{\mathbf{r}}_1\|_1}\|_2^2) \le \|\hat{\mathbf{r}}_2\|_1^2(1 - \|\boldsymbol{\Pi}\frac{\hat{\mathbf{r}}_2}{\|\hat{\mathbf{r}}_2\|_1}\|_2^2)$, then $Var[\bar{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_{\sigma 1}}] \le Var[\bar{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_{\sigma 2}}]$. Specifically, compared to the basic α-random walk sampling estimator $\bar{\mathbf{x}}$ where the corresponding residual vector is $\boldsymbol{\sigma}$, $\bar{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_\sigma}$ is an estimator defined above with residual vector $\mathbf{r}$. If $\|\hat{\mathbf{r}}\|_1^2(1 - \|\boldsymbol{\Pi}\frac{\hat{\mathbf{r}}}{\|\hat{\mathbf{r}}\|_1}\|_2^2) \le 1 - \|\boldsymbol{\pi}_\sigma\|_2^2$, then $Var[\bar{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_\sigma}] \le Var[\bar{\mathbf{x}}]$.*

Corollary 2 gives the condition when the variance is reduced: the 1-norm of the residual vector (the sum of all the absolute values of the residuals) is reduced. Based on this, we can develop a progressive α-random walk sampling algorithm to implement such a historical-information-aided variance reduction technique. Moreover, we show that such a progressive α-random walk sampling technique can be easily integrated with our power-iteration technique developed in Section 3.2 to further reduce the variance of the estimator.

The resulting algorithm PPW is outlined in Algorithm 5. The α-random walk samples are divided into several batches and an additional parameter B is used to control the batch size. In each batch, we first compute the residual vector $\hat{\mathbf{r}}$ corresponding to the current estimation (Line 3), then we adapt the source distribution to $\frac{|\hat{\mathbf{r}}|}{\|\hat{\mathbf{r}}\|_1}$, and start α-random walks from nodes sampled from such a distribution (Line 4-7). After that, we perform K power iterations to reduce the variance of the estimator obtained in each batch (Line 8). Note that when the batch size as well as the sample size is set properly, every adaption to the source distribution is expected to reduce the variance. Together with the power iteration technique, we can often obtain a very good estimator for the PPR vector as shown in our experiments.

Algorithm 5: PPW

```
Input: A graph $G$, source distribution $\boldsymbol{\sigma}$, decay parameter $\alpha$, power iteration number $K$,
       sample size $T$, batch size $B$
Output: The estimated PageRank vector $\hat{\boldsymbol{\pi}}_\sigma$
1 $\hat{\boldsymbol{\pi}}_\sigma \leftarrow \mathbf{0}, \mathbf{r} \leftarrow \boldsymbol{\sigma}$;
2 for $i = 1 : B$ do
3     $\hat{\mathbf{r}} \leftarrow \boldsymbol{\sigma} + \frac{1-\alpha}{\alpha}P\hat{\boldsymbol{\pi}}_\sigma - \frac{1}{\alpha}\hat{\boldsymbol{\pi}}_\sigma$;
4     for $i = 1 : \lceil T/B \rceil$ do
5         Sample a node $s'$ from probability distribution $\frac{|\hat{\mathbf{r}}|}{\|\hat{\mathbf{r}}\|_1}$;
6         Run an $\alpha$-random walk from $s'$; suppose that it stops at $u$;
7         $\hat{\boldsymbol{\pi}}_\sigma(u) \leftarrow \hat{\boldsymbol{\pi}}_\sigma(u) + sgn(\hat{\mathbf{r}}(s'))\frac{\|\hat{\mathbf{r}}\|_1}{\lceil T/B \rceil}$;
8     $\hat{\boldsymbol{\pi}}_\sigma \leftarrow \sum_{k=0}^{K-1}\alpha(1-\alpha)^k\mathbf{P}^k\boldsymbol{\sigma} + (1-\alpha)^K\mathbf{P}^K\hat{\boldsymbol{\pi}}_\sigma$;
9 return $\hat{\boldsymbol{\pi}}_\sigma$;
```

Lemma 3.15. *The time complexity of Algorithm 5 is $O(\frac{T}{\alpha} + Km + Bm)$. In practice, B is often treated as a small constant, thus it can be simplified as $O(\frac{T}{\alpha} + Km)$.*

Proof. For a PPR estimation $\hat{\boldsymbol{\pi}}_\sigma$, in order to compute a corresponding residual vector $\hat{\mathbf{r}}$, it only requires propagating fractions to its neighbors and adding the source distribution and subtracting itself, which takes $O(m)$ time. So in each batch, PPW only requires an $O(m)$ additional cost, there are B batches in total, which takes

$O(Bm)$. To implement the weighted sampling process, we can utilize the alias sampling technique [36]. The time complexity of building alias table is bounded by $O(n)$. After that, it only takes $O(1)$ time to draw a sample from a weighted distribution $\frac{|\hat{\mathbf{r}}|}{\|\hat{\mathbf{r}}\|_1}$. So, the total running time of PPW is $O(\frac{T}{\alpha} + Km + Bm)$. In practice, B is often treated as a small constant. As a result, PPW shares the same time complexity with that of PW. □

Lemma 3.16. *Suppose that $\|\hat{r}\|_1 < 1$ in the last batch, Algorithm 5 can achieve an $(\epsilon, \frac{1}{n})$-error with probability higher than $1 - \frac{1}{n}$ in time $O(B \cdot n \log n \log \frac{1}{\epsilon})$. In practice, B is often treated as a small constant, thus it can be simplified as $O(n \log n \log \frac{1}{\epsilon})$.*

Proof. Let X' be the random variable that represents the amount on t in each α-random walk sample. Since $\|\hat{r}\|_1 < 1$ in the last batch in Algorithm 5, we can prove that $X' = \|\hat{\mathbf{r}}\|_1(1-\alpha)^K \mathbf{P}^K \mathbf{e}_{t'}(t) < (1-\alpha)^K \mathbf{P}^K \mathbf{e}_{t'}(t)$. Following the proofs of Lemma 3.11 and Corollary 1, we can also prove that Algorithm 5 can achieve an (ϵ, δ)-error in the last batch. The overall running time of the whole algorithm is $O(B \cdot nlognlog\frac{1}{\epsilon})$. In practice, B is often set as a small constant, it can be simplified as $O(nlognlog\frac{1}{\epsilon})$. □

Lemma 3.16 shows that by properly setting the number of batches B (usually a small constant such as $B = 3$ in practice), the variance of PPW is lower than PW and the same error and time complexity can also be guaranteed as PW (by a similar analysis of Lemma 3.11). Although PPW has the same complexity as SpeedPPR, PPW is easier to implement and also much more accurate than SpeedPPR, as shown in our experiments. Moreover, PPW is based on two totally new and powerful ideas ("first walk, then propagate" and "progressively sampling") which could be of independent interests.

4 NEW SPANNING FORESTS ESTIMATORS

When α is small, the expected length of the α-random walk can be very long, rendering the time cost for drawing a sample very high. To address this problem, Liao et al. [27] developed a spanning forests (SF) sampling technique which was shown to be less sensitive to the parameter α. However, in [27], the variance of the SF based estimators did not analyzed. To fill this gap, we first theoretically analyze the variance of the SF based estimators. Then, we propose two novel techniques to reduce the variances of the SF based estimators.

4.1 Variance Analysis of SF Based Estimators

First, we show that the SF based estimators developed for single-source PPR query [27] (Lemma 2.3 and Lemma 2.4) can be easily extended for any source distribution $\boldsymbol{\sigma}$. Let $F \in \mathcal{F}$ be a random rooted spanning forest and $\rho(F)$ be the root set of F. Suppose that F is sampled with probability $P(F) \propto \prod_{u \in \rho(F)} \frac{\alpha}{1-\alpha} d_{out}(u)$. Denote by $V_{\rho[t]}$ the connected component that t belongs to. Then, we have the following results.

Lemma 4.1. *Let $\tilde{\mathbf{x}}$ be a vector of random variables that is defined as $\tilde{\mathbf{x}} = \sum_{t \in \rho(F)} \sum_{u \in V_{\rho[t]}} \boldsymbol{\sigma}(u)\mathbf{e}_t$. Then, $\tilde{\mathbf{x}}$ is an unbiased estimator of $\boldsymbol{\pi_\sigma}$, i.e., $E[\tilde{\mathbf{x}}] = \boldsymbol{\pi_\sigma}$. When the graph is undirected, we further define a vector of random variables $\dot{\mathbf{x}}$ as*

$$\dot{\mathbf{x}} = \sum_{u \in V} \frac{\sum_{v \in V_{\rho[u]}} \boldsymbol{\sigma}(v)}{\sum_{v \in V_{\rho[u]}} d(v)} d(u)\mathbf{e}_u.$$

Then, $\dot{\mathbf{x}}$ is also an unbiased estimator of $\boldsymbol{\pi_\sigma}$, i.e., $E[\dot{\mathbf{x}}] = \boldsymbol{\pi_\sigma}$.

Proof. By the linearity of expectation, since $\boldsymbol{\pi_\sigma} = \sum_{u \in V} \boldsymbol{\sigma}(u)\boldsymbol{\pi}_u$, $E[\tilde{\mathbf{x}}] = \sum_{u \in V} \boldsymbol{\sigma}(u)\boldsymbol{\pi}_u = \boldsymbol{\pi_\sigma}$, $E[\dot{\mathbf{x}}] = \sum_{u \in V} \boldsymbol{\sigma}(u)\boldsymbol{\pi}_u = \boldsymbol{\pi_\sigma}$. □

As indicated in [27], the estimator $\dot{\mathbf{x}}$ has a smaller variance compared to $\dot{\mathbf{x}}$. Below, we derive the variances of these two estimators ([27] did not explicitly give the variances of these two estimators).

Lemma 4.2. *Let $\mathbf{Q}$ denote the co-occurrence probability matrix where $(\mathbf{Q})_{ii} = 1$ for each node $i \in V$, and $(\mathbf{Q})_{ij}$ equals the probability that i and j have the same root in a rooted spanning forest F sampled with probability $P(F) \propto \prod_{u \in \rho(F)} \frac{\alpha}{1-\alpha} d_{out}(u)$. Then, we have*

$$Var[\tilde{\mathbf{x}}] = \boldsymbol{\sigma}^T \mathbf{Q} \boldsymbol{\sigma} - \|\boldsymbol{\pi_\sigma}\|_2^2.$$

Proof. Suppose that we have sampled a rooted spanning forest with probability $P(F) \propto \prod_{u \in \rho(F)} \frac{\alpha}{1-\alpha} d_{out}(u)$. Given a specific F, each node is rooted in one node. If a node i is rooted in j, we say $\rho[i] = j$. Let $\mathbf{S}$ denote the matrix that $(\mathbf{S})_{ij} = 1$ if j is rooted in i, i.e. $\rho[j] = i$, $(\mathbf{S})_{ij} = 0$ otherwise. Then $\mathbf{S}$ can be written as:

$$\mathbf{S} = [\mathbf{e}_{\rho[1]}, \mathbf{e}_{\rho[2]}, \cdots, \mathbf{e}_{\rho[n]}].$$

According to this definition, we have $\tilde{\mathbf{x}} = \mathbf{S}\boldsymbol{\sigma}$. To derive the variance, we need to compute $E[(\tilde{\mathbf{x}}(u))^2] = E[(\mathbf{e}_u^T \mathbf{S}\boldsymbol{\sigma})^T(\mathbf{e}_u^T \mathbf{S}\boldsymbol{\sigma})] = E[\boldsymbol{\sigma}^T \mathbf{S}^T \mathbf{e}_u \mathbf{e}_u^T \mathbf{S}\boldsymbol{\sigma}]$ for each node u. Also, $E[\tilde{\mathbf{x}}(u)]^2 = (\boldsymbol{\pi_\sigma}(u))^2$. $Var[\tilde{\mathbf{x}}] = \sum_{u \in V} Var[\tilde{\mathbf{x}}(u)] = \sum_{u \in V} E[(\tilde{\mathbf{x}}(u))^2] - \sum_{u \in V} E[\tilde{\mathbf{x}}(u)]^2 = E[\boldsymbol{\sigma}^T \mathbf{S}^T \mathbf{S}\boldsymbol{\sigma}] - \|\boldsymbol{\pi_\sigma}\|_2^2 = \boldsymbol{\sigma}^T E[\mathbf{S}^T \mathbf{S}]\boldsymbol{\sigma} - \|\boldsymbol{\pi_\sigma}\|_2^2$. It remains to analyze $E[\mathbf{S}^T\mathbf{S}]$. Note that $(\mathbf{S}^T\mathbf{S})_{ij} = \mathbf{e}_{\rho[i]}^T \mathbf{e}_{\rho[j]}$, and $\mathbf{S}^T\mathbf{S}$ can be seen as a "co-root" matrix. That is, $(\mathbf{S}^T\mathbf{S})_{ij} = 1$ if i and j has the same root, 0 otherwise. Thus, $E[\mathbf{S}^T\mathbf{S}] = \mathbf{Q}$, the lemma follows. □

If the underlying graph is undirected, by a similar analysis, we can derive an upper bound of the variance of $\dot{\mathbf{x}}$. Specifically, let $(q_d)_{ij}$ be a random variable defined as: if node i and node j belong to the same connected component of a random spanning forest F sampled with probability $P(F) \propto \prod_{u \in \rho(F)} \frac{\alpha}{1-\alpha} d(u)$, then $(q_d)_{ij} = \frac{\sum_{u \in V_{\rho[i]}} (d(u))^2}{(\sum_{u \in V_{\rho[i]}} d(u))^2}$, and $(q_d)_{ij} = 0$ otherwise. Then, we have the following result.

Lemma 4.3. *Let $\mathbf{Q}_d$ be a matrix where $(\mathbf{Q}_d)_{ij}$ is the expectation of the random variable $(q_d)_{ij}$. Then, we have $Var[\dot{\mathbf{x}}] = \boldsymbol{\sigma}^T \mathbf{Q}_d \boldsymbol{\sigma} - \|\boldsymbol{\pi_\sigma}\|_2^2$, and $Var[\dot{\mathbf{x}}] < Var[\tilde{\mathbf{x}}]$.*

Proof. Let F be a rooted spanning forest sampled with probability $P(F) \propto \prod_{u \in \rho(F)} \frac{\alpha}{1-\alpha} d(u)$. If node i is rooted in j, we say $\rho[i] = j$. Let $\tilde{\mathbf{S}}$ be the matrix that $(\tilde{\mathbf{S}})_{ij} = \frac{d(i)}{\sum_{u \in V_{\rho[i]}} d(u)}$ if i and j are in the same connected component, and $(\tilde{\mathbf{S}})_{ij} = 0$ otherwise. Clearly, by this definition, we have $\dot{\mathbf{x}} = \tilde{\mathbf{S}}\boldsymbol{\sigma}$. To derive the variance $Var[\dot{\mathbf{x}}]$, we need to compute $E[(\dot{\mathbf{x}}(u))^2] = E[(\mathbf{e}_u^T \tilde{\mathbf{S}}\boldsymbol{\sigma})^T(\mathbf{e}_u^T \tilde{\mathbf{S}}\boldsymbol{\sigma})] = E[\boldsymbol{\sigma}^T \tilde{\mathbf{S}}^T \mathbf{e}_u \mathbf{e}_u^T \tilde{\mathbf{S}}\boldsymbol{\sigma}]$ for each node u. Also, we have $E[\dot{\mathbf{x}}(u)]^2 = (\boldsymbol{\pi_\sigma}(u))^2$. $Var[\dot{\mathbf{x}}] = \sum_{u \in V} Var[\dot{\mathbf{x}}(u)] = \sum_{u \in V} E[(\dot{\mathbf{x}}(u))^2] - \sum_{u \in V} E[\dot{\mathbf{x}}(u)]^2 = E[\boldsymbol{\sigma}^T \tilde{\mathbf{S}}^T \tilde{\mathbf{S}}\boldsymbol{\sigma}] - \|\boldsymbol{\pi_\sigma}\|_2^2 = \boldsymbol{\sigma}^T E[\tilde{\mathbf{S}}^T \tilde{\mathbf{S}}]\boldsymbol{\sigma} - \|\boldsymbol{\pi_\sigma}\|_2^2$. It remains to analyze $E[\tilde{\mathbf{S}}^T\tilde{\mathbf{S}}]$. Note that $(\tilde{\mathbf{S}}^T\tilde{\mathbf{S}})_{ij} = \frac{\sum_{u \in V_{\rho[i]}} (d(u))^2}{(\sum_{u \in V_{\rho[i]}} d(u))^2}$ if i and j has the same root (they are in the same connected component), and $(\tilde{\mathbf{S}}^T\tilde{\mathbf{S}})_{ij} = 0$ otherwise. Thus, we have $E[\tilde{\mathbf{S}}^T\tilde{\mathbf{S}}] = \mathbf{Q}_d$, and thereby $Var[\dot{\mathbf{x}}] = \boldsymbol{\sigma}^T \mathbf{Q}_d \boldsymbol{\sigma} - \|\boldsymbol{\pi_\sigma}\|_2^2$. Since $\frac{\sum_{u \in V_{\rho[i]}} (d(u))^2}{(\sum_{u \in V_{\rho[i]}} d(u))^2} \leq 1$, we can easily derive that $(\mathbf{Q}_d)_{ij} \leq (\mathbf{Q})_{ij}$ for arbitrary i and j; the equality holds only when all connected components of the spanning forests have only one node, thus the variance inequality follows. □

According to Lemma 4.2 and Lemma 4.3, the variance of the SF based estimators is not only related to the objective distribution $\boldsymbol{\pi_\sigma}$, but also depends on the source distribution $\boldsymbol{\sigma}$. Note that the expected time complexity of sampling an α-random walk is $\tau_{walk} = \frac{1}{\alpha}$, while the expected time complexity of sampling a rooted spanning forest is $\tau_{forest} = \frac{1}{\alpha}\sum_{u\in V}\pi(u,u)$. A direct result is that $\tau_{forest} < n\tau_{walk}$. That is, the time for sampling one spanning forest is lower than sampling n α-random walks. When α is small, the margin is larger. If $\boldsymbol{\sigma}$ is a one-hot distribution, i.e. $\mathbf{e}_s$, $Var[\tilde{\mathbf{x}}] = (\mathbf{Q})_{ss} - \|\boldsymbol{\pi_\sigma}\|_2^2 = 1 - \|\boldsymbol{\pi_\sigma}\|_2^2$, which is exactly equal to the variance of the α-random walk based estimator $\bar{\mathbf{x}}$. However, sampling one α-random walk ($\frac{1}{\alpha}$) is obviously faster than sampling one spanning forest ($\frac{1}{\alpha}\sum_{u\in V}\pi(u,u)$). Therefore, there is no advantage to use SF based estimator when $\boldsymbol{\sigma}$ is a one-hot distribution. However, when $\boldsymbol{\sigma}$ is a "balanced" distribution (i.e. $\|\boldsymbol{\sigma}\|$ gets smaller), the variance of SF based estimator will become smaller while the variance of α-random walk based estimator remains the same. Below, we analyze the special case when $\boldsymbol{\sigma} = \frac{\vec{1}}{n}$.

Variance analysis for the special case. When $\boldsymbol{\sigma} = \frac{\vec{1}}{n}$, we have $Var[\tilde{\mathbf{x}}] = \frac{1}{n^2}\vec{1}^T\mathbf{Q}\vec{1} - \|\boldsymbol{\pi_\sigma}\|_2^2$. Note that the quantity $\frac{1}{n}\vec{1}^T\mathbf{Q}\vec{1} = \frac{1}{n}\sum_{u\in V}\sum_{v\in V}p^r(u,v) = \frac{1}{n}\sum_{u\in V}n_u$, where $p^r(u,v)$ is the probability that u and v have the same root, and n_u is the expected number of nodes in the connected component that u belongs to. Denote by $n_r = \frac{1}{n}\vec{1}^T\mathbf{Q}\vec{1}$ and by $n_{rd} = \frac{\vec{1}^T\mathbf{Q}_d\vec{1}}{n}$. Then, we have the following results.

Lemma 4.4. *If* $n_r < \frac{\tau_{forest}}{n\tau_{walk}} + (n - \frac{\tau_{forest}}{n\tau_{walk}})\|\boldsymbol{\pi_\sigma}\|_2^2 = n\|\boldsymbol{\pi_\sigma}\|_2^2 + \frac{\tau_{forest}}{n\tau_{walk}}(1 - \|\boldsymbol{\pi_\sigma}\|_2^2)$*, the estimator* $\tilde{\mathbf{x}}$ *is better than* $\boldsymbol{\pi_\sigma}$*. On undirected graphs, the estimator* $\dot{\mathbf{x}}$ *is strictly better than* $\bar{\mathbf{x}}$*.*

Proof. We can compare two estimators by their variance divided by their corresponding time complexity. Note that by sampling n α-random walks, the resulting variance is $\frac{1}{n} - \frac{\|\boldsymbol{\pi_\sigma}\|_2^2}{n}$, and the total time complexity is $n\tau_{walk} = O(\frac{n}{\alpha})$. According to our analysis, we have $Var[\tilde{\mathbf{x}}] = \frac{n_r}{n} - \|\boldsymbol{\pi_\sigma}\|_2^2$, and the time complexity of sampling a spanning forest is τ_{forest}. Thus, the condition that $\tilde{\mathbf{x}}$ is better than $\bar{\mathbf{x}}$ is that $\frac{\frac{n_r}{n} - \|\boldsymbol{\pi_\sigma}\|_2^2}{\tau_{forest}} < \frac{\frac{1}{n} - \frac{\|\boldsymbol{\pi_\sigma}\|_2^2}{n}}{n\tau_{walk}}$. Next, we analyze the variance$Var[\dot{\mathbf{x}}] = \frac{n_{rd}}{n} - \|\boldsymbol{\pi_\sigma}\|_2^2$. Note that $\vec{1}^T\mathbf{Q}_d\vec{1} = E[\tilde{\mathbf{S}}^T\tilde{\mathbf{S}}]$ is the sum of all elements of $\mathbf{Q}_d$. In an arbitrary spanning forest, we have $\vec{1}^T\tilde{\mathbf{S}}^T\tilde{\mathbf{S}}\vec{1} = \sum_{i\in V}\frac{|V_{\rho[i]}|(\sum_{u\in V_{\rho[i]}}(d(u))^2)}{(\sum_{u\in V_{\rho[i]}}d(u))^2}$. According to the Cauchy-Schwartz inequality, $|V_{\rho[i]}|(\sum_{u\in V_{\rho[i]}}(d(u))^2) \le (\sum_{u\in V_{\rho[i]}}d(u))^2$ holds. Thus, we have $\vec{1}^T\tilde{\mathbf{S}}^T\tilde{\mathbf{S}}\vec{1} < 1$ and $n_{rd} < 1$. The variance of $\dot{\mathbf{x}}$ satisfies $Var[\dot{\mathbf{x}}] < \frac{1}{n} - \|\boldsymbol{\pi_\sigma}\|_2^2 < \frac{1}{n} - \frac{\|\boldsymbol{\pi_\sigma}\|_2^2}{n}$, and $\tau_{forest} < n\tau_{walk}$, thus $\dot{\mathbf{x}}$ is better than the α-random walk based estimator $\bar{\mathbf{x}}$. ◻

Based on the SF-based estimators, we can easily devise two basic Monte Carlo algorithms MCF and MCFV (using the estimator $\dot{\mathbf{x}}$ for undirected graphs), which are shown in Algorithm 6. Note that by our analysis, Algorithm 6 is ineffective when the source distribution is "unbalanced" ($\|\boldsymbol{\sigma}\|_2$ is large) and it is more suitable for a "balanced" distribution. Note that in [27], a forward push procedure is applied firstly before using the SF-based estimators, which ensures that the desired distribution is relatively "balanced", thus resulting in good performance in practice. Our analysis presented here provides a formal explanation for the cases under which SF-based estimators perform better.

Algorithm 6: MCF (MCFV)

```
Input: A graph $G$, source distribution $\boldsymbol{\sigma}$, decay parameter $\alpha$, sample size $T$
Output: The estimated PageRank vector $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$
$\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} \leftarrow \mathbf{0}, \hat{\mathbf{r}} \leftarrow \boldsymbol{\sigma}$;
for $\omega = 1 : T$ do
    $root \leftarrow$ Loop-earsed $\alpha$-random walk sampling $(G, \alpha)$;
    for $i = 1 : n$ do
        if $G$ is undirected then
            $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(i) \leftarrow \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(i) + \frac{\sum_{v\in V_{\rho[i]}}\hat{\boldsymbol{r}}(v)}{\sum_{v\in V_{\rho[i]}}d(v)}\frac{d(i)}{T}$;
    else
        $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(root(i)) \leftarrow \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(root(i)) + \frac{\hat{\boldsymbol{r}}(i)}{T}$;
return $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$;
```

4.2 Reducing Variance by Power Iterations

Here we develop a power iteration approach to reduce the variances of SF-based estimators. Since $\tilde{\mathbf{x}}$ is an unbiased estimator of $\boldsymbol{\pi_\sigma}$, we know that $\alpha\boldsymbol{\sigma} + (1-\alpha)\mathbf{P}\tilde{\mathbf{x}}$ is also an unbiased estimator of $\boldsymbol{\pi_\sigma}$. Let $F \in \mathcal{F}$ be a random rooted spanning forest and $\rho(F)$ be the root set of F. Denote by t is a root node and by V_t the connected component that t belongs to. Then, we have the following result.

Lemma 4.5. *Let* $\tilde{\mathbf{x}}^{\langle 1\rangle}$ *be a vector of random variables that* $\tilde{\mathbf{x}}^{\langle 1\rangle} = \alpha\boldsymbol{\sigma} + (1-\alpha)\sum_{t\in\rho(F)}(\sum_{w\in N^{out}(u)}(\sum_{u\in V_t}\boldsymbol{\sigma}(u)/d_{out}(t))\mathbf{e}_w)$*. Then,* $\tilde{\mathbf{x}}^{\langle 1\rangle}$ *is an unbiased estimator of* $\boldsymbol{\pi_\sigma}$*, i.e.,* $E[\tilde{\mathbf{x}}^{\langle 1\rangle}] = \boldsymbol{\pi_\sigma}$*. On undirected graphs, we define* $\dot{\mathbf{x}}^{\langle 1\rangle}$ *as*

$$\dot{\mathbf{x}}^{\langle 1\rangle} = \alpha\boldsymbol{\sigma} + (1-\alpha)\sum_{u\in V}\sum_{w\in N(u)}\frac{\sum_{v\in V_{\rho[u]}}\boldsymbol{\sigma}(v)}{\sum_{v\in V_{\rho[u]}}d(v)}\mathbf{e}_w.$$

Then, $\dot{\mathbf{x}}^{\langle 1\rangle}$ *is an unbiased estimator of* $\boldsymbol{\pi_\sigma}$*, i.e.,* $E[\dot{\mathbf{x}}^{\langle 1\rangle}] = \boldsymbol{\pi_\sigma}$*.*

Proof. By the linearity of expectation, $E[\tilde{\mathbf{x}}^{\langle 1\rangle}] = \alpha\boldsymbol{\sigma} + (1-\alpha)\mathbf{P}E[\tilde{\mathbf{x}}] = \alpha\boldsymbol{\sigma} + (1-\alpha)\mathbf{P}\tilde{\mathbf{x}} = \boldsymbol{\pi_\sigma}$. ◻

Intuitively, after sampling a rooted spanning forest, the estimator $\tilde{\mathbf{x}}^{\langle 1\rangle}$ updates all the out-neighbors of roots instead of only updating the roots. Also, the updates of $\dot{\mathbf{x}}^{\langle 1\rangle}$ on u not only considers the information within $V_{\rho[u]}$, but also the information of the neighbors of nodes in $V_{\rho[u]}$. By Lemma 3.4, the variance of the resulting estimator is $(1-\alpha)^2$ times smaller than the original estimator. According to Lemma 3.6, this can be further extended to K power iterations.

Lemma 4.6. *For any* $K > 0$, $\tilde{\mathbf{x}}^{\langle K\rangle} = \sum_{k=0}^{K-1}\alpha(1-\alpha)^k\mathbf{P}^k\boldsymbol{\sigma} + (1-\alpha)^K\mathbf{P}^K\tilde{\mathbf{x}}$ *is an unbiased estimator of* $\boldsymbol{\pi_\sigma}$*. If the underlying graph is undirected,* $\dot{\mathbf{x}}^{\langle K\rangle} = \sum_{k=0}^{K-1}\alpha(1-\alpha)^k\mathbf{P}^k\boldsymbol{\sigma} + (1-\alpha)^K\mathbf{P}^K\dot{\mathbf{x}}$ *is also an unbiased estimator of* $\boldsymbol{\pi_\sigma}$*.*

Lemma 4.7. $Var[\tilde{\mathbf{x}}^{\langle K\rangle}] \le (1-\alpha)^{2K}(\boldsymbol{\sigma}^T\mathbf{Q}\boldsymbol{\sigma} - \|\boldsymbol{\pi_\sigma}\|_2^2)$.

Proof. By iteratively applying the results in Lemma 3.4, we can obtain that the variance is $(1-\alpha)^{2K}$ times smaller than the original estimator, which indicates that $Var[\tilde{\mathbf{x}}^{\langle 1\rangle}] \le (1-\alpha)^{2K}(\boldsymbol{\sigma}^T\mathbf{Q}\boldsymbol{\sigma} - \|\boldsymbol{\pi_\sigma}\|_2^2)$. ◻

By a similar analysis, we can derive that:

Lemma 4.8. $Var[\dot{\mathbf{x}}^{\langle 1\rangle}] \le (1-\alpha)^{2K}(\boldsymbol{\sigma}^T\mathbf{Q}_d\boldsymbol{\sigma} - \|\boldsymbol{\pi_\sigma}\|_2^2)$.

Similar to the results shown in Lemma 3.8, by the linearity property of PPR vector, applying a power iteration at each sample is identical to applying a power iteration after all samples have been

Algorithm 7: PF (PFV)

```
Input: A graph $G$, source distribution $\boldsymbol{\sigma}$, decay parameter $\alpha$, power iteration number $K$,
       sample size $T$
Output: The estimated PageRank vector $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$
1  $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} \leftarrow \mathbf{0}, \hat{\boldsymbol{r}} \leftarrow \boldsymbol{\sigma}$;
2  for $\omega = 1 : T$ do
3      $root \leftarrow$ Loop-earsed $\alpha$-random walk sampling $(G, \alpha)$;
4      for $i = 1 : n$ do
5          if $G$ is undirected then
6              $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(i) \leftarrow \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(i) + \frac{\sum_{v \in V_{\rho[i]}} \hat{\boldsymbol{r}}(v)}{\sum_{v \in V_{\rho[i]}} d(v)} \frac{d(i)}{T}$;
7      else
8          $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(root(i)) \leftarrow \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(root(i)) + \frac{\hat{\boldsymbol{r}}(i)}{T}$;
9  $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} \leftarrow \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} + (1-\alpha)^K \mathbf{P}^K \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$;
10 return $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$;
```

drawn. Thus, we can first perform spanning forest sampling and then use K power iterations to reduce the variance of the estimator. The resulting algorithm PF is outlined in Algorithm 7. It is easy to derive that the time complexity of Algorithm 7 is $O(T'\tau + Km)$, where T' is the sample size and τ denotes the time for sampling a spanning forest.

4.3 Reducing Variance Using Former Samples

Similar to the α-random walk based estimator, we can also use the historical information of the former samples to reduce the variances of the SF-based estimators. The basis idea is that for any existing estimation $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$ obtained by former samples, the corresponding residual vector $\hat{\mathbf{r}}$ can be computed, based on which we can derive new SF-based estimators.

Specifically, given that $\hat{\mathbf{r}} = \boldsymbol{\sigma} + \frac{1-\alpha}{\alpha} P \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} - \frac{1}{\alpha} \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$, we have $\boldsymbol{\pi}_{\boldsymbol{\sigma}} = \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} + \mathbf{\Pi}\hat{\mathbf{r}}$. It remains to use $\tilde{\mathbf{x}}$ and $\dot{\mathbf{x}}$ to estimate $\mathbf{\Pi}\hat{\mathbf{r}}$. This leads to new SF-based estimators.

Let F be a rooted spanning forest sampled with probability $P(F) \propto \prod_{u \in \rho(F)} \frac{\alpha}{1-\alpha} d_{out}(u)$ and $\rho(F)$ be the root set of F. For a root $t \in \rho(F)$, let V_t denote the connected component that t belongs to. Then, we have the following results.

Lemma 4.9. $\tilde{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}} = \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} + \sum_{t \in \rho(F)} \sum_{u \in V_t} \hat{\mathbf{r}}(u) \mathbf{e}_t$ *is an unbiased estimator of* $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$.

Proof. By Lemma 2.3 and the linearity of expectation, we have $E[\tilde{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}}] = \boldsymbol{\pi}_{\boldsymbol{\sigma}}$, since $\boldsymbol{\pi}_{\boldsymbol{\sigma}} = \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} + \mathbf{\Pi}\hat{\mathbf{r}}$. □

Lemma 4.10. $Var[\tilde{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}}] = \hat{\mathbf{r}}^T \mathbf{Q} \hat{\mathbf{r}} - \|\mathbf{\Pi}\hat{\mathbf{r}}\|_2^2$.

Proof. Similar to the proof of Lemma 4.2, it is easy to derive that $Var[\tilde{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}}] = \hat{\mathbf{r}}^T \mathbf{Q} \hat{\mathbf{r}} - \|\mathbf{\Pi}\hat{\mathbf{r}}\|_2^2$. □

Suppose that $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}1}$ and $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}2}$ are two estimations of $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$, $\hat{\mathbf{r}}_1$ and $\hat{\mathbf{r}}_2$ are the corresponding residual vectors. Define the $\mathbf{Q}$-norm of a vector $\mathbf{x}$ as $\|\mathbf{x}\|_{\mathbf{Q}} = \mathbf{x}^T \mathbf{Q} \mathbf{x}$. Then, we have the following corollary.

Corollary 3. *If* $\hat{\mathbf{r}}_1$, $\hat{\mathbf{r}}_2$ *satisfy* $\|\hat{\mathbf{r}}_1\|_1^2(\|\frac{\hat{\mathbf{r}}_1}{\|\hat{\mathbf{r}}_1\|_1}\|_{\mathbf{Q}} - \|\mathbf{\Pi}\frac{\hat{\mathbf{r}}_1}{\|\hat{\mathbf{r}}_1\|_1}\|_2^2) \leq \|\hat{\mathbf{r}}_2\|_1^2(\|\frac{\hat{\mathbf{r}}_2}{\|\hat{\mathbf{r}}_2\|_1}\|_{\mathbf{Q}} - \|\mathbf{\Pi}\frac{\hat{\mathbf{r}}_2}{\|\hat{\mathbf{r}}_2\|_1}\|_2^2)$, $Var[\tilde{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}1}}] \leq Var[\tilde{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}2}}]$.

Corollary 3 gives a formal condition under which the variance can be reduced. According to Corollary 3, since $\frac{\hat{\mathbf{r}}_1}{\|\hat{\mathbf{r}}_1\|_1}$ is a normalized vector, the value of $\|\frac{\hat{\mathbf{r}}_1}{\|\hat{\mathbf{r}}_1\|_1}\|_{\mathbf{Q}} - \|\mathbf{\Pi}\frac{\hat{\mathbf{r}}_1}{\|\hat{\mathbf{r}}_1\|_1}\|_2^2$ is related to the distribution of $\hat{\mathbf{r}}$ and may not be very different for different residual vectors (as indicated in our experiments). As a result, the variance is mainly determined by the $L1$-norm of the residual vector. Since the $L1$-norm of the residual vector often decreases with the sample size increases, we can devise a progressive sampling algorithm (similar to Algorithm 5) to achieve variance reduction.

For undirected graphs, we also have the following lemma.

Lemma 4.11. *Let F be a rooted spanning forest sampled with probability $P(F) \propto \prod_{u \in \rho(F)} \frac{\alpha}{1-\alpha} d(u)$. Denote by $V_{\rho[u]}$ the connected component that u belongs to. Then,*

$$\tilde{\mathbf{x}} = \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} + \sum_{u \in V} \frac{\sum_{v \in V_{\rho[u]}} \hat{\mathbf{r}}(v)}{\sum_{v \in V_{\rho[u]}} d(v)} d(u) \mathbf{e}_u$$

is an unbiased estimator of $\boldsymbol{\pi}_{\boldsymbol{\sigma}}$.

Proof. By Lemma 2.4 and the linearity of expectation, we have $E[\tilde{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}}] = \boldsymbol{\pi}_{\boldsymbol{\sigma}}$, since $\boldsymbol{\pi}_{\boldsymbol{\sigma}} = \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} + \mathbf{\Pi}\hat{\mathbf{r}}$. □

Likewise, we can derive the variance of $\dot{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}}$.

Lemma 4.12. $Var[\dot{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}}] = \hat{\mathbf{r}}^T \mathbf{Q}_d \hat{\mathbf{r}} - \|\mathbf{\Pi}\hat{\mathbf{r}}\|_2^2$.

Proof. Similar to the proof of Lemma 4.3, we can derive that $Var[\tilde{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}}] = \hat{\mathbf{r}}^T \mathbf{Q} \hat{\mathbf{r}} - \|\mathbf{\Pi}\hat{\mathbf{r}}\|_2^2$. □

Let $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}1}$ and $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}2}$ be two estimations of $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$, $\hat{\mathbf{r}}_1$ and $\hat{\mathbf{r}}_2$ be the corresponding residual vectors. Define the $\mathbf{Q}_d$-norm of a vector $\mathbf{x}$ as $\|\mathbf{x}\|_{\mathbf{Q}_d} = \mathbf{x}^T \mathbf{Q}_d \mathbf{x}$. Then, we have the following corollary.

Corollary 4. *If* $\hat{\mathbf{r}}_1$, $\hat{\mathbf{r}}_2$ *satisfy* $\|\hat{\mathbf{r}}_1\|_1^2(\|\frac{\hat{\mathbf{r}}_1}{\|\hat{\mathbf{r}}_1\|_1}\|_{\mathbf{Q}_d} - \|\mathbf{\Pi}\frac{\hat{\mathbf{r}}_1}{\|\hat{\mathbf{r}}_1\|_1}\|_2^2) \leq \|\hat{\mathbf{r}}_2\|_1^2(\|\frac{\hat{\mathbf{r}}_2}{\|\hat{\mathbf{r}}_2\|_1}\|_{\mathbf{Q}_d} - \|\mathbf{\Pi}\frac{\hat{\mathbf{r}}_2}{\|\hat{\mathbf{r}}_2\|_1}\|_2^2)$, $Var[\dot{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}1}}] \leq Var[\dot{\mathbf{x}}_{\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}2}}]$.

Corollary 4 provides a formal condition under which the variance can be reduced. Similar to Corollary 3, the variance of the estimation is mainly determined by the $L1$-norm of the residual vector.

Based on the above results, we can devise a progressive sampling algorithm to implement such a historical information-aided variance reduction technique. The resulting algorithm PPF is given in Algorithm 8. The algorithm has an additional parameter, i.e., the batch size B. In each of B batch, a number of spanning forests are sampled (Line 4-10). Note that in order to achieve a lower variance, it applies K power iterations after sampling in each batch (Line 11), which is identical to use $\tilde{\mathbf{x}}^{\langle K \rangle}$ ($\dot{\mathbf{x}}^{\langle K \rangle}$) for estimation. With a number of samples, the $L1$-norm of the residual vector is expected to be reduced. Then, we construct a new estimator based on the residual vector which is derived from the existing estimation (Line 3). In the next batch, the variance of the estimator is expected to be lower. It is easy to see that the time complexity of PPF is $O(T'\tau + Km)$ since the cost for computing the residual is less than the costs for sampling spanning forests and power iterations. Compared to PF, PPF draws the same number of samples and conducts the same number of power iterations, but it can reduce the variance which is also verified in our experiments.

5 EXPERIMENTS

5.1 Experimental Setup

Datasets. We use 5 real-life datasets in the experiments. The detailed information of these datasets is shown in Table 1. There are 3 undirected graphs Youtube, LiveJournal and Orkut, and 2 directed graphs Pokec and Twitter in Table 1. For undirected graphs, we replace each undirected edge with two directed edges in both directions. These datasets are widely used in previous studies [28, 42, 45] to evaluate PPR computation algorithms. All these datasets can be obtained from [26]. For single-source PPR query, we uniformly

Algorithm 8: PPF (PPFV)

```
Input: A graph $G$, source distribution $\boldsymbol{\sigma}$, decay parameter $\alpha$, power iteration number $K$,
       sample size $T$, batch size $B$
Output: The estimated PageRank vector $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$
$\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} \leftarrow \mathbf{0}, \mathbf{r} \leftarrow \boldsymbol{\sigma}$;
for $i = 1 : B$ do
    $\hat{\mathbf{r}} \leftarrow \boldsymbol{\sigma} + \frac{1-\alpha}{\alpha} P \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} - \frac{1}{\alpha} \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$;
    for $\omega = 1 : \lceil T/B \rceil$ do
        $root \leftarrow$ Loop-eared $\alpha$-random walk sampling $(G, \alpha)$;
        for $i = 1 : n$ do
            if $G$ is undirected then
                $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(i) \leftarrow \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(i) + \frac{\sum_{v \in V_{\rho[i]}} \hat{\boldsymbol{r}}(v)}{\sum_{v \in V_{\rho[i]}} d(v)} \frac{d(i)}{\lceil T/B \rceil}$;
            else
                $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(root(i)) \leftarrow \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(root(i)) + \frac{\hat{\boldsymbol{r}}(i)}{\lceil T/B \rceil}$;
    $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} \leftarrow \sum_{k=0}^{K} \alpha(1-\alpha)^k \mathbf{P}^k \boldsymbol{\sigma} + (1-\alpha)^{K+1} \mathbf{P}^{K+1} \hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$;
return $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$;
```

Table 1: Datasets

Dataset	n	m	m/n	Type
Youtube	1,134,890	2,987,624	2.63	undirected
Pokec	1,632,803	30,622,564	18.75	directed
LiveJournal	4,846,609	42,851,237	8.84	undirected
Orkut	3,072,441	117,185,083	38.14	undirected
Twitter	41,652,230	1,468,365,182	35.25	directed

generate 50 source nodes as the query set (same query set generation methods are also used in previous studies [27, 41, 42]) and report the average time and $L1$-error (Let $\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}$ be an estimated PageRank vector, then the $L1$-error is defined as $\|\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}} - \boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_1 = \sum_{u \in V} |\hat{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(u) - \boldsymbol{\pi}_{\boldsymbol{\sigma}}(u)|$).

Different algorithms. For single-source PPR query, we compare the proposed algorithms with two state-of-the-art algorithms which are SpeedPPR [45] and SpeedL (SpeedLV) [27]. We do not include other algorithms for comparison because all of them are outperformed by these two algorithms [27, 45]. For the proposed algorithms, PW (Power α-random Walk sampling) denotes Algorithm 4 which combines α-random walk sampling and power iterations. PPW (Progressive Power α-random Walk sampling) is Algorithm 5 which improves PW by utilizing former samples. PF (PFV) (Power spanning Forests sampling) is Algorithm 7 which combines spanning forests sampling and power iterations. PPF (PPFV) (Progressive Power spanning Forests sampling) is Algorithm 8 which improves PF by using former samples. For PageRank centrality computation, we use two Monte Carlo algorithms MCW (α-random walk sampling) and MCF (MCFV) (spanning forests sampling) as baselines. We also include a single-source PPR algorithm FORA [42] for comparison, since it was also extended for PageRank centrality computation in [41]. For high-precision PPR algorithms, we include the state-of-the-art algorithm PowerPush [45] for comparison.

Parameters. For PW and PPW, there is a parameter T which controls the sample size of the α-random walk. For a fair comparison, we set T as $n \log n$ by default to make sure that the number of random walk samples is similar to that used in SpeedPPR [45]. For PPW, there is an additional parameter B (the batch size); and we set $B = 3$ by default, as we can achieve very good performance with $B = 3$. For PF (PFV) and PPF (PPFV), there is a parameter T' to control the number of spanning forests samples. We set $T' = T \times \frac{\tau_{forest}}{\tau_{walk}}$ by default to guarantee that the sample size is similar to that of SpeedL (SpeedLV) for a fair comparison. For the parameter K, we set $K = log_{1-\alpha}\epsilon^2$, $\epsilon = 0.5$ by default for PW and PF (PFV), and $K = log_{1-\alpha}\epsilon^2/B$, $\epsilon = 0.5$ for PPW and PPF (PPFV) according to Corollary 1. Since most previous studies [42, 45] use a balance strategy, for all our algorithms, we also adopt such a balance strategy, adaptively setting the parameter K (the number of power iterations) to ensure that the sampling time is roughly equal to the time taken by K-power iterations. Note that with such a balance strategy, the same error guarantee still holds for all our algorithms, as discussed in Section 3.2. We will also study the effect of the parameters B and K of our algorithms. For all the baseline algorithms, we use the default parameter settings as used in their original papers [27, 41, 45].

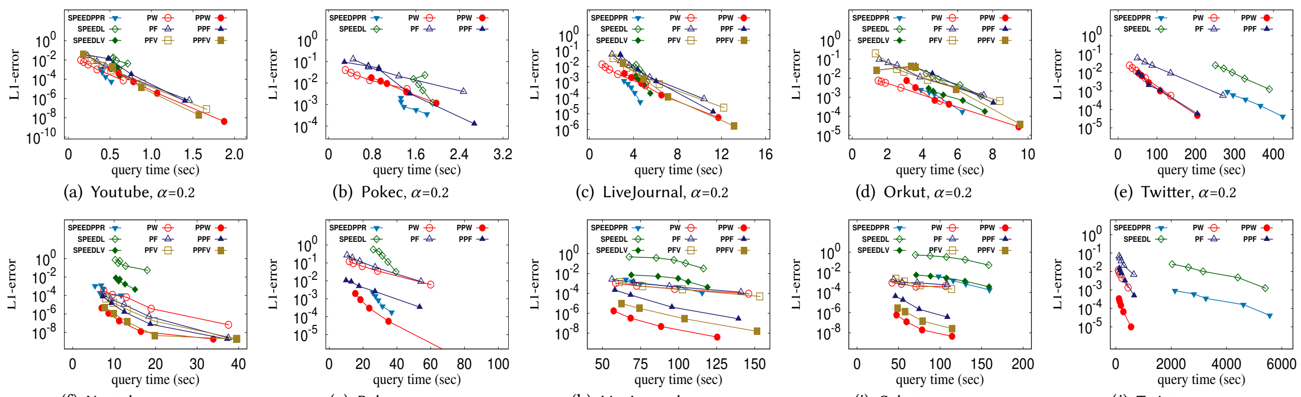
All the experiments are conducted on a Linux 20.04 server with Intel 2.0 GHz CPU and 128GB memory. We obtain the ground-truth PPR results by the power iteration algorithm with an $L1$-error 10^{-16}, which is near to "C++ double precision". All the proposed algorithms are implemented with C++. For SpeedPPR, SpeedL (SpeedLV) and PowerPush, we use the open-source C++ implementations by their original authors [27, 45].

Table 2: Comparison between α-RW and SF based estimators

Dataset	$\boldsymbol{\alpha}$	$n\|\boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_2^2$	$\frac{n\tau_{walk}}{\tau_{forest}}$	$\frac{\vec{1}^T Q \vec{1}}{n}$	$L1$-error $\bar{\mathbf{x}}$	$\tilde{\mathbf{x}}$	$\dot{\mathbf{x}}$
Youtube	0.2	62.69	10.4	891.74	0.18	0.44	**0.028**
	0.01	89.92	190.91	210729	0.16	1.25	**0.003**
Pokec	0.2	2.61	6.37	39.22	**0.19**	0.31	\
	0.01	35.98	29.11	2362.33	**0.13**	0.57	\
LiveJournal	0.2	3.95	8.36	84.23	0.18	0.34	**0.032**
	0.01	7.26	168.83	55341.7	0.16	1.11	**0.12**
Orkut	0.2	2.73	6.69	65.57	0.19	0.41	**0.03**
	0.01	4.12	152.54	36480	0.18	1.18	**0.003**
Twitter	0.2	10.69	3.4	214.87	**0.17**	0.25	\
	0.01	6.4	3.02	345.5	**0.13**	0.27	\

5.2 Comparing α-RW and SF based Estimators

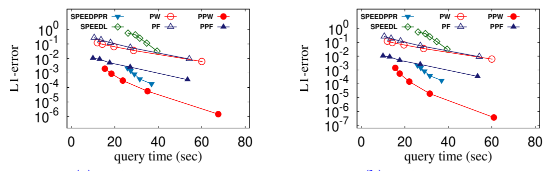
We compare the performance of the α-random walk (α-RW) based estimator $\bar{\mathbf{x}}$ and the spanning forests (SF) based estimators ($\tilde{\mathbf{x}}$ and $\dot{\mathbf{x}}$) to verify the analysis in Section 4.1. Note that as we analyzed in Section 4.1, for the single-source PPR computation where the source distribution $\boldsymbol{\sigma}$ is an one-hot distribution, the α-RW based estimator is clearly better than SF based estimators. However, for the PageRank centrality computation where $\boldsymbol{\sigma} = \frac{\vec{1}}{n}$ is a balanced distribution, the SF based estimators can be better than the α-RW based estimator. Thus, in this experiment, we focus mainly on the problem of PageRank centrality computation. Since the performance of the α-RW and SF based estimators is related to $n_r = \frac{\vec{1}^T Q \vec{1}}{n}$, $n\|\boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_2^2$ and $\frac{n\tau_{walk}}{\tau_{forest}}$ as shown in Lemma 4.4, we perform SF sampling to estimate n_r, $\|\boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_2^2$ and τ_{forest}, and perform α-RW sampling to estimate τ_{walk}. To estimate the PageRank centrality, we simulate $n \log n$ α-random walks for $\bar{\mathbf{x}}$ and we sample spanning forests to make sure that the running time of these two algorithms are almost the same. The results are shown in Table 2. As can be seen, $\bar{\mathbf{x}}$ is always better than $\tilde{\mathbf{x}}$. This is because n_r is often much larger than $n\|\boldsymbol{\pi}_{\boldsymbol{\sigma}}\|_2^2$ on real-world graphs, thus the condition in Lemma 4.4 is hard to meet. The accuracy of $\tilde{\mathbf{x}}$ is closely dependent on n_r; the smaller it is, the higher accuracy the SF based estimator $\tilde{\mathbf{x}}$ achieves. In addition, we can see that the improved SF estimator $\dot{\mathbf{x}}$ substantially outperforms both $\bar{\mathbf{x}}$ and $\tilde{\mathbf{x}}$ on undirected graphs. This is because (1) the variance of $\dot{\mathbf{x}}$ is strictly smaller than both $\bar{\mathbf{x}}$ and $\tilde{\mathbf{x}}$; and (2) the time for sampling n α-random walks is much higher than that for sampling a spanning forest as shown in Table 2 (column 4). These results confirm our theoretical analysis in Section 4.1.



(a) Youtube, α=0.2 (b) Pokec, α=0.2 (c) LiveJournal, α=0.2 (d) Orkut, α=0.2 (e) Twitter, α=0.2

(f) Youtube, α=0.01 (g) Pokec, α=0.01 (h) LiveJournal, α=0.01 (i) Orkut, α=0.01 (j) Twitter, α=0.01

Figure 4: Performance of different algorithms for computing single-source personalized PageRank



(a) Pokec, $B = 7$ (b) Pokec, $B = 9$

Figure 5: Results of computing single-source personalized PageRank on Pokec with different batch size (α = 0.01)

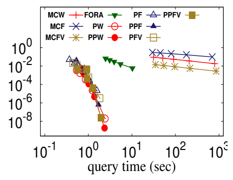
5.3 Results of Single-source PPR Query

In this experiment, we compare the performance of the proposed algorithms (PW, PPW, PF, PFV, PPF, and PPFV) with the state-of-the-art algorithms (SpeedPPR, SpeedL and SpeedLV) for processing single-source PPR query. Fig. 4 plots the $L1$-error as a function of query time for each algorithm on all datasets, with $\alpha = 0.2$ and $\alpha = 0.01$ respectively. We vary T to make sure the number of samples of all approximate algorithms are similar, and vary K to ensure a relatively good performance for all algorithms. Note that for undirected graphs, there are three additional algorithms SpeedLV, PFV and PPFV as shown in Fig. 4, which are based on the improved SF estimator $\dot{\mathbf{x}}$. As a result, there are 9 curves for undirected graphs and only 6 curves for directed graphs. As can be seen, among all our algorithms, PPW achieves the best overall performance, followed by PPF (PPFV), PW, and PF (PFV). We also note that PPW considerably improves PW by the progressive sampling strategy, since the variance is further reduced by using the historical information of former samples. When $\alpha = 0.2$, our algorithms including PPW, PPF (PPFV), and PW can achieve comparable performance as the state-of-the-art algorithms on all datasets. On Figs.4(b-c), although SpeedPPR performs better than our algorithms in some cases, our best algorithm can achieve much lower $L1$-error using a little more time. Moreover, on the largest dataset Twitter, our algorithms (PW, PPW and PPF) are significantly better than SpeedPPR. When $\alpha = 0.01$, however, we can clearly see that our best algorithms including PPW and PPF (PPFV) substantially outperform the state-of-the-art algorithms on most datasets. We also note that on Pokec (Fig. 4(g)), SpeedPPR performs better than our algorithms. This is because the parameter batch size B is not proper in this case. However, when enlarging the default batch size $B = 3$ to $B = 7$, PPW can achieve 10^{-4} $L1$-error using 22 seconds, which is much better than SpeedPPR (40 seconds). The results of different B ($B = 7$ and $B = 9$) on Pokec can be found in Fig. 5. We can clearly see that PPW can outperform SpeedPPR in these paramter settings. Although Pokec is an exception, we find that the default batch size $B = 3$ is proper for most datasets (see Fig. 12). In addition, our best algorithms can obtain extremely accurate results (the $L1$ errors of our best algorithms can even be less than 10^{-8}) on most datasets using very low query time. For example, on the Orkut dataset, PPW, PPFV, and PPF can achieve the $L1$ errors 4.6×10^{-9}, 2.2×10^{-8}, 3.6×10^{-7} using 114, 115 and 114 seconds respectively. However, on the same dataset, SpeedPPR, SpeedL and SpeedLV obtain the $L1$ errors 1.8×10^{-4}, 5.2×10^{-2}, 3.7×10^{-4} using 160, 159 and 159 seconds respectively. These results indicate that the proposed variance reduction techniques are very powerful to improve the performance of Monte Carlo based PPR computation algorithms. Moreover, compared to the state-of-the-art SpeedPPR algorithm, our best algorithm PPW is much easier to implement, thus it is highly recommended for PPR estimation in real-world applications.

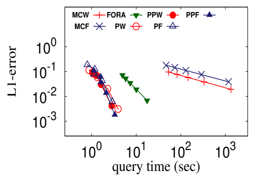
5.4 Results of PageRank Computation

In this experiment, we compare the performance of our algorithms with the baseline algorithms MCW, MCF (MCFV) and FORA for PageRank centrality computation (i.e., $\boldsymbol{\sigma} = \frac{\vec{1}}{n}$). The results are shown in Fig. 6. Since MCFV, PFV and PPFV can only be applied for undirected graphs, there are 10 curves for undirected graphs and only 7 curves for directed graphs. We vary T and K to ensure a good performance of for all algorithms. As can be seen, when $\alpha = 0.2$, all our algorithms achieve comparable performance on most datasets and all of them significantly outperform the state-of-the-art algorithms. To compare MCW, MCF, and MCFV on undirected graphs, we can clearly see that the improved SF based algorithm MCFV is always better than the α-RW based algorithm MCW, because the source distribution $\boldsymbol{\sigma} = \frac{1}{n}$ is a "*balanced*" distribution. This result further confirms our theoretical analysis in Section 4.1.

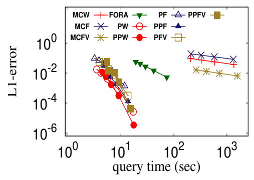
When $\alpha = 0.01$, we can observe that PW and PF (PPF) improves over MCW and MCF (MCFV); and PPW and PPF (PPFV) further improves PW and PF (PFV) respectively (in this case, MCW, MCF and MCFV run out of 24 hours, thus we do not show the results in Fig. 6). Furthermore, our best algorithms PPW and PPF (PPFV) also substantially outperform FORA on all datasets. For example, on Orkut, both PPW and PPFV consume around 200 seconds to compute the PageRank vector with $L1$-error around 10^{-10}. The state-of-the-art algorithm FORA, however, consumes much more time (1200 seconds) but achieves a much larger $L1$-error (10^{-3}). These results demonstrate the high effectiveness of the proposed variance reduction techniques.



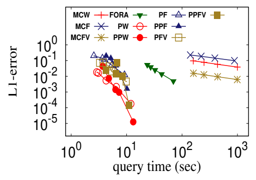
(a) Youtube, α=0.2



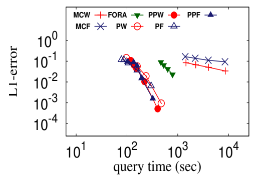
(b) Pokec, α=0.2



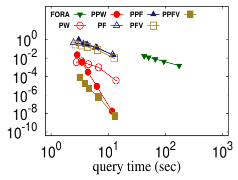
(c) LiveJournal, α=0.2



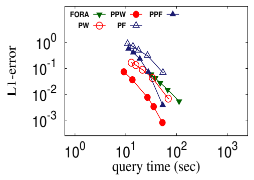
(d) Orkut, α=0.2



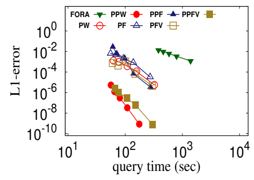
(e) Twitter, α=0.2



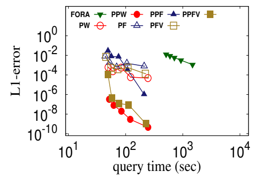
(f) Youtube, α=0.01



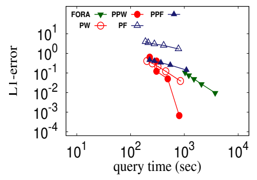
(g) Pokec, α=0.01



(h) LiveJournal, α=0.01

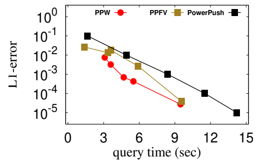


(i) Orkut, α=0.01

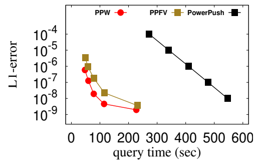


(j) Twitter, α=0.01

Figure 6: Performance of different algorithms for computing PageRank centrality



(a) Orkut, α = 0.2



(b) Orkut, α = 0.01

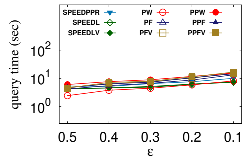
Figure 7: Comparison with high-precision algorithm

5.5 Comparison with High-precision Algorithm

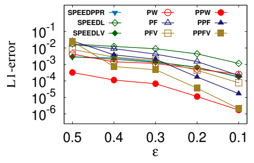
In previous experiments, our algorithms can achieve very low $L1$-errors within a short time. Here we compare the performance of our best algorithms (PPW and PPFV) with the state-of-the-art high precision PPR algorithm PowerPush [45]. The results on Orkut are shown in Fig. 7. For the other datasets, the results are consistent. As can be seen, our algorithms can achieve smaller $L1$-error within shorter time compared to PowerPush. For example, when $\alpha = 0.01$, PowerPush takes around 500 seconds to achieve an $L1$-error 10^{-8} while both PPW and PPFV require less than 150 seconds, which is at least 3× faster. This result indicates that our variance-reduced solutions are very efficient for high-precision PPR computation.

5.6 Results with Relative Error Guarantee

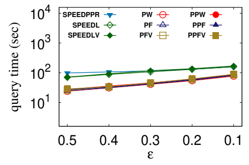
In this experiments, we evaluate the performance of our solutions by varying the relative error ϵ. For the proposed methods, we set parameters to make sure that an (ϵ, δ)-error is satisfied. Specifically, according to Lemma 3.11, PW can guarantee an relative error ϵ by setting the number of random walks T as $n \log n$, and set $K = \log_{1-\alpha} \epsilon^2$. We also adopt the balanced strategy used in [42, 45] for a fair comparison. For PPW, since in each batch, the variance is reduced in PPW, the same error guarantee still holds when $T = n \log n$, $K = \log_{1-\alpha} \epsilon^2$. Additionally, we set $B = 3$. For SF-based algorithms, similar to SpeedL and SpeedLV [27], it is hard to derive an (ϵ, δ)-error bound. Following [27], we set the number of spanning forests to make sure that the Monte-Carlo phase of the algorithm takes similar time with that of α-RW based algorithm SpeedPPR so that they are expected to obtain a similar (ϵ, δ)-error. For the baseline methods FORA [42], SpeedPPR [45] and SpeedL, SpeedLV [27], we set parameters following their original implementations. We compare the running time of different algorithms with varying ϵ from 0.5 to 0.1. We also plot the $L1$-error of various algorithms with varying ϵ. We conduct experiments on both directed and undirected graphs. Fig. 8 and Fig. 9 show the results on



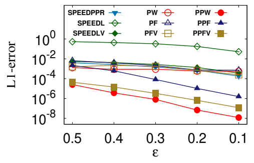
(a) Orkut, time (α = 0.2)



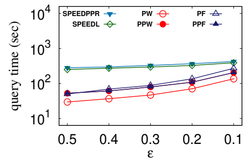
(b) Orkut, error (α = 0.2)



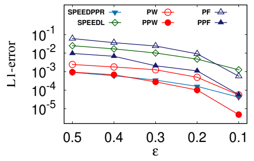
(c) Orkut, time (α = 0.01)



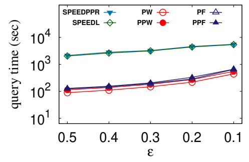
(d) Orkut, error (α = 0.01)



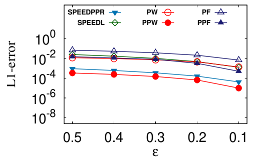
(e) Twitter, time (α = 0.2)



(f) Twitter, error (α = 0.2)



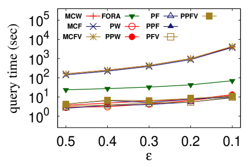
(g) Twitter, time (α = 0.01)



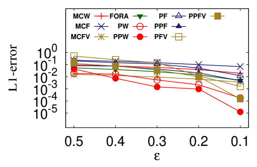
(h) Twitter, error (α = 0.01)

Figure 8: Results of different algorithms with relative error guarantee for single source PPR query

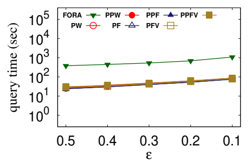
Orkut and Twitter respectively. The results on the other datasets are consistent. It can be seen that when ϵ gets smaller, the running time of all algorithms become longer, and the $L1$-error gets smaller. For single source query, PW is significantly better than SpeedPPR to achieve the same relative error ϵ; PPW takes a little more time than PW, but it can achieve much lower $L1$-error than PW. Moreover, compared to SpeedPPR, PPW can achieve a much lower $L1$-error using less time. For example, when $\alpha = 0.01$ and $\epsilon = 0.1$, PPW is an order of magnitude faster than SpeedPPR. Furthermore, on Orkut, the $L1$-error of PPW is even 4 orders of magnitude smaller than that of SpeedPPR. PF (PPF) is worse than PW (PPW) and PFV (PPFV) is competitive with PW (PPW). Likewise, for PageRank centrality



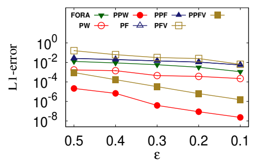
(a) Orkut, time ($\alpha = 0.2$)



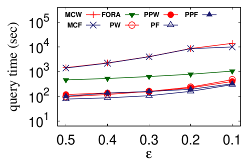
(b) Orkut, error ($\alpha = 0.2$)



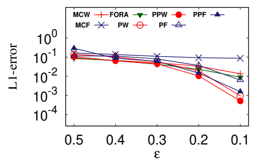
(c) Orkut, time ($\alpha = 0.01$)



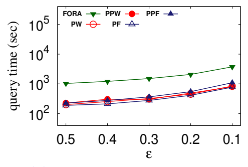
(d) Orkut, error ($\alpha = 0.01$)



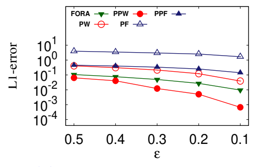
(e) Twitter, time ($\alpha = 0.2$)



(f) Twitter, error ($\alpha = 0.2$)



(g) Twitter, time ($\alpha = 0.01$)



(h) Twitter, error ($\alpha = 0.01$)

Figure 9: Results of different algorithms with relative error guarantee for PageRank centrality computation

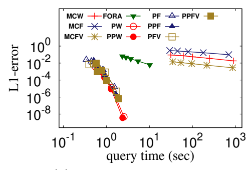
computation, our algorithms are also significantly better than the state-of-the-art algorithm FORA. These results indicate that our variance-reduced techniques are indeed very efficient and effective for PPR computations with relative error guarantees.

5.7 Results of PPR Computation with Random σ

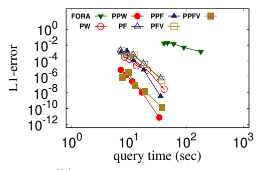
In this experiment, we evaluate our algorithms for PPR computation with a random distribution $\boldsymbol{\sigma}$. Specifically, we draw n uniform random numbers in $[0, 1]$ and normalize the vector to generate a source distribution $\boldsymbol{\sigma}$. We generate 50 such source distributions as the query set and report the average performance. The results on Youtube are shown in Fig. 10. We can also observe similar results on the other datasets. As shown in Fig. 10, the results are very similar to those of the PageRank centrality computation. The proposed algorithms PW, PPW, PF (PFV), PPF (PPFV) significantly outperform the baseline algorithms MCW, MCF (MCFV) and FORA for both $\alpha = 0.2$ and $\alpha = 0.01$. For example, when $\alpha = 0.01$, PPW and PPFV consume around 40 seconds to compute the PPR vector with $L1$-error around 10^{-12} and 10^{-10}. The state-of-the-art algorithm FORA, however, consumes much more time (180 seconds) but achieve a much larger $L1$-error (10^{-3}). These results further confirm the high effectiveness of the proposed solutions.

5.8 Results of Varying Parameters

Here we study the effect of the parameters K and B in our algorithms. Recall that PW, PF (PFV), PPW, and PPF (PPFV) have a parameter K, while PPW and PPF (PPFV) also have an additional parameter B. To study the effect of K, we vary K from 20 to 100 for all algorithms. Fig. 11 shows the results of our algorithms with varying K on Youtube given that $\alpha = 0.01$. We can also observe

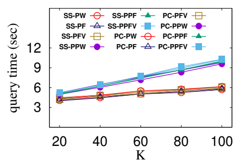


(a) Youtube, $\alpha = 0.2$

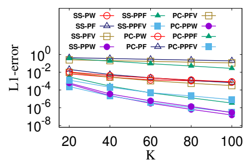


(b) Youtube, $\alpha = 0.01$

Figure 10: Performance of various algorithms for computing PPR with a random source distribution σ

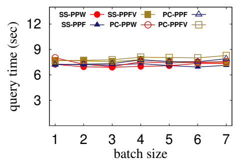


(a) Youtube ($\alpha = 0.01$)

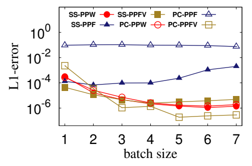


(b) Youtube ($\alpha = 0.01$)

Figure 11: Results of our algorithms with varying K

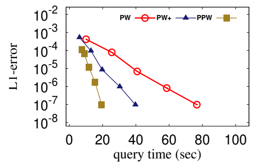


(a) Youtube ($\alpha = 0.01$)

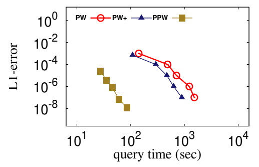


(b) Youtube ($\alpha = 0.01$)

Figure 12: Results of our algorithms with varying B



(a) Orkut ($\alpha = 0.2$)



(b) Orkut ($\alpha = 0.01$)

Figure 13: Results of different implementations of the power iteration procedure

similar results on the other datasets and with other values of α. Note that in Fig. 11, an algorithm with a prefix "SS" (PC) means that it is used to compute the single-source PPR query (PageRank centrality). As expected, when K gets larger, the $L1$-errors of each algorithm become smaller, but its computation time also grows. As a result, in our previous experiments, we adaptively set K so that the time costs for sampling and power iterations are roughly equal, which can achieve good performance in practice.

To investigate the effect of B, we vary B from 1 to 7 for PPW, PPF, and PPFV. The results on Youtube with $\alpha = 0.01$ are shown in Fig. 12. The results on the other datasets and with other α values are consistent. As shown in Fig. 12, the query time of all algorithms is insensitive w.r.t. the parameter B, as the time complexity of our algorithms is not related to B. We can also see that when B increases, the $L1$-errors of most our algorithms decrease first and then increase. This is because when B becomes too large, the number of samples in each batch is not sufficient to make the norm of the residual vector decrease, thus may result in larger variances. As a result, we recommend to set B as a small integer, such as $B = 3$ as used in our previous experiments, for real-world PPR computation applications.

Table 3: Recommendation for different PPR algorithms. "SSQ" is the abbreviation of "single source query", "PRC" is the abbreviation of "PageRank computation".

		directed graph		**undirected graph**	
		large α (e.g. $\alpha = 0.2$)	small α ($\alpha = 0.01$)	large α (e.g. $\alpha = 0.2$)	small α ($\alpha = 0.01$)
SSQ	relative error	PW	PW	PW, PFV	PW, PFV
	$L1$-error	PW, PPW	PPW	PW, PPFV	PPW, PPFV
PRC	relative error	PW	PW	PW, PFV	PW, PFV
	$L1$-error	PPW	PPW	PPW, PPFV	PPW, PPFV

5.9 Comparing Various Power Procedures

Note that in all previous experiments, our algorithms are integrated with a basic power iteration procedure. However, the power iteration procedure can be optimized using the PowerPush algorithm proposed in [45]. Specifically, the optimization techniques in PowerPush include using local updates when the size of active nodes is small, and using dynamic $L1$-error threshold to achieve a desired $L1$-error. We adopt the implementation of PowerPush in our PW algorithm, and the resulting algorithm is denoted by PW+. Note that PPW is very hard to integrate with PowerPush, because we do not know the $L1$-error that we need to achieve in each batch of PPW. Likewise, both PF and PFV can also integrate with PowerPush, but their results are very similar to PW+. To avoid redundancy, we mainly compare the performance of PW, PW+ and PPW in terms of both running time and $L1$-error. Fig. 13 shows the results on Orkut, and similar results can also be observed on the other datasets. As can be seen, PW+ can achieve the same $L1$-error using $2\times$ shorter time than PW, which is consistent with the results reported in [45] where PowerPush is around $2\times$ faster than the basic power iteration method. However, such an improvement is limited. For comparison, PPW can achieve the same $L1$-error in much shorter time ($10\times$ faster than PW when $\alpha = 0.01$). This result indicates that our estimators which utilize historical sample information are much more powerful than the optimized implementation of power iteration used in PowerPush.

5.10 Summary of findings

Here we summarize our main findings. Among all the proposed methods, PW is competitive or better than the state-of-the-art PPR computation algorithms. PPW further significantly improves over PW by utilizing the progressively sampling technique. When the applications only require to guarantee a relative error ϵ, PW is better than all other algorithms. When we want to achieve a lower $L1$-error, PPW is much better than the other algorithms. For SF based algorithms, PF and PPF are outperformed by PW and PPW. On undirected graphs, by utilizing the variance-reduced estimators $\dot{\mathbf{x}}$, PFV and PPFV are competitive to PW and PPW. Based on these results, we make the following recommendations for selecting a PPR algorithm for practical applications in Table 3. For example, for PPR computation on undirected graphs, when α is relatively small (e.g. $\alpha = 0.01$) and we only require a relative error guarantee, we recommend to use PW or PFV. When we want more accurate results, however, we recommend PPW and PPFV.

6 RELATED WORK

Existing algorithms for (personalized) PageRank computation can be classified into two categories: deterministic algorithms and approximate algorithms. Deterministic PageRank algorithms are mainly based on the power method [8, 34, 47] or the forward push method [1, 5, 21]. Many efforts have been made to improve the efficiency of the power method. Notable improved techniques of the power method include block matrix elimination [22, 34], Chebyshev polynomial speedup [8, 9], and core-tree decomposition [33]. There also exist studies [45] to optimize forward push by combining it with the power method. Those deterministic algorithms can achieve a high precision, but they often require considerable time overheads, which is unacceptable for processing online PageRank queries on large graphs.

Among the extensive studies on PageRank computation, ones that are most related to our work are approximate algorithms. Most of them are based on the α-random walk sampling. [2, 30] directly applied the α-random walk sampling to estimate PageRank centrality. Recently, Lofgren et al. [31] proposed a bidirectional method to improve the efficiency of α-random walk sampling by combining it with forward push. Following such a bidirectional technique, several advanced methods [28, 39, 42, 45] were also proposed to further improve the efficiency of the α-random walk sampling method. However, all of these studies focus mainly on optimizing the forward push procedure, and little optimization has been done on the Monte Carlo sampling. More recently, spanning forests based sampling technique was proposed in [27], which combines spanning forests sampling with forward push. Such a spanning forests based sampling technique performs very well on undirected graphs with a small α. However, on directed graphs, the spanning forests based sampling technique is still inefficient when α is small. Moreover, there still lacks a formal comparison between α-random walk sampling based methods and spanning forests sampling based methods. In this paper, we provide a formal comparison between these two Monte Carlo techniques and develop two novel variance reduction techniques to improve these Monte Carlo methods.

In addition, it is worth mentioning that there exist many other studies on the variants of PageRank computation problems, such as single target personalized PageRank query [32, 37, 40], Top-k personalized PageRank query [3, 12–14, 18, 43], PageRank computation on parallel [17, 20, 38] and distributed [16, 29] settings, as well as PageRank computation on dynamic graphs [48–50]. All these studies, however, are orthogonal to our work.

7 CONCLUSION

In this work, we develop two novel techniques to reduce the variances of the Monte Carlo estimators for personalized PageRank (PPR) computation. Specifically, we first show that applying few power iterations on two existing Monte Carlo estimators, including both the α-random walk and the spanning forests based estimators, can substantially reduce the variance of these two estimators. Second, we also develop a progressive sampling technique which utilizes the historical information of the former samples to further reduce the variance of the Monte Carlo estimators. Equipped with these two novel variance reduction techniques, we propose several new and efficient algorithms for PPR computation. Extensive experiments on 5 large real-life datasets show that the newly-proposed algorithms significantly outperform the state-of-the-art bidirectional algorithms, in terms of both computation time and estimation accuracy.

REFERENCES

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

[2] 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.

[3] 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.

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

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

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

[7] 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*.

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

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

[10] Chris H. Q. Ding, Xiaofeng He, Parry Husbands, Hongyuan Zha, and Horst D. Simon. 2003. PageRank: HITS and a Unified Framework for Link Analysis. In *SDM*. 249–253.

[11] François Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens. 2007. Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation. *IEEE Trans. Knowl. Data Eng.* 19, 3 (2007), 355–369.

[12] 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.

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

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

[15] David F. Gleich. 2015. PageRank Beyond the Web. *SIAM Rev.* 57, 3 (2015), 321–363.

[16] Tao Guo, Xin Cao, Gao Cong, Jiaheng Lu, and Xuemin Lin. 2017. Distributed Algorithms on Exact Personalized PageRank. In *SIGMOD*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). 479–494.

[17] Wentian Guo, Yuchen Li, Mo Sha, and Kian-Lee Tan. 2017. Parallel Personalized Pagerank on Dynamic Graphs. *VLDB* 11, 1 (2017), 93–106.

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

[19] Taher H. Haveliwala. 2002. Topic-sensitive PageRank. In *WWW*. 517–526.

[20] Guanhao Hou, Xingguang Chen, Sibo Wang, and Zhewei Wei. 2021. Massively Parallel Algorithms for Personalized PageRank. *VLDB* 14, 9 (2021), 1668–1680.

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

[22] 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.

[23] Heung-Nam Kim and Abdulmotaleb El-Saddik. 2011. Personalized PageRank vectors for tag recommendations: inside FolkRank. In *Proceedings of the 2011 ACM Conference on Recommender Systems*. 45–52.

[24] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*.

[25] Amy Nicole Langville and Carl Dean Meyer. 2006. *Google's PageRank and beyond - the science of search engine rankings*. Princeton University Press.

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

[27] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, and Guoren Wang. 2022. Efficient Personalized PageRank Computation: A Spanning Forests Sampling Based Approach. In *SIGMOD2*. 2048–2061.

[28] 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.

[29] Wenqing Lin. 2019. Distributed Algorithms for Fully Personalized PageRank on Large Graphs. In *WWW*. 1084–1094.

[30] Qin Liu, Zhenguo Li, John C. S. Lui, and Jiefeng Cheng. 2016. PowerWalk: Scalable Personalized PageRank via Random Walks with Vertex-Centric Decomposition. In *CIKM*. 195–204.

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

[32] Peter Lofgren and Ashish Goel. 2013. Personalized PageRank to a Target Node. *CoRR* abs/1304.4658 (2013).

[33] 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.

[34] Kijung Shin, Jinhong Jung, Lee Sael, and U Kang. 2015. BEAR: Block Elimination Approach for Random Walk with Restart on Large Graphs. In *SIGMOD*, Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives (Eds.). 1571–1585.

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

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

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

[38] Runhui Wang, Sibo Wang, and Xiaofang Zhou. 2019. Parallelizing approximate single-source personalized PageRank queries on shared memory. *VLDB* 28, 6 (2019), 923–940.

[39] 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.

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

[41] 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. *TODS* (2019), 18:1–18:37.

[42] 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.

[43] 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.

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

[45] 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.

[46] 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.

[47] 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.

[48] Hongyang Zhang, Peter Lofgren, and Ashish Goel. 2016. Approximate Personalized PageRank on Dynamic Graphs. In *KDD*. ACM, 1315–1324.

[49] Junchao Zhang, Junjie Chen, Jiancheng Song, and Rong-Xiang Zhao. 2013. Monte Carlo Based Personalized PageRank on Dynamic Networks. *Int. J. Distributed Sens. Networks* 9 (2013).

[50] 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.