Approximating Single-Source Personalized PageRank with Absolute Error Guarantees

Zhewei Wei[1]
Renmin University of China, Beijing, China

Ji-Rong Wen
Renmin University of China, Beijing, China

Mingji Yang
Renmin University of China, Beijing, China

Abstract

Personalized PageRank (PPR) is an extensively studied and applied node proximity measure in graphs. For a pair of nodes s and t on a graph $G = (V, E)$, the PPR value $\pi(s, t)$ is defined as the probability that an α-discounted random walk from s terminates at t, where the walk terminates with probability α at each step. We study the classic *Single-Source PPR query*, which asks for PPR approximations from a given source node s to all nodes in the graph. Specifically, we aim to provide approximations with *absolute error* guarantees, ensuring that the resultant PPR estimates $\hat{\pi}(s, t)$ satisfy $\max_{t \in V} \left|\hat{\pi}(s, t) - \pi(s, t)\right| \le \varepsilon$ for a given error bound ε. We propose an algorithm that achieves this with high probability, with an expected running time of

- $\widetilde{O}\left(\sqrt{m}/\varepsilon\right)$ for directed graphs[2], where $m = |E|$;
- $\widetilde{O}\left(\sqrt{d_{\max}}/\varepsilon\right)$ for undirected graphs, where $d_{\max}$ is the maximum node degree in the graph;
- $\widetilde{O}\left(n^{\gamma-1/2}/\varepsilon\right)$ for power-law graphs, where $n = |V|$ and $\gamma \in \left(\frac{1}{2}, 1\right)$ is the extent of the power law.

These sublinear bounds improve upon existing results. We also study the case when *degree-normalized absolute error* guarantees are desired, requiring $\max_{t \in V} \left|\hat{\pi}(s, t)/d(t) - \pi(s, t)/d(t)\right| \le \varepsilon_d$ for a given error bound ε_d, where the graph is undirected and $d(t)$ is the degree of node t. We give an algorithm that provides this error guarantee with high probability, achieving an expected complexity of $\widetilde{O}\left(\sqrt{\sum_{t \in V} \pi(s, t)/d(t)}\Big/\varepsilon_d\right)$. This improves over the previously known $O(1/\varepsilon_d)$ complexity.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Graph Algorithms, Sublinear Algorithms, Personalized PageRank

Digital Object Identifier 10.4230/LIPIcs.ICDT.2024.9

Related Version *Full Version*: `https://arxiv.org/pdf/2401.01019.pdf` [50]

Funding This work was partially done at Gaoling School of Artificial Intelligence, Peng Cheng Laboratory, Beijing Key Laboratory of Big Data Management and Analysis Methods, and MOE Key Lab of Data Engineering and Knowledge Engineering. This research was supported in part by National Natural Science Foundation of China (No. U2241212, No. 61972401, No. 61932001, No. 61832017, No. U2001212), the major key project of PCL (PCL2021A12), Beijing Natural Science Foundation (No. 4222028), Beijing Outstanding Young Scientist Program No.BJJWZYJH012019100020098, Alibaba Group through Alibaba Innovative Research Program, and Huawei-Renmin University joint program on Information Retrieval.

Acknowledgements We also wish to acknowledge the support provided by Engineering Research Center of Next-Generation Intelligent Search and Recommendation, Ministry of Education. Finally, we thank the anonymous reviewers for their valuable comments.

[1] Zhewei Wei is the corresponding author.
[2] $\widetilde{O}(\cdot)$ suppresses polylog(n) factors.

1 Introduction

In graph mining, computing *node proximity* values efficiently is a fundamental problem with broad applications, as they provide quantitative amounts to measure the closeness or relatedness between the nodes. A basic and extensively used proximity measure is *Personalized PageRank (PPR)* [14], which is a direct variant of Google's renowned *PageRank* centrality [14]. PPR has found multifaced applications for local graph partitioning [4, 53, 17], node embedding [37, 40, 54], and graph neural networks [27, 12, 42], among many others [21].

We study the classic problem of approximating *Single-Source PPR (SSPPR)*, where we are given a source node s in the graph, and our goal is to approximate the PPR values of all nodes in the graph w.r.t. s. Particularly, we concentrate on the complexity bounds for approximating SSPPR with *absolute error* or *degree-normalized absolute error* guarantees. After examining the existing bounds for the problem, we present novel algorithms with improved complexities to narrow the margin between the previous upper bounds and the known lower bounds.

In the remainder of this section, we formally state the problem, discuss the existing bounds, and introduce our motivations and contributions.

1.1 Problem Formulation

We consider a directed or undirected graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. For undirected graphs, we conceptually view each undirected edge as two opposing directed edges. We assume that every node in V has a nonzero out-degree.

A *random walk* on G starts from some *source node* $s \in V$ and, at each step, transitions to an out-neighbor of the current node chosen uniformly at random. For a constant *decay factor* $\alpha \in (0, 1)$, an *α-discounted random walk* proceeds in the same way as a random walk, except that it terminates with probability α before each step. The *Personalized PageRank (PPR)* value for a pair of nodes s and t in V, denoted by $\pi(s, t)$, is defined as the probability that an α-discounted random walk from s terminates at t.

We study the problem of estimating *Single-Source PPR (SSPPR)*, that is, deriving $\hat{\pi}(s, t)$ as estimates for PPR values $\pi(s, t)$ from a given source node s to all $t \in V$. We focus on the complexities of approximating SSPPR with *absolute error* or *degree-normalized absolute error* guarantees. The corresponding two types of queries, dubbed as the *SSPPR-A query* and the *SSPPR-D query*, are formally defined below.

▶ **Definition 1** (SSPPR-A Query: Approximate SSPPR Query with Absolute Error Bounds). *Given a source node $s \in V$ and an error parameter ε, the query requires PPR estimates $\hat{\pi}(s, t)$ for all $t \in V$, such that $\left|\hat{\pi}(s, t) - \pi(s, t)\right| \le \varepsilon$ holds for all $t \in V$.*

▶ **Definition 2** (SSPPR-D Query: Approximate SSPPR Query with Degree-Normalized Absolute Error Bounds). *On an undirected graph, given a source node $s \in V$ and an error parameter ε_d, the query requires PPR estimates $\hat{\pi}(s, t)$ for all $t \in V$, such that $\left|\hat{\pi}(s, t)/d(t) - \pi(s, t)/d(t)\right| \le \varepsilon_d$ holds for all $t \in V$. Here, $d(t)$ denotes the degree of node t.*

In a word, the SSPPR-A query requires the maximum absolute error to be bounded above by ε, while the SSPPR-D query considers the absolute errors normalized (i.e., divided) by the degree of each node. Note that we restrict the SSPPR-D query to undirected graphs.

This paper aims to develop *sublinear* algorithms for the SSPPR-A and SSPPR-D queries with improved complexities over existing methods. By "sublinear algorithms," we refer to algorithms whose complexity bounds are sublinear in the size of the graph (but they can simultaneously depend on the error parameter, e.g., $O\big(\sqrt{n}/\varepsilon\big)$ is considered sublinear). We allow the algorithms to return the results as a sparse vector, which enables the output size to be $o(n)$. Also, we regard the algorithms as acceptable if they answer the queries *with high probability (w.h.p.)*, defined as "with probability at least $1 - 1/n$."

1.2 Prior Complexity Bounds

Lower Bounds

To our knowledge, only trivial lower bounds are known for the SSPPR-A and SSPPR-D queries. More precisely, for the SSPPR-A query, since the algorithm needs to return nonzero estimates for those nodes t with $\pi(s,t) > \varepsilon$, and there may exist $\Theta\big(\min(1/\varepsilon, n)\big)$ such nodes (note that $\sum_{t\in V}\pi(s,t) = 1$), the trivial lower bound for answering the query is $\Omega\big(\min(1/\varepsilon, n)\big)$. As we are considering sublinear bounds, we write this as $\Omega(1/\varepsilon)$.

Similarly, for the SSPPR-D query, the algorithm needs to return nonzero estimates for nodes t with $\pi(s,t)/d(t) > \varepsilon_d$, so the lower bound is $\Omega\left(1/\varepsilon_d \cdot \sum_{t\in V}\pi(s,t)/d(t)\right)$. Note that $\sum_{t\in V}\pi(s,t)/d(t) < \sum_{t\in V}\pi(s,t) = 1$ for nontrivial graphs. Additionally, in the full version of this paper [50], we show that if each source node $s \in V$ is chosen with probability proportional to its degree (i.e., with probability $d(s)/(2m)$), the average lower bound becomes $\Omega(1/\varepsilon_d \cdot n/m)$.

Upper Bounds

From a theoretical point of view, we summarize the best bounds to date for the SSPPR-A query as follows.

- The *Monte Carlo method* [16] straightforwardly simulates a number of α-discounted random walks from s, and computes the fraction of random walks that terminate at t as the estimate $\hat{\pi}(s,t)$ for each $t \in V$. By standard Chernoff bound arguments, it requires expected $\widetilde{O}\left(1/\varepsilon^2\right)$ time to achieve the guarantees w.h.p.
- *Forward Push* [4, 5] is a celebrated *"local-push"* algorithm for the SSPPR query, which takes as input a parameter $r_{\max}$ and runs in $O(1/r_{\max})$ time. However, it only guarantees that the degree-normalized absolute error (i.e., $\max_{v\in V}\left|\hat{\pi}(s,v)/d(v) - \pi(s,v)/d(v)\right|$) is bounded by $r_{\max}$ on undirected graphs, and that the ℓ_1*-error* (i.e., $\sum_{v\in V}\left|\hat{\pi}(s,v) - \pi(s,v)\right|$) is bounded by $m \cdot r_{\max}$ on directed graphs. Thus, if we apply Forward Push to answer the SSPPR-A query: on undirected graphs, we need to set $r_{\max} = \varepsilon/d_{\max}$, where $d_{\max}$ is the maximum node degree in G; on directed graphs, we need to set $r_{\max} = \varepsilon/m$. These settings lead to pessimistic bounds of $O(d_{\max}/\varepsilon)$ and $O(m/\varepsilon)$, respectively. Note that $d_{\max}$ can reach $\Theta(n)$ in the worst case, and the $O(m/\varepsilon)$ bound is not sublinear.
- *Backward Push* [1, 2] is a "local-push" algorithm for approximating $\pi(v,t)$ from all $v \in V$ to a given *target node* $t \in V$, known as the *Single-Target PPR (STPPR)* query. It takes as input a parameter $r_{\max}$ and cleanly returns estimates with an absolute error bound of $r_{\max}$. However, if we enforce Backward Push to answer the SSPPR-A query, we need to perform it with $r_{\max} = \varepsilon$ for each $t \in V$, resulting in a complexity of $O(m/\varepsilon)$ again. This bound is inferior, but we mention it here since it enlightens our algorithms.

In conclusion, the currently best sublinear bounds for the SSPPR-A query are $\widetilde{O}\left(1/\varepsilon^2\right)$ provided by Monte Carlo and $O(d_{\max}/\varepsilon)$ on undirected graphs by Forward Push.

As for the SSPPR-D query, Forward Push [5] provides an elegant $O(1/\varepsilon_d)$ bound. To our knowledge, no other prior methods are explicitly tailored to the SSPPR-D query.

1.3 Motivations

Motivations for the SSPPR-A Query

Although approximating SSPPR with absolute error guarantees is a natural problem, surprisingly, it has not been studied in depth in the literature. We believe that this is partly because of its inherent hardness. In particular, a line of recent research for approximating SSPPR [46, 30, 51, 29, 28] mainly focuses on providing *relative error* guarantees for PPR values above a specified threshold. We note that absolute error guarantees are harder to achieve than relative or degree-normalized absolute error guarantees, as the latter ones allow larger actual errors for nodes with larger PPR values or degrees. Specifically, an SSPPR algorithm with absolute error guarantees can be directly modified to obtain relative or degree-normalized absolute error guarantees.

In contrast, an interesting fact is that, for the relatively less-studied STPPR query, a simple Backward Push is sufficient and efficient for absolute error guarantees. As a result, when PPR values with absolute error guarantees are desired in some applications, STPPR methods are employed instead of SSPPR methods [54, 43].

These facts stimulate us to derive better bounds for the SSPPR-A query. As discussed, a large gap exists between the existing upper bounds and the lower bound of $\Omega(1/\varepsilon)$. The previous upper bounds, namely $\widetilde{O}\left(1/\varepsilon^2\right)$, $O(m/\varepsilon)$, and $O(d_{\max}/\varepsilon)$ on undirected graphs, motivate us to devise a new algorithm that:

- runs in linear time w.r.t. $1/\varepsilon$;
- runs in sublinear time w.r.t. m;
- beats the $O(d_{\max}/\varepsilon)$ bound on undirected graphs.

Motivations for the SSPPR-D Query

Our study of the SSPPR-D query is motivated by a classic approach of using approximate SSPPR to perform *local graph partitioning* [5, 3, 53, 17]. This task aims to detect a cut with provably small conductance near a specified seed node without scanning the whole graph. To this end, this classic approach computes approximate PPR values $\hat{\pi}(s,v)$ from the seed node s, sorts the nodes in decreasing order of $\hat{\pi}(s,v)/d(v)$, and then finds a desired cut based on this order. As the quality of this approach relies heavily on the approximation errors of the values $\hat{\pi}(s,v)/d(v)$, it is natural to consider the SSPPR-D query. Notably, in carrying out this framework, the seminal and celebrated `PageRank-Nibble` algorithm [5] employs Forward Push as a subroutine for approximating PPR values. As it turns out, the error bounds of Forward Push match the requirements of the SSPPR-D query, and its cost dominates the overall complexity of `PageRank-Nibble`. Therefore, an improved upper bound for the SSPPR-D query can potentially lead to faster local graph partitioning algorithms.

However, the SSPPR-D query is rarely studied afterward despite its significance. To our knowledge, no existing method overcomes the $O(1/\varepsilon_d)$ bound of Forward Push, nor has any previous work pointed out the gap between this bound and the aforementioned lower bound. Motivated by this, we formulate this problem and propose an algorithm that beats the known $O(1/\varepsilon_d)$ bound.

1.4 Our Results

We propose algorithms for the SSPPR-A and SSPPR-D queries under a unified framework, melding Monte Carlo and Backward Push in a novel and nontrivial way. Roughly speaking, we use Backward Push to reduce the variances of the Monte Carlo estimators, and we propose a novel technique called *Adaptive Backward Push* to control the cost of Backward Push for each node and balance its total cost with that of Monte Carlo. We summarize the improved bounds achieved by our algorithms as follows.

Improved Upper Bounds for the SSPPR-A Query

We present an algorithm that answers the SSPPR-A query w.h.p., with a complexity of:

- expected $\widetilde{O}\big(\sqrt{m}/\varepsilon\big)$ for directed graphs;
- expected $\widetilde{O}\big(\sqrt{d_{\max}}/\varepsilon\big)$ for undirected graphs.

These bounds are strictly sublinear in m and linear in $1/\varepsilon$. Also, the $\widetilde{O}\big(\sqrt{d_{\max}}/\varepsilon\big)$ bound improves over the previous $O(d_{\max}/\varepsilon)$ bound by up to a factor of $\Theta\big(\sqrt{n}\big)$.

Additionally, we study the special case that the underlying graph is a *power-law graph* (a.k.a. *scale-free graph*). This is a renowned and widely used model for describing large real-world graphs [10, 13]. Under power-law assumptions (see Assumption 3 in Subsection 3.3), we prove that the complexity of our algorithm diminishes to $\widetilde{O}\left(n^{\gamma-1/2}/\varepsilon\right)$ for both directed and undirected graphs, where $\gamma \in \left(\frac{1}{2}, 1\right)$ is the extent of the power law. Notably, as $\gamma < 1$, we have $\gamma - \frac{1}{2} < \frac{1}{2}$, so this bound is strictly $o\big(\sqrt{n}/\varepsilon\big)$. Also, when $\gamma \to \frac{1}{2}$, this bound approaches $\widetilde{O}(1/\varepsilon)$, matching the lower bound of $\Theta(1/\varepsilon)$ up to logarithmic factors. We summarize the complexity bounds of answering the SSPPR-A query in Table 1.

▶ Remark. Our algorithm for the SSPPR-A query can be adapted to approximate a more generalized form of Personalized PageRank [14], where the source node is randomly chosen from a given probability distribution vector. We only need to construct an alias structure [41] for the distribution (this can be done in asymptotically the same time as inputting the vector) so that we can sample a source node in $O(1)$ time when performing Monte Carlo. This modification does not change our algorithm's error guarantees and complexity bounds. Particularly, this allows us to estimate the PageRank [14] values, in which case we can sample the source nodes uniformly at random from V without using the alias method.

Improved Upper Bounds for the SSPPR-D Query

We present an algorithm that answers the SSPPR-D query w.h.p., with an expected complexity of $\widetilde{O}\Big(1/\varepsilon_d \cdot \sqrt{\sum_{t\in V} \pi(s,t)/d(t)}\Big)$. This improves upon the previous $O(1/\varepsilon_d)$ bound of Forward Push towards the lower bound of $\Omega\left(1/\varepsilon_d \cdot \sum_{t\in V} \pi(s,t)/d(t)\right)$. To see the superiority of our bound, let us consider the case when each node $s \in V$ is chosen as the source node with probability $d(s)/(2m)$. This setting corresponds to the practical scenario where a node with larger importance is more likely to be chosen as the source node. We show that under this setting, our bound becomes $\widetilde{O}\left(1/\varepsilon_d \cdot \sqrt{n/m}\right)$, which is lower than $O(1/\varepsilon_d)$ by up to a factor of $\Theta\big(\sqrt{n}\big)$. Recall that under this setting, the lower bound becomes $\Omega(1/\varepsilon_d \cdot n/m)$. In Table 2, we summarize the complexity bounds of answering the SSPPR-D query.

▶ Remark. If we treat α as a variable (as is the case in the context of local graph partitioning), the complexity bounds of our algorithms for these two queries both exhibit a linear dependence on $1/\alpha$, which is the same as existing upper bounds. For the sake of simplicity, we treat α as a constant and omit this term in this work.

Paper Organization. The remainder of this paper is organized as follows. Section 2 discusses some related work for PPR computation, and Section 3 offers the preliminaries. Section 4 presents the ideas and the main procedure of our proposed algorithm for the SSPPR-A query. In Section 5, we prove our results for the SSPPR-A query by analyzing our proposed algorithm. Some deferred proofs and our algorithm and analyses for the SSPPR-D query can be found in the full version of this paper [50].

Table 1 Complexity bounds of answering the SSPPR-A query on different types of graphs. For power-law graphs, the graph can be either directed or undirected, and $\gamma \in \left(\frac{1}{2}, 1\right)$ denotes the exponent of the power law. We plug in $m = \widetilde{O}(n)$ for power-law graphs.

	Directed Graphs	Undirected Graphs	Power-Law Graphs
Monte Carlo [16]	$\widetilde{O}\left(\frac{1}{\varepsilon^2}\right)$	$\widetilde{O}\left(\frac{1}{\varepsilon^2}\right)$	$\widetilde{O}\left(\frac{1}{\varepsilon^2}\right)$
Forward Push [4]	$O\left(\frac{m}{\varepsilon}\right)$	$O\left(\frac{d_{\max}}{\varepsilon}\right)$	$\widetilde{O}\left(\frac{n}{\varepsilon}\right)$
Ours	$\widetilde{O}\left(\frac{\sqrt{m}}{\varepsilon}\right)$	$\widetilde{O}\left(\frac{\sqrt{d_{\max}}}{\varepsilon}\right)$	$\widetilde{O}\left(\frac{n^{\gamma-1/2}}{\varepsilon}\right) = o\left(\frac{\sqrt{n}}{\varepsilon}\right)$, approaching $\widetilde{O}\left(\frac{1}{\varepsilon}\right)$ when $\gamma \to \frac{1}{2}$

Table 2 Complexity bounds of answering the SSPPR-D query on undirected graphs.

	Parameterized complexity for a given s	Average complexity when each $s \in V$ is chosen with probability $d(s)/(2m)$
Forward Push [4]	$O\left(\frac{1}{\varepsilon_d}\right)$	$O\left(\frac{1}{\varepsilon_d}\right)$
Lower Bound	$\Omega\left(\frac{1}{\varepsilon_d}\sum_{t\in V}\frac{\pi(s,t)}{d(t)}\right)$	$\Omega\left(\frac{1}{\varepsilon_d}\cdot\frac{n}{m}\right)$
Ours	$\widetilde{O}\left(\frac{1}{\varepsilon_d}\sqrt{\sum_{t\in V}\frac{\pi(s,t)}{d(t)}}\right)$	$\widetilde{O}\left(\frac{1}{\varepsilon_d}\sqrt{\frac{n}{m}}\right)$

2 Other Related Work

As a classic task in graph mining, PPR computation has been extensively studied in the past decades, and numerous efficient approaches have been proposed. Many recent methods combine the basic techniques of Monte Carlo, Forward Push, and Backward Push to achieve improved efficiency [32, 45, 49, 46, 30, 51, 29, 28]. A key ingredient in integrating these techniques is the *invariant* equation provided by Forward Push or Backward Push. While our algorithms also leverage the invariant of Backward Push to unify it with Monte Carlo, we adopt a novel approach based on Adaptive Backward Push and conduct different analyses.

For SSPPR approximation, `FORA` [47, 46] is a representative sublinear algorithm among a recent line of research [46, 30, 51, 29, 28]. `FORA` uses Forward Push and Monte Carlo to provide relative error guarantees for PPR values above a specified threshold w.h.p., and the subsequent work proposes numerous optimizations for it. However, this method cannot be directly applied to the SSPPR-A query. A notable extension of `FORA` is `SpeedPPR` [51], which further incorporates Power Method [14] to achieve higher efficiency. Nevertheless, the complexity of `SpeedPPR` is no longer sublinear. Among other studies for SSPPR [11, 59, 35, 39, 15, 26, 56], `BEAR` [39] and `BEPI` [26] are two representative approaches based on matrix manipulation. However, they incur inferior complexities due to the large overhead of matrix computation.

Algorithm 1 `BackwardPush`.

```
Input: graph $G$, decay factor $\alpha$, target node $t$, threshold $r_{\max}$
Output: backward reserves $q(v,t)$ and residues $r(v,t)$ for all $v \in V$
1 $q(v,t) \leftarrow 0$ for all $v \in V$
2 $r(t,t) \leftarrow 1$ and $r(v,t) \leftarrow 0$ for all $v \in V \setminus \{t\}$
3 while $\exists v \in V$ such that $r(v,t) > r_{\max}$ do
4     pick an arbitrary $v \in V$ with $r(v,t) > r_{\max}$
5     for each $u \in \mathcal{N}_{\text{in}}(v)$ do
6         $r(u,t) \leftarrow r(u,t) + (1-\alpha) \cdot r(v,t)/d_{\text{out}}(u)$
7     $q(v,t) \leftarrow q(v,t) + \alpha \cdot r(v,t)$
8     $r(v,t) \leftarrow 0$
9 return $q(v,t)$ and $r(v,t)$ for all $v \in V$
```

There also exist many studies for other PPR queries, such as Single-Pair query [20, 33, 32, 45], Single-Target query [43], and top-k query [7, 18, 20, 19, 52, 49]. Some recent work further considers computing PPR on dynamic graphs [57, 36, 58, 55] or in parallel/distributed settings [8, 22, 23, 38, 31, 44, 24]. These methods often utilize specifically designed methodologies and techniques, hence they are orthogonal to our work.

To sum up, despite the large body of studies devoted to PPR computation, the SSPPR-A and SSPPR-D queries are still not explored in depth. This is because the relevant approaches either are unsuitable for these two queries or exhibit at least linear complexities. We also note that many related studies optimize PPR computation from an engineering viewpoint instead of a theoretical one, and thus they do not provide better complexity bounds.

3 Notations and Tools

3.1 Notations

We use $d_{\text{in}}(v)$ and $d_{\text{out}}(v)$ to denote the in-degree and out-degree of a node $v \in V$, respectively. Additionally, $\mathcal{N}_{\text{in}}(v)$ and $\mathcal{N}_{\text{out}}(v)$ denote the in-neighbor and out-neighbor set of v, respectively. In the case of undirected graphs, we use $d(v)$ to represent the degree of v, and we define $d_{\max} = \max_{v \in V} d(v)$ to indicate the maximum degree in G.

3.2 Backward Push

Backward Push [2, 34] is a simple and classic algorithm for approximating STPPR, that is, estimating $\pi(v,t)$ from all $v \in V$ to a given target node $t \in V$. It works by repeatedly performing *reverse pushes*, which conceptually simulate random walks from the backward direction deterministically. It takes as input a parameter $r_{\max}$ to control the depth of performing the pushes: a smaller $r_{\max}$ leads to deeper pushes.

Specifically, Backward Push maintains *reserves* $q(v,t)$ and *residues* $r(v,t)$ for all $v \in V$, where $q(v,t)$ is an underestimate of $\pi(v,t)$ and $r(v,t)$ is the probability mass to be propagated. A reverse push operation for a node v transfers α portion of $r(v,t)$ to $q(v,t)$ and propagates the remaining probability mass to the in-neighbors of v, as per the probability that a random walk at the in-neighbors proceeds to v. As shown in Algorithm 1, Backward Push initially sets all reserves and residues to be 0 except that $r(t,t) = 1$, and then repeatedly performs reverse pushes to nodes v with $r(v,t) > r_{\max}$. After that, it returns $q(v,t)$'s as the estimates.

In this paper, we use the following properties of Backward Push [34]:

- The results of Backward Push satisfy $r(v,t) \le r_{\max}$ and $\left|q(v,t) - \pi(v,t)\right| \le r_{\max}$, $\forall v \in V$.
- The results of Backward Push satisfy the following invariant:

$$\pi(v,t) = q(v,t) + \sum_{u \in V} \pi(v,u) r(u,t), \ \forall v \in V. \tag{1}$$

- The complexity of Backward Push is

$$O\left(\frac{1}{r_{\max}} \sum_{v \in V} \pi(v,t) d_{\text{in}}(v)\right). \tag{2}$$

- Running Backward Push with parameter $r_{\max}$ for each $t \in V$ takes $O(m/r_{\max})$ time.

3.3 Power-Law Assumption

Power-law graphs are an extensively used model for describing real-world graphs [10, 13]. Regarding PPR computation, it is observed in [9] that the PPR values on power-law graphs also follow a power-law distribution. Formally, in our analyses for power-law graphs, we use the following assumption, which has been adopted in several relevant works for graph analysis [9, 32, 48]:

▶ **Assumption 3** (Power-Law Graph)**.** *In a power-law graph, for any source node $v \in V$, the i-th largest PPR value w.r.t. v equals $\Theta\left(\frac{i^{-\gamma}}{n^{1-\gamma}}\right)$, where $1 \le i \le n$ and $\gamma \in \left(\frac{1}{2}, 1\right)$ is the exponent of the power law.*

4 Our Algorithm for the SSPPR-A Query

This section presents our algorithm for answering the SSPPR-A query with improved complexity bounds. Our algorithm for the SSPPR-D query is given in the full version of this paper [50]. Before diving into the details, we give high-level ideas and introduce key techniques for devising and analyzing the algorithm.

4.1 High-Level Ideas

Recall that for the SSPPR-A query, a fixed absolute error bound is required for each node $t \in V$. We find it hard to achieve this using Forward Push and Monte Carlo, as they inherently incur larger errors for nodes with larger degrees or PPR values (for Monte Carlo, this can be seen when analyzing it using Chernoff bounds). Thus, to answer the SSPPR-A query, it is crucial to reduce the errors for these hard-case nodes efficiently. A straightforward idea is to run Backward Push from these nodes, although using an STPPR algorithm to answer the SSPPR query seems counterintuitive. As performing Backward Push alone for all nodes requires $O(m/\varepsilon)$ time, we combine it with Monte Carlo to achieve a lower complexity.

In a word, our proposed algorithms employ Backward Push to reduce the number of random walks needed in Monte Carlo. Intuitively, by running Backward Push for a node t, we simulate random walks backward from t. Consequently, when performing forward random walks in Monte Carlo, our objective shifts from reaching node t to reaching the intermediate nodes touched by Backward Push. This method significantly reduces the variances of the Monte Carlo estimators since the random walks can increment the estimated values even if they fail to reach t. However, a major difficulty is that we cannot afford to perform deep Backward Push for each $t \in V$, which is both expensive and unnecessary. To address this issue, we propose the following central technique: Adaptive Backward Push.

Adaptive Backward Push. A crucial insight behind our algorithms is that performing deep Backward Push for each $t \in V$ is wasteful. This is because doing so yields accurate estimates for $\pi(v,t)$ for all $v \in V$, but for SSPPR queries, only $\pi(s,t)$ is required to be estimated. Thus, instead of performing Backward Push deeply to propagate enough probability mass to each node, we only need to ensure that the probability mass pushed backward from t to s is sufficient to yield an accurate estimate for $\pi(s,t)$. This motivates us to perform Backward Push *adaptively*. More precisely, we wish to set smaller $r_{\max}(t)$ for t with larger $\pi(s,t)$, rendering the Backward Push process for them deeper. An intuitive explanation is that, for larger $\pi(s,t)$, the minimally acceptable estimates $\pi(s,t) - \varepsilon$ are larger, so we need to perform Backward Push deeper to push more probability mass to s. As an extreme example, if we know that $\pi(s,t) \le \varepsilon$ for some t, then we can simply return $\hat{\pi}(s,t) = 0$ as its PPR approximation, so we do not need to perform Backward Push for these nodes with small PPR values. On the other hand, our technique also adaptively balances the cost of Backward Push and Monte Carlo, which will be introduced in the next subsection.

To implement Adaptive Backward Push, a natural idea would be directly setting $r_{\max}(t)$ to be inversely proportional to $\pi(s,t)$. At first glance, this idea seems paradoxical since the $\pi(s,t)$ values are exactly what we aim to estimate. However, it turns out that rough estimates of them suffice for our purpose. In fact, Monte Carlo offers a simple and elegant way to roughly approximate $\pi(s,t)$ with relatively low overheads. Thus, we need to run Monte Carlo to obtain rough PPR estimates before performing Adaptive Backward Push. Despite the simplicity of the ideas, the detailed procedure of the algorithm and its analyses are nontrivial, as we elaborate below.

4.2 Techniques

Now, we introduce several additional techniques used in our algorithms.

Three-Phase Framework. As discussed above, before performing Adaptive Backward Push, we need to perform Monte Carlo to obtain rough PPR estimates. Also, the results of Backward Push and Monte Carlo are combined to derive the final results. For ease of analysis, we conduct Monte Carlo twice, yielding two independent sets of approximations. This leads to our *three-phase framework*:

(I) running Monte Carlo to obtain rough PPR estimates;
(II) performing Adaptive Backward Push to obtain backward reserves and residues;
(III) running Monte Carlo and combining the results with those of Backward Push to yield the final estimates.

However, there are some difficulties in carrying out this framework. We discuss them in detail below and provide a more accurate description of our algorithm in Subsection 4.3.

Identifying Candidate Nodes. As mentioned earlier, if we know that $\pi(s,t) \le \varepsilon$ for some t, then we can safely return $\hat{\pi}(s,t) = 0$. This means that we only need to consider nodes t with $\pi(s,t) > \varepsilon$. Fortunately, when running Monte Carlo to obtain rough estimates, we can also identify these nodes (w.h.p.), thus avoiding the unnecessary cost of performing Adaptive Backward Push for other nodes. We use C to denote the *candidate set* that consists of these identified *candidate nodes*. In light of this, Phase I serves two purposes: determining the candidate nodes $t \in C$ and obtaining rough estimates for their PPR values, denoted as $\pi'(s,t)$'s. Subsequently, in Phase II, we perform Adaptive Backward Push for these candidate nodes $t \in C$, where the thresholds are set to be inversely proportional to $\pi'(s,t)$.

Estimation Formula. Let us take a closer look at the results of Backward Push for a candidate node $t \in C$. From the invariant of Backward Push (Equation 1), we have

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

This essentially expresses the desired PPR value $\pi(s,t)$ as $q(s,t)$ plus a linear combination of $\pi(s,v)$'s with coefficients $r(v,t)$'s. Now, we can obtain an estimate $\hat{\pi}(s,t)$ by substituting $\pi(s,v)$'s by their Monte Carlo estimates obtained in Phase III, denoted by $\pi''(s,v)$'s:

$$\hat{\pi}(s,t) = q(s,t) + \sum_{v \in V} \pi''(s,v) r(v,t). \tag{3}$$

As Backward Push guarantees that $r(v,t) \le r_{\max}$ for all $v \in V$, the coefficients of these Monte Carlo estimates are small, making the variance of $\hat{\pi}(s,t)$ small. Thus, by carefully setting the parameters of Backward Push and Monte Carlo, we can bound this variance and apply *Chebyshev's inequality* to obtain the desired absolute error bound on $\hat{\pi}(s,t)$ with constant success probability. Then, we leverage the *median trick* [25] (see Appendix A) to amplify this probability while only introducing an additional logarithmic factor to the complexity. Therefore, in Phase III, we run Monte Carlo several times to obtain independent samples of $\hat{\pi}(s,t)$, denoted as $\hat{\pi}_i(s,t)$. After that, we compute their median values as the final estimates.

Balancing Phases II and III. Finally, to optimize the overall complexity of the algorithm, it is essential to strike a balance between the cost of Phases II and III. In fact, the following balancing strategy is another manifestation of the adaptiveness of our algorithm. In particular, performing more Adaptive Backward Push in Phase II results in fewer random walks needed in Phase III. In our analysis part (Section 5), we find that the complexity bound of Adaptive Backward Push is inversely proportional to the number of random walk samplings in Phase III (denoted as n_r), and the cost of Monte Carlo in Phase III is proportional to n_r. The problem is that we do not know a priori the optimal setting of n_r in terms of balancing Phases II and III. As a workaround, we propose to try running Adaptive Backward Push with exponentially decreasing n_r, and terminate this process once its paid cost exceeds the expected cost of simulating n_r random walks. Intuitively, in this way, our algorithm achieves the same asymptotic complexity as if it knew the optimal n_r. We will describe the detailed process and formally state this claim below.

4.3 Main Algorithm

Algorithm 2 outlines the pseudocode of our proposed algorithm for the SSPPR-A query. It consists of three phases: Phase I (line 2 to line 3), Phase II (line 5 to line 14), and Phase III (line 16 to line 19). We recap the purposes of the three phases as follows:

(I) running Monte Carlo to obtain estimates $\pi'(s,v)$ and derive the candidate set C;

(II) performing Adaptive Backward Push to obtain reserves $q(s,t)$ and residues $r(v,t)$ for candidate nodes $t \in C$, where the associated n_r (the number of random walk samplings in Phase III) is exponentially decreased and the stopping rule is designed to balance Phases II and III;

(III) running Monte Carlo several times, combining their results $\pi''(s,v)$ with those of Adaptive Backward Push, and finally taking the median values as the results $\hat{\pi}(s,t)$.

Algorithm 2 Our algorithm for the SSPPR-A query.

```
   Input: graph $G = (V, E)$, decay factor $\alpha$, source node $s$, error parameter $\varepsilon$
   Output: estimates $\hat{\pi}(s,t)$ for all $t \in V$
1  // Phase I
2  $\pi'(s,v)$ for all $v \in V \leftarrow$ Monte Carlo estimates with $\left\lceil 12\ln\left(2n^3\right)/\varepsilon \right\rceil$ random walks
3  $C \leftarrow \left\{t \in V : \pi'(s,t) > \frac{1}{2}\varepsilon\right\}$
4  // Phase II
5  $n_r \leftarrow \lceil n/\varepsilon \rceil, n_t \leftarrow \left\lceil 18\ln\left(2n^2\right)\right\rceil$
6  $r_{\max}(t) \leftarrow \frac{\varepsilon^2 n_r}{6\pi'(s,t)}$ for all $t \in C$
7  while True do
8      // the process in this loop is called an iteration
9      // in this iteration, try running Backward Push with $\frac{1}{2}r_{\max}(t)$'s
10     for each $t \in C$ do
11         run BackwardPush $\left(G, \alpha, t, \frac{1}{2}r_{\max}(t)\right)$, but once the total cost of Backward
             Push in this iteration exceeds the expected cost of simulating $n_t \cdot \frac{1}{2}n_r$
             random walks, terminate and break the outer loop (i.e., jump to line 14)
12     $n_r \leftarrow \left\lfloor \frac{1}{2}n_r \right\rfloor$
13     $r_{\max}(t) \leftarrow \frac{1}{2}r_{\max}(t)$ for all $t \in C$
14 $q(v,t), r(v,t)$ for $v \in V \leftarrow$ BackwardPush$\big(G, \alpha, t, r_{\max}(t)\big)$
15 // Phase III
16 for $i$ from 1 to $n_t$ do
17     $\pi''(s,v)$ for all $v \in V \leftarrow$ Monte Carlo estimates with $n_r$ random walks
18     $\hat{\pi}_i(s,t) \leftarrow q(s,t) + \sum_{v \in V} \pi''(s,v) r(v,t)$ for all $t \in C$
19 $\hat{\pi}(s,t) \leftarrow \mathrm{median}_{i=1}^{n_t}\left\{\hat{\pi}_i(s,t)\right\}$ for all $t \in C$
20 return $\hat{\pi}(s,t)$ for all $t \in V$
```

Concretely, in Phase I, the algorithm runs Monte Carlo (as introduced in Subsection 1.2) with $\left\lceil 12\ln\left(2n^3\right)/\varepsilon\right\rceil$ random walks (line 2), obtain estimates $\pi'(s,v)$, and sets the candidate set C to be $\left\{t \in V : \pi'(s,t) > \frac{1}{2}\varepsilon\right\}$ (line 3). Next, Phase II implements Adaptive Backward Push. It first initializes n_r, the number of random walk samplings in Phase III, to be $\lceil n/\varepsilon \rceil$, and sets n_t, the number of trials for the median trick, to be $\left\lceil 18\ln\left(2n^2\right)\right\rceil$ (line 5). Also, $r_{\max}(t)$ is initialized to be $\frac{\varepsilon^2 n_r}{6\pi'(s,t)}$ for each candidate node $t \in C$ (line 6). In the subsequent loop (line 7 to line 13), the algorithm repeatedly tries to run Backward Push (Algorithm 1) for each candidate node $t \in C$ with parameter $\frac{1}{2}r_{\max}(t)$, and iteratively halves n_r as well as $r_{\max}$ until the cost of Backward Push exceeds the expected cost of simulating random walks in Phase III. In that case, the algorithm immediately terminates Backward Push and jumps to line 14 (without halving n_r and $r_{\max}(t)$), where Backward Push is invoked again with the current $r_{\max}(t)$ to obtain the reserves and residues. Finally, in Phase III, the algorithm runs Monte Carlo with n_r random walks (line 17) and computes estimates $\hat{\pi}_i(s,t)$ according to Equation 3 (line 18). This process is repeated n_t times (line 16), and the final approximations $\hat{\pi}(s,t)$ are computed as $\mathrm{median}_{i=1}^{n_t}\left\{\hat{\pi}_i(s,t)\right\}$ (line 19).

Note that in line 18, for each $t \in C$, we only need to iterate through nodes v with a nonzero residue $r(v,t)$ to compute the summation. Thus, we can implement line 18 in asymptotically the same time as running Backward Push in Phase II.

In the following section, we demonstrate the correctness and efficiency of Algorithm 2.

5 Analyses for the SSPPR-A Query

We give correctness and complexity analyses of our Algorithm 2 for the SSPPR-A query. The analyses for the SSPPR-D query are given in the full version of this paper [50].

First, we give error bounds for the Monte Carlo estimates in Phase I. These are typical results for the Monte Carlo method, and we give a proof in the full version of this paper [50] for completeness.

▶ **Lemma 4.** *Let $\pi'(s,v)$ denote the estimate for $\pi(s,v)$ obtained in Phase I of Algorithm 2. With probability at least $1-1/n^2$, we have $\frac{1}{2}\pi(s,v) \le \pi'(s,v) \le \frac{3}{2}\pi(s,v)$ for all $v \in V$ with $\pi(s,v) \ge \frac{1}{4}\varepsilon$, and $\pi'(s,v) \le \pi(s,v) + \frac{1}{4}\varepsilon$ for all $v \in V$ with $\pi(s,v) < \frac{1}{4}\varepsilon$.*

We say that Phase I *succeeds* if the properties in Lemma 4 are satisfied. Our following discussions are implicitly conditioned on the success of Phase I, and we shall not specify this explicitly for ease of presentation. We only take this condition into account when considering the overall success probability of the algorithm. Also, we regard $\pi'(s,v)$ as fixed values when analyzing the remaining phases. Next, we show that Phase I prunes non-candidate nodes properly and guarantees constant relative error bounds for candidate nodes.

▶ **Lemma 5.** *All non-candidate nodes $t' \notin C$ satisfy $\pi(s,t') \le \varepsilon$, and all candidate nodes $t \in C$ satisfy $\frac{1}{2}\pi(s,t) \le \pi'(s,t) \le \frac{3}{2}\pi(s,t)$.*

The proof of Lemma 5 can be found in the full version of this paper [50]. Based on Lemma 5, we prove the correctness and complexity bounds of Algorithm 2 separately.

Correctness Analysis

Primarily, the following lemma justifies the unbiasedness of the estimators $\hat{\pi}_i(s,t)$:

▶ **Lemma 6.** *For all the candidate nodes $t \in C$, $\hat{\pi}_i(s,t)$ are unbiased estimators for $\pi(s,t)$, i.e., $\mathrm{E}\left[\hat{\pi}_i(s,t)\right] = \pi(s,t)$.*

The proof of Lemma 6 can be found in the full version of this paper [50]. Next, we prove the following key lemma, which bounds the variances of the estimators $\hat{\pi}_i(s,t)$ in terms of n_r, $r_{\max}(t)$, and $\pi(s,t)$:

▶ **Lemma 7.** *The variances $\mathrm{Var}\left[\hat{\pi}_i(s,t)\right]$ are bounded above by $1/n_r \cdot r_{\max}(t)\pi(s,t)$, for all $t \in C$. Here, n_r is the number of random walks in Phase III. This leads to $\mathrm{Var}\left[\hat{\pi}_i(s,t)\right] \le \frac{1}{3}\varepsilon^2$.*

Proof. We first calculate

$$\mathrm{Var}\left[\hat{\pi}_i(s,t)\right] = \mathrm{Var}\left[q(s,t) + \sum_{v \in V} \pi''(s,v) r(v,t)\right] = \mathrm{Var}\left[\sum_{v \in V} \pi''(s,v) r(v,t)\right].$$

To bound this variance, we use the fact that the Monte Carlo estimators $\pi'(s,v)$'s are *negatively correlated*, so the variance of their weighted sum is bounded by the sum of their weighted variances:

$$\mathrm{Var}\left[\hat{\pi}_i(s,t)\right] \le \sum_{v \in V} \mathrm{Var}\left[\pi''(s,v) r(v,t)\right] = \sum_{v \in V} \left(r(v,t)\right)^2 \mathrm{Var}\left[\pi''(s,v)\right].$$

Next, we plug in $\mathrm{Var}\left[\pi''(s,v)\right] = \pi(s,v)\big(1-\pi(s,v)\big)/n_r$, the variances of binomial random variables divided by n_r, to obtain

$$\begin{aligned}\mathrm{Var}\left[\hat{\pi}_i(s,t)\right] \le & \sum_{v\in V}\big(r(v,t)\big)^2 \cdot \frac{\pi(s,v)\big(1-\pi(s,v)\big)}{n_r} \\ = & \frac{1}{n_r}\sum_{v\in V} r(v,t)\big(\pi(s,v)r(v,t)\big)\big(1-\pi(s,v)\big) \le \frac{1}{n_r}\sum_{v\in V} r(v,t)\big(\pi(s,v)r(v,t)\big).\end{aligned}$$

Using the properties of Backward Push (see Subsection 3.2) that $r(v,t) \le r_{\max}(t)$ for all $v \in V$ and $\sum_{v\in V}\pi(s,v)r(v,t) = \pi(s,t) - q(s,t) \le \pi(s,t)$, we have

$$\mathrm{Var}\left[\hat{\pi}_i(s,t)\right] \le \frac{1}{n_r}\cdot r_{\max}(t)\sum_{v\in V}\pi(s,v)r(v,t) \le \frac{1}{n_r}\cdot r_{\max}(t)\pi(s,t).$$

This proves the first part of the lemma. Finally, by the setting of $r_{\max}(t) = \frac{\varepsilon^2 n_r}{6\pi'(s,t)}$ and the result in Lemma 5 that $\pi'(s,t) \ge \frac{1}{2}\pi(s,t)$ for candidate nodes $t \in C$, we obtain

$$\mathrm{Var}\left[\hat{\pi}_i(s,t)\right] \le \frac{1}{n_r}\cdot\frac{\varepsilon^2 n_r}{6\pi'(s,t)}\cdot\pi(s,t) \le \frac{1}{n_r}\cdot\frac{\varepsilon^2 n_r}{3\pi(s,t)}\cdot\pi(s,t) = \frac{1}{3}\varepsilon^2.$$

We conclude that $\mathrm{Var}\left[\hat{\pi}_i(s,t)\right] \le \frac{1}{3}\varepsilon^2$, as claimed. ◀

Now, we prove the following theorem, which verifies the correctness of Algorithm 2:

▶ **Theorem 8.** *Algorithm 2 answers the SSPPR-A query (defined in Definition 1) correctly with probability at least $1 - 1/n$.*

Proof. First, for non-candidate nodes $t' \notin C$, Lemma 5 guarantees that $\pi(s,t') \le \varepsilon$, so it is acceptable that Algorithm 2 returns $\hat{\pi}(s,t') = 0$ as their PPR estimates. For candidate nodes $t \in C$, Lemma 6 together with Chebyshev's inequality guarantees that

$$\Pr\left[\left|\hat{\pi}_i(s,t) - \pi(s,t)\right| \ge \varepsilon\right] \le \frac{\mathrm{Var}\left[\hat{\pi}_i(s,t)\right]}{\varepsilon^2} \le \frac{1}{3}.$$

Recall that Algorithm 2 sets the final estimates $\hat{\pi}(s,t)$ to be $\mathrm{median}_{i=1}^{n_t}\left\{\hat{\pi}_i(s,t)\right\}$, where $n_t = \left\lceil 18\ln\left(2n^2\right)\right\rceil$, and that $\hat{\pi}_i(s,t)$ for $1 \le i \le n_t$ are obtained from independent trials of n_r random walk samplings. Thus, by applying the median trick (Theorem 15), we know that for any $t \in C$, the probability that $\left|\hat{\pi}(s,t) - \pi(s,t)\right| \ge \varepsilon$ is at most $1/\left(2n^2\right)$. Since $|C| \le n$, by applying union bound for all $t \in C$, we prove that with probability at least $1 - 1/(2n)$, $\left|\hat{\pi}(s,t) - \pi(s,t)\right| \le \varepsilon$ holds for all $t \in C$, matching the error bound required in Definition 1. Lastly, recall that this probability is conditioned on the success of Phase I, whose probability is at least $1 - 1/n^2$ by Lemma 4. Thus, we conclude that the overall success probability is at least $\left(1 - 1/n^2\right)\left(1 - 1/(2n)\right) > 1 - 1/n$. ◀

Complexity Analysis

First, we formalize the claim given in Subsection 4.2 regarding the balance between Phases II and III, as follows. We prove the claim in the full version of this paper [50].

▷ Claim 9. Algorithm 2 achieves the asymptotic complexity as if the optimal n_r (in terms of balancing the complexities of Phases II and III) is previously known and Adaptive Backward Push is only performed once with this n_r.

Based on Claim 9, we can derive the complexity of Algorithm 2 on general directed graphs:

▶ **Theorem 10.** *The expected time complexity of Algorithm 2 on directed graphs is*

$$\widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t\in V}\pi(s,t)\sum_{v\in V}\pi(v,t)d_{\text{in}}(v)}\right).$$

Furthermore, this complexity is upper bounded by $\widetilde{O}\big(\sqrt{m}/\varepsilon\big)$.

Proof. First, the expected complexity of Phase I is $\widetilde{O}(1/\varepsilon)$, which is negligible in the overall complexity. In Phase II, Backward Push is invoked with parameter $r_{\max}(t)=\frac{\varepsilon^2 n_r}{6\pi'(s,t)}$ for each $t\in C$, where $\pi'(s,t)$ satisfies $\pi'(s,t)\le\frac{3}{2}\pi(s,t)$ by Lemma 5. Therefore, using the complexity of Backward Push (Equation 2), the complexity of Phase II is bounded by

$$\begin{aligned}
&O\left(\sum_{t\in C}\frac{\sum_{v\in V}\pi(v,t)d_{\text{in}}(v)}{r_{\max}(t)}\right)=O\left(\sum_{t\in C}\sum_{v\in V}\frac{\pi'(s,t)\pi(v,t)d_{\text{in}}(v)}{\varepsilon^2 n_r}\right)\\
\le&O\left(\sum_{t\in C}\sum_{v\in V}\frac{\pi(s,t)\pi(v,t)d_{\text{in}}(v)}{\varepsilon^2 n_r}\right)\le O\left(\sum_{t\in V}\sum_{v\in V}\frac{\pi(s,t)\pi(v,t)d_{\text{in}}(v)}{\varepsilon^2 n_r}\right)\\
=&O\left(\frac{1}{\varepsilon^2 n_r}\sum_{t\in V}\pi(s,t)\sum_{v\in V}\pi(v,t)d_{\text{in}}(v)\right). \qquad (4)
\end{aligned}$$

On the other hand, Phase III performs $n_t\cdot n_r=\left\lceil 18\ln\left(2n^2\right)\right\rceil\cdot n_r=\widetilde{O}(n_r)$ random walks, so the total complexity of Phases II and III is $\widetilde{O}\left(1/\left(\varepsilon^2 n_r\right)\cdot\sum_{t\in V}\pi(s,t)\sum_{v\in V}\pi(v,t)d_{\text{in}}(v)+n_r\right)$. By the AM–GM inequality, the optimal setting of n_r minimizes this bound to be

$$\widetilde{O}\left(\sqrt{n_r\cdot\frac{1}{\varepsilon^2 n_r}\sum_{t\in V}\pi(s,t)\sum_{v\in V}\pi(v,t)d_{\text{in}}(v)}\right)=\widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t\in V}\pi(s,t)\sum_{v\in V}\pi(v,t)d_{\text{in}}(v)}\right).$$

By Claim 9, Algorithm 2 achieves this complexity.

In order for a simple upper bound, we use $\pi(v,t)\le 1$ for all $v,t\in V$ to obtain

$$\begin{aligned}
&\widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t\in V}\pi(s,t)\sum_{v\in V}\pi(v,t)d_{\text{in}}(v)}\right)\le\widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t\in V}\pi(s,t)\sum_{v\in V}d_{\text{in}}(v)}\right)\\
=&\widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t\in V}\pi(s,t)\cdot m}\right)=\widetilde{O}\left(\frac{\sqrt{m}}{\varepsilon}\right),
\end{aligned}$$

as claimed.

A subtle technicality here is that these complexity bounds are conditioned on the success of Phase I, and the expected complexity of Phase II may be unbounded if Phase I fails. To resolve this technical issue, we can switch to using naïve Power Method [14] once the actual cost of the algorithm reaches $\Theta\left(n^2\right)$. In this way, even if Phase I fails, the algorithm will solve the query in $\widetilde{O}\left(n^2\right)$ time. As the failure probability of Phase I is at most $1/n^2$ (Lemma 4), this merely adds a term of $1/n^2\cdot\widetilde{O}\left(n^2\right)=\widetilde{O}(1)$ to the overall expected complexity, and thus does not affect the resultant bounds. We will omit this technicality in later proofs. ◀

Next, we analyze the complexity of Algorithm 2 on general undirected graphs. To this end, the following previously known symmetry theorem will be helpful.

▶ **Theorem 11** (Symmetry of PPR on Undirected Graphs [6, Lemma 1]). *For all nodes $u, v \in V$, we have $\pi(u,v)d(u) = \pi(v,u)d(v)$.*

Now, we can derive a better complexity bound of Algorithm 2 on undirected graphs.

▶ **Theorem 12.** *The expected time complexity of Algorithm 2 on undirected graphs is*

$$\widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t\in V}\pi(s,t)d(t)}\right).$$

Furthermore, this complexity is upper bounded by $\widetilde{O}\big(\sqrt{d_{\max}}/\varepsilon\big)$.

Proof. Using Theorem 11, we can simplify the first complexity in Theorem 10 as follows:

$$\widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t\in V}\pi(s,t)\sum_{v\in V}\pi(v,t)d(v)}\right) = \widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t\in V}\pi(s,t)\sum_{v\in V}\pi(t,v)d(t)}\right)$$
$$=\widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t\in V}\pi(s,t)d(t)\sum_{v\in V}\pi(t,v)}\right) = \widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t\in V}\pi(s,t)d(t)}\right).$$

To prove the upper bound of $\widetilde{O}\big(\sqrt{d_{\max}}/\varepsilon\big)$, we use $d(t) \le d_{\max}$ for all $t \in V$ to obtain

$$\widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t\in V}\pi(s,t)d(t)}\right) \le \widetilde{O}\left(\frac{\sqrt{d_{\max}}}{\varepsilon}\cdot\sqrt{\sum_{t\in V}\pi(s,t)}\right) = \widetilde{O}\left(\frac{\sqrt{d_{\max}}}{\varepsilon}\right).$$ ◀

We note that the bounds $\widetilde{O}\big(\sqrt{m}/\varepsilon\big)$ and $\widetilde{O}\big(\sqrt{d_{\max}}/\varepsilon\big)$ above are for worst-case graphs, and for power-law graphs (Assumption 3), we can derive better bounds. In the full version of this paper [50], we prove the following lemma, which bounds $\sum_{t\in V}\left(\pi(v,t)\right)^2$ for any $v \in V$:

▶ **Lemma 13.** *On a power-law graph, $\sum_{t\in V}\left(\pi(v,t)\right)^2 = O\left(n^{2\gamma-2}\right)$ holds for any $v \in V$.*

Now, we can bound the complexity on power-law graphs as follows:

▶ **Theorem 14.** *The expected complexity of Algorithm 2 on power-law graphs is $\widetilde{O}\left(n^{\gamma-1/2}/\varepsilon\right)$.*

Proof. We prove the theorem using *Cauchy–Schwarz inequality* and power-law assumptions to simplify the first complexity given in Theorem 10. First, reordering the summations yields

$$\widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t\in V}\pi(s,t)\sum_{v\in V}\pi(v,t)d_{\text{in}}(v)}\right) = \widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{v\in V}d_{\text{in}}(v)\sum_{t\in V}\pi(s,t)\pi(v,t)}\right),$$

where

$$\sum_{t\in V}\pi(s,t)\pi(v,t) \le \sqrt{\left(\sum_{t\in V}\left(\pi(s,t)\right)^2\right)\left(\sum_{t\in V}\left(\pi(v,t)\right)^2\right)}$$

by Cauchy–Schwarz inequality. By Lemma 13, both $\sum_{t\in V}\left(\pi(s,t)\right)^2$ and $\sum_{t\in V}\left(\pi(v,t)\right)^2$ are bounded by $O\left(n^{2\gamma-2}\right)$. Consequently, $\sum_{t\in V}\pi(s,t)\pi(v,t) \le O\left(n^{2\gamma-2}\right)$, and the expression in question can be bounded by

$$\widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{v\in V}d_{\text{in}}(v)\cdot n^{2\gamma-2}}\right) = \widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{m\cdot n^{2\gamma-2}}\right) = \widetilde{O}\left(\frac{n^{\gamma-1/2}}{\varepsilon}\right),$$

where we used the fact that $m = \widetilde{O}(n)$ on power-law graphs. ◀

References

1 Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. Local computation of pagerank contributions. In *Proc. 5th Int. Workshop Algorithms Models Web Graph*, volume 4863, pages 150–165, 2007. doi:10.1007/978-3-540-77004-6_12.

2 Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. Local computation of pagerank contributions. *Internet Math.*, 5(1):23–45, 2008. doi:10.1080/15427951.2008.10129302.

3 Reid Andersen and Fan R. K. Chung. Detecting sharp drops in pagerank and a simplified local partitioning algorithm. In *Proc. 4th Int. Conf. Theory Appl. Models Comput.*, volume 4484, pages 1–12, 2007. doi:10.1007/978-3-540-72504-6_1.

4 Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Local graph partitioning using pagerank vectors. In *Proc. 47th Annu. IEEE Symp. Found. Comput. Sci.*, pages 475–486, 2006. doi:10.1109/FOCS.2006.44.

5 Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Using pagerank to locally partition a graph. *Internet Math.*, 4(1):35–64, 2007. doi:10.1080/15427951.2007.10129139.

6 Konstantin Avrachenkov, Paulo Gonçalves, and Marina Sokol. On the choice of kernel and labelled data in semi-supervised learning methods. In *Proc. 10th Int. Workshop Algorithms Models Web Graph*, volume 8305, pages 56–67, 2013. doi:10.1007/978-3-319-03536-9_5.

7 Konstantin Avrachenkov, Nelly Litvak, Danil Nemirovsky, Elena Smirnova, and Marina Sokol. Quick detection of top-k personalized pagerank lists. In *Proc. 8th Int. Workshop Algorithms Models Web Graph*, volume 6732, pages 50–61, 2011. doi:10.1007/978-3-642-21286-4_5.

8 Bahman Bahmani, Kaushik Chakrabarti, and Dong Xin. Fast personalized pagerank on mapreduce. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 973–984, 2011. doi:10.1145/1989323.1989425.

9 Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *Proc. VLDB Endowment*, 4(3):173–184, 2010. doi:10.14778/1929861.1929864.

10 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. doi:10.1126/science.286.5439.509.

11 Pavel Berkhin. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Math.*, 3(1):41–62, 2006. doi:10.1080/15427951.2006.10129116.

12 Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 2464–2473, 2020. doi:10.1145/3394486.3403296.

13 Béla Bollobás, Christian Borgs, Jennifer T. Chayes, and Oliver Riordan. Directed scale-free graphs. In *Proc. ACM-SIAM Symp. Discrete Algorithms*, pages 132–139, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644133.

14 Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw.*, 30(1-7):107–117, 1998. doi:10.1016/S0169-7552(98)00110-X.

15 Mustafa Coşkun, Ananth Grama, and Mehmet Koyutürk. Efficient processing of network proximity queries via chebyshev acceleration. In *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 1515–1524, 2016. doi:10.1145/2939672.2939828.

16 Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Math.*, 2(3):333–358, 2005. doi:10.1080/15427951.2005.10129104.

17 Kimon Fountoulakis, Farbod Roosta-Khorasani, Julian Shun, Xiang Cheng, and Michael W. Mahoney. Variational perspective on local graph clustering. *Math. Program.*, 174(1-2):553–573, 2019. doi:10.1007/S10107-017-1214-8.

18 Yasuhiro Fujiwara, Makoto Nakatsuji, Makoto Onizuka, and Masaru Kitsuregawa. Fast and exact top-k search for random walk with restart. *Proc. VLDB Endowment*, 5(5):442–453, 2012. doi:10.14778/2140436.2140441.

19 Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Takeshi Mishima, and Makoto Onizuka. Efficient ad-hoc search for personalized pagerank. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 445–456, 2013. doi:10.1145/2463676.2463717.

20 Yasuhiro Fujiwara, Makoto Nakatsuji, Takeshi Yamamuro, Hiroaki Shiokawa, and Makoto Onizuka. Efficient personalized pagerank with accuracy assurance. In *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 15–23, 2012. doi:10.1145/2339530.2339538.

21 David F. Gleich. Pagerank beyond the web. *SIAM Rev.*, 57(3):321–363, 2015. doi:10.1137/140976649.

22 Tao Guo, Xin Cao, Gao Cong, Jiaheng Lu, and Xuemin Lin. Distributed algorithms on exact personalized pagerank. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 479–494, 2017. doi:10.1145/3035918.3035920.

23 Wentian Guo, Yuchen Li, Mo Sha, and Kian-Lee Tan. Parallel personalized pagerank on dynamic graphs. *Proc. VLDB Endowment*, 11(1):93–106, 2017. doi:10.14778/3151113.3151121.

24 Guanhao Hou, Xingguang Chen, Sibo Wang, and Zhewei Wei. Massively parallel algorithms for personalized pagerank. *Proc. VLDB Endowment*, 14(9):1668–1680, 2021. doi:10.14778/3461535.3461554.

25 Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986. doi:10.1016/0304-3975(86)90174-X.

26 Jinhong Jung, Namyong Park, Lee Sael, and U Kang. Bepi: Fast and memory-efficient method for billion-scale random walk with restart. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 789–804, 2017. doi:10.1145/3035918.3035950.

27 Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *Proc. 7th Int. Conf. Learn. Representations*, 2019. URL: https://openreview.net/forum?id=H1gL-2A9Ym.

28 Meihao Liao, Rong-Hua Li, Qiangqiang Dai, Hongyang Chen, Hongchao Qin, and Guoren Wang. Efficient personalized pagerank computation: The power of variance-reduced monte carlo approaches. *Proc. ACM Manage. Data*, 1(2):160:1–160:26, 2023. doi:10.1145/3589305.

29 Meihao Liao, Rong-Hua Li, Qiangqiang Dai, and Guoren Wang. Efficient personalized pagerank computation: A spanning forests sampling based approach. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 2048–2061, 2022. doi:10.1145/3514221.3526140.

30 Dandan Lin, Raymond Chi-Wing Wong, Min Xie, and Victor Junqiu Wei. Index-free approach with theoretical guarantee for efficient random walk with restart query. In *Proc. 36th Int. Conf. Data Eng.*, pages 913–924, 2020. doi:10.1109/ICDE48307.2020.00084.

31 Wenqing Lin. Distributed algorithms for fully personalized pagerank on large graphs. In *Proc. Int. Conf. World Wide Web*, pages 1084–1094, 2019. doi:10.1145/3308558.3313555.

32 Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. Personalized pagerank estimation and search: A bidirectional approach. In *Proc. 9th ACM Int. Conf. Web Search Data Mining*, pages 163–172, 2016. doi:10.1145/2835776.2835823.

33 Peter Lofgren, Siddhartha Banerjee, Ashish Goel, and Seshadhri Comandur. Fast-ppr: scaling personalized pagerank estimation for large graphs. In *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 1436–1445, 2014. doi:10.1145/2623330.2623745.

34 Peter Lofgren and Ashish Goel. Personalized pagerank to a target node. *CoRR*, abs/1304.4658, 2013. doi:10.48550/arXiv.1304.4658.

35 Takanori Maehara, Takuya Akiba, Yoichi Iwata, and Ken-ichi Kawarabayashi. Computing personalized pagerank quickly by exploiting graph structures. *Proc. VLDB Endowment*, 7(12):1023–1034, 2014. doi:10.14778/2732977.2732978.

36 Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. Efficient pagerank tracking in evolving networks. In *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 875–884, 2015. doi:10.1145/2783258.2783297.

37 Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 1105–1114, 2016. doi:10.1145/2939672.2939751.

38 Jieming Shi, Renchi Yang, Tianyuan Jin, Xiaokui Xiao, and Yin Yang. Realtime top-k personalized pagerank over large graphs on gpus. *Proc. VLDB Endowment*, 13(1):15–28, 2019. doi:10.14778/3357377.3357379.

39 Kijung Shin, Jinhong Jung, Lee Sael, and U Kang. Bear: Block elimination approach for random walk with restart on large graphs. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 1571–1585, 2015. doi:10.1145/2723372.2723716.

40 Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *Proc. Int. Conf. World Wide Web*, pages 539–548, 2018. doi:10.1145/3178876.3186120.

41 Alastair J Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 8(10):127–128, 1974. doi:10.1049/el:19740097.

42 Hanzhi Wang, Mingguo He, Zhewei Wei, Sibo Wang, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Approximate graph propagation. In *Proc. 27th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 1686–1696, 2021. doi:10.1145/3447548.3467243.

43 Hanzhi Wang, Zhewei Wei, Junhao Gan, Sibo Wang, and Zengfeng Huang. Personalized pagerank to a target node, revisited. In *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 657–667, 2020. doi:10.1145/3394486.3403108.

44 Runhui Wang, Sibo Wang, and Xiaofang Zhou. Parallelizing approximate single-source personalized pagerank queries on shared memory. *VLDB J.*, 28(6):923–940, 2019. doi:10.1007/S00778-019-00576-7.

45 Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. Hubppr: Effective indexing for approximate personalized pagerank. *Proc. VLDB Endowment*, 10(3):205–216, 2016. doi:10.14778/3021924.3021936.

46 Sibo Wang, Renchi Yang, Runhui Wang, Xiaokui Xiao, Zhewei Wei, Wenqing Lin, Yin Yang, and Nan Tang. Efficient algorithms for approximate single-source personalized pagerank queries. *ACM Trans. Database Syst.*, 44(4):18:1–18:37, 2019. doi:10.1145/3360902.

47 Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. Fora: Simple and effective approximate single-source personalized pagerank. In *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 505–514, 2017. doi:10.1145/3097983.3098072.

48 Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Yu Liu, Xiaoyong Du, and Ji-Rong Wen. Prsim: Sublinear time simrank computation on large power-law graphs. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 1042–1059, 2019. doi:10.1145/3299869.3319873.

49 Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. Topppr: Top-k personalized pagerank queries with precision guarantees on large graphs. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 441–456, 2018. doi:10.1145/3183713.3196920.

50 Zhewei Wei, Ji-Rong Wen, and Mingji Yang. Approximating single-source personalized pagerank with absolute error guarantees. *CoRR*, abs/2401.01019, 2024. doi:10.48550/arXiv.2401.01019.

51 Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. Unifying the global and local approaches: An efficient power iteration with forward push. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 1996–2008, 2021. doi:10.1145/3448016.3457298.

52 Yubao Wu, Ruoming Jin, and Xiang Zhang. Fast and unified local search for random walk based k-nearest-neighbor query in large graphs. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 1139–1150, 2014. doi:10.1145/2588555.2610500.

53 Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. Local higher-order graph clustering. In *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 555–564, 2017. doi:10.1145/3097983.3098069.

54 Yuan Yin and Zhewei Wei. Scalable graph embeddings via sparse transpose proximities. In *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 1429–1437, 2019. doi:10.1145/3292500.3330860.

55 Minji Yoon, Woojeong Jin, and U Kang. Fast and accurate random walk with restart on dynamic graphs with guarantees. In *Proc. Int. Conf. World Wide Web*, pages 409–418, 2018. doi:10.1145/3178876.3186107.

56 Minji Yoon, Jinhong Jung, and U Kang. TPA: Fast, scalable, and accurate method for approximate random walk with restart on billion scale graphs. In *Proc. 34th Int. Conf. Data Eng.*, pages 1132–1143, 2018. doi:10.1109/ICDE.2018.00105.

57 Weiren Yu and Xuemin Lin. IRWR: Incremental random walk with restart. In *Proc. 36th ACM SIGIR Int. Conf. Res. Develop. Inf. Retrieval*, pages 1017–1020, 2013. doi:10.1145/2484028.2484114.

58 Weiren Yu and Julie A. McCann. Random walk with restart over dynamic graphs. In *Proc. 16th Int. Conf. Data Mining*, pages 589–598, 2016. doi:10.1109/ICDM.2016.0070.

59 Fanwei Zhu, Yuan Fang, Kevin Chen-Chuan Chang, and Jing Ying. Incremental and accuracy-aware personalized pagerank through scheduled approximation. *Proc. VLDB Endowment*, 6(6):481–492, 2013. doi:10.14778/2536336.2536348.

A Median Trick

▶ **Theorem 15** (Median Trick [25]). *Let* $X_1, X_2, \ldots, X_{n_t}$ *be* n_t *i.i.d. random variables such that* $\Pr\left[|X_i - \mu| \geq \lambda\right] \leq \frac{1}{3}$ *for any* $1 \leq i \leq n_t$*, where* $\mu = \mathrm{E}[X_i]$*, and let* $X = \mathrm{median}_{1 \leq i \leq n_t}\{X_i\}$*. For a given probability* p_f*, if* $n_t \geq 18\ln(1/p_f) = \Theta\big(\log(1/p_f)\big)$*, then* $\Pr\left[\big|X - \mu\big| \geq \lambda\right] \leq p_f$*. Here,* $\mathrm{median}_{1 \leq i \leq n_t}\{X_i\}$ *is defined as the* $\lceil \frac{n_t}{2} \rceil$*-th smallest element in* $X_1, X_2, \ldots, X_{n_t}$*.*