Estimating Single-Node PageRank in $\tilde{O}\left(\min\{d_t, \sqrt{m}\}\right)$ Time

Hanzhi Wang
Renmin University of China
Beijing, China
hanzhi_wang@ruc.edu.cn

Zhewei Wei
Renmin University of China
Beijing, China
zhewei@ruc.edu.cn

ABSTRACT

PageRank is a famous measure of graph centrality that has numerous applications in practice. The problem of computing a single node's PageRank has been the subject of extensive research over a decade. However, existing methods still incur large time complexities despite years of efforts. Even on undirected graphs where several valuable properties held by PageRank scores, the problem of locally approximating the PageRank score of a target node remains a challenging task. Two commonly adopted techniques, Monte-Carlo based random walks and backward push, both cost $O(n)$ time in the worst-case scenario, which hinders existing methods from achieving a sublinear time complexity like $O(\sqrt{m})$ on an undirected graph with n nodes and m edges.

In this paper, we focus on the problem of single-node PageRank computation on undirected graphs. We propose a novel algorithm, *SetPush*, for estimating single-node PageRank specifically on undirected graphs. With non-trival analysis, we prove that our *SetPush* achieves the $\tilde{O}\left(\min\left\{d_t, \sqrt{m}\right\}\right)$ time complexity for estimating the target node t's PageRank with constant relative error and constant failure probability on undirected graphs. We conduct comprehensive experiments to demonstrate the effectiveness of *SetPush*.

PVLDB Reference Format:
Hanzhi Wang and Zhewei Wei. Estimating Single-Node PageRank in $\tilde{O}\left(\min\{d_t, \sqrt{m}\}\right)$ Time. PVLDB, 16(11): 2949-2961, 2023.
doi:10.14778/3611479.3611500

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/wanghzccls/SetPush-code.

1 INTRODUCTION

PageRank is first proposed by Google [33] to rank the importance of web pages in the search engine. It is formulated based on two intuitive arguments: (i) highly linked pages are more important than the pages with fewer links; (ii) the page that linked by an important page is also important. If we convert the web structure to a graph, the PageRank scores of all pages in the web correspond to the probability distribution of simulating random walks on the graph. Specifically, consider a graph $G = (V, E)$ with $|V| = n$ nodes and $|E| = m$ edges. We select a node s from the graph's vertex set V uniformly at random, and simulate an α-random walk from node s. The PageRank score of node $t \in V$ is equal to the probability that an α-random walk simulated from node s terminates at node t. Here we call s the source node. α-random walk refers to the random walk process that at each step (e.g., at node u), the walk either terminates at u with probability α, or moves to a randomly selected neighbor of u with probability $1 - \alpha$. We call α *the teleport probability* or *the damping factor*, which is a constant satisfying $\alpha \in (0, 1)$.

Over the last decade, PageRank has emerged as one of the most well-adopted graph centrality measure [18]. The applications of PageRank has been far beyond its origin in web search, covering a wide range of research domains, such as social networks, recommender systems, databases, as well as biology, chemistry, neuroscience and etc. For example, in social networks, PageRank serves as a classic role in evaluating the centrality of individuals. Kwak et al. [24] use PageRank to characterize the properties of Twitter. In recommender systems, the PageRank scores of items are adopted to find potential predictions [9]. Moreover, for the problem of database queries, the PageRank score indicates a query direction to the frequently retrieved results, and thus accelerates the query efficiency [6]. Additionally, PageRank are adopted to study molecules in chemistry [31], gene in biology [32] and brain regions in neuroscience [43]. More applications of PageRank can be found in the comprehensive survey summarized by Gleich [18].

At the same time, a plethora of variants stem from PageRank, including Personalized PageRank [33], heat kernel PageRank [14], reverse PageRank [7], weighted PageRank [41] and so on. For example, Personalized PageRank, one of the most famous variant of PageRank, has been an essential node proximity metric adopted in various web search and representation tasks [8, 19, 22]. Recall that PageRank serves as a global centrality measure in a graph. In comparison, the Personalized PageRank value of a node indicates a localized score, reflecting the relative importance of the node with respect to a given source node. Likewise, the heat kernel PageRank has a successful history in the local clustering scenario. A series of algorithms [14, 23, 42] leverage the scores of Heat Kernel PageRank to identify a well-connected cluster around the given seed node. These variants and their wide-spread applications also demonstrate the prominence of PageRank in graph analysis and mining tasks.

Given the huge success achieved by PageRank, the problem of computing PageRank scores has been the subject of extensive research for more than a decade [2, 7, 10, 17, 27–29]. One particular interest is the problem of *single-node PageRank computation*, which aims to compute a single node's PageRank on large-scale graphs. Such problem is an important primitive in graph analysis and learning tasks of both practical and theoretical interest.

Proceedings of the VLDB Endowment, Vol. 16, No. 11 ISSN 2150-8097.
doi:10.14778/3611479.3611500

Zhewei Wei is the corresponding author. The 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.

From the theoretical aspect, the query time complexity of single-node PageRank has a close connection to various graph analysis problems. For example, as we shall show in Section 2, node t's PageRank is equal to the average over all nodes u's $\boldsymbol{\pi}_u(t)$, where $\boldsymbol{\pi}_u(t)$ denotes the Personalized PageRank (PPR) score of node t with respect to node u. We call such problem single-target PPR queries, in which we aim to estimate $\boldsymbol{\pi}_u(t)$ of every node $u \in V$. The theoretical insight for single-node PageRank computation can therefore be used for single-target PPR queries by definition. Moreover, Bressan et al. [10] propose a novel method called *SubgraphPush* for single-node PageRank computation, and adapt the SubgraphPush method to computing single-node Heat Kernel PageRank (HKPR) by leveraging the analogue between PageRank and HKPR.

On the other hand, in many practical cases, all we need is an approximation of a few nodes' PageRank scores. For example, in the application scenario of web search, the changes in the importance of a few popular websites (e.g., the top-10 most popular websites ranked last year) is of particular interest. Since websites' global importance can be reflected from their PageRank scores, the PageRank scores of the ten websites are therefore frequently requested. Note that it would be prohibitively slow to score all nodes in the graph every time, especially on large-scale graphs with millions or even billions of nodes and edges. Therefore, an ideal solution is a *local* algorithm, which is able to efficiently return the target node's approximation scores by only exploring a small fraction of graph edges around the target node. However, as pointed out by [10], most of existing approaches require an $\Omega(n)$ time complexity for the single-node PageRank computation. Designing an efficient local algorithm with $o(n)$ query time complexity remains a challenge.

Single-Node PageRank Computation on Undirected Graphs. Existing methods for single-node PageRank computation mainly focus on directed graphs, which, however, incur large query time complexity despite decades of efforts due to the hardness. In this paper, we settle for a slightly less ambitious target to efficiently estimate single-node PageRank *on undirected graphs*. Note that the problem of single-node PageRank computation on undirected graphs is still of great importance from both practical and theoretical aspects. Specific reasons are illustrated in the following.

- From the theoretical aspect, a number of existing algorithms do not offer any worst-case guarantee on directed graphs without considering a uniform random choice of the target node. For these methods, meaningful complexity bounds can only be derived on undirected graphs when we consider an arbitrary target node (e.g., the LocalPush [28], FastPPR [29], and BiPPR [27] methods as listed in Table 1). On the other hand, there are several crucial properties of the PageRank scores that are only held on undirected graphs. This motivates us to study the problem of single-node PageRank computation specifically on undirected graphs for achieving better complexity results by utilizing these crucial properties delicately.
- Second, from the practical aspect, many downstream graph mining and learning tasks are only defined on undirected graphs. For example, in the scenario of local clustering, the celebrated local clustering method [3] employs (Personalized) PageRank vector to identify local clusters, while the well-adopted *conductance* metric to measure the quality of identified clusters is defined on undirected graphs. Therefore, in local clustering, all we need is the PageRank scores on undirected graphs. Additionally, Graph Neural Networks (GNNs) have drawn increasing attention in recent years. A plethora of GNN models leverage PageRank computation to propagate node features [8, 13, 22]. Since the graph Laplacian matrix for feature propagation is only applicable to undirected graphs, this line of research invokes PageRank computation algorithms only on undirected graphs.

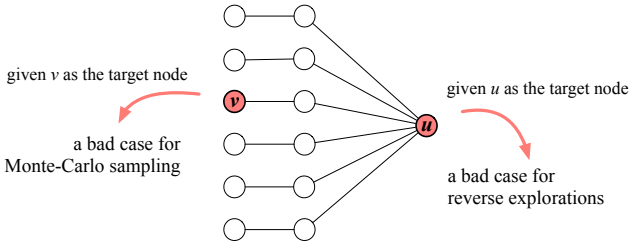

Figure 1: A special case.

Limitations of Existing Methods on Undirected Graphs. Below we briefly illustrate the limitations of existing methods for the single-node PageRank computation on undirected graphs. A simplified problem formulation is given as follows. A formal definition can be found in Section 2. Specifically, the inputs to the single-node PageRank problem are an undirected graph $G = (V, E)$ and a target node $t \in V$. The goal is to estimate the target node t's PageRank $\boldsymbol{\pi}(t)$ within a constant relative error. We also allow a constant failure probability for scalability. For the single-node PageRank computation problem, existing methods can be broadly classified into three categories:

- The Monte-Carlo method [16, 17, 27] estimate $\boldsymbol{\pi}(t)$ by repeatedly simulating α-random walks in the graph. However, according to the Pigeonhole principle, the lower bound of the required number of random walks is $\Omega(1/n)$. Thus, in the worse-case scenario where $\boldsymbol{\pi}(t) = O(1/n)$, the Monte-Carlo method requires at least $O(n)$ computational time for estimating a single node's PageRank. In Figure 1, we provide a toy example to illustrate the hard instance by regarding node v as the given target node which satisfies $\boldsymbol{\pi}(v) = \Theta(1/n)$.
- The reverse exploration method attempts to derive an estimate of $\boldsymbol{\pi}(t)$ by reversely exploring the graph from the target node t to its ancestors. A primitive operation commonly adopted in these methods is *backward push*, which deterministically pushes the probability mass initially at the target node t reversely to its ancestors step by step. Unfortunately, in each backward push operation (e.g., at node u), we at least require $O(d_u)$ time to reversely push the probability mass currently at u to every neighbor of u, where d_u denotes the degree of node u. Thus, in the worst case where $d_u = \Omega(n)$, we cost $O(n)$ time only after one step of backward push. Figure 1 provides a toy example for this bad case where $d_u = \Theta(n)$.
- The hybrid method combines the Monte-Carlo method and the reverse exploration method together. However, a simple combination cannot resolve the limitations of the Monte-Carlo and reverse exploration methods as mentioned above. In fact, despite years of efforts, the problem of computing single-node PageRank on undirected graphs has not been well solved.

Table 1: Comparison of algorithms for solving the problem of single-node PageRank computation on undirected graphs under constant relative error and failure probability. d and $d_{\max}$ denotes the average and maximum degree of graph G, respectively. The complexity results marked by ⋆ are only applicable on undirected graphs.

Query Time Complexity of Our *SetPush*	Baseline Methods	Query Time Complexities of Baseline Methods	Improvement of *SetPush* over Baselines (the larger, the better)
$\tilde{O}\left(\min\left\{d_t, \sqrt{m}\right\}\right)\star$	The Power Method [33]	$\tilde{O}(m)$	$\max\left\{m/d_t, \sqrt{m}\right\}$
	Monte-Carlo [17]	$\tilde{O}(n)$	$\max\left\{n/d_t, \sqrt{n/d}\right\}$
	LocalPush [28]	$\tilde{O}\left(\min\left\{n \cdot d_t, m\right\}\right)\star$	$\sqrt{m}$
	RBS [36]	$\tilde{O}\left(n\right)$	$\max\left\{n/d_t, \sqrt{n/d}\right\}$
	FastPPR [29]	$\tilde{O}\left(\sqrt{n \cdot d_t}\right)\star$	$\max\left\{\sqrt{n/d_t}, \sqrt{d_t/d}\right\}$
	BiPPR [26, 27]	$\tilde{O}\left(\sqrt{n \cdot d_t}\right)\star$	$\max\left\{\sqrt{n/d_t}, \sqrt{d_t/d}\right\}$
	SubgraphPush [10]	$\tilde{O}\left(\min\left\{\frac{m^{2/3} \cdot d_{\max}^{1/3}}{d^{2/3}}, \frac{m^{4/5}}{d^{3/5}}\right\}\right)$	$\max\left\{\min\left\{\frac{n^{\frac{2}{3}} \cdot d_{\max}^{1/3}}{d_t}, \frac{n^{\frac{4}{5}} \cdot d^{\frac{1}{5}}}{d_t}\right\}, \min\left\{\frac{n^{\frac{1}{6}} \cdot d_{\max}^{1/3}}{d^{\frac{1}{2}}}, \frac{n^{\frac{3}{10}}}{d^{\frac{3}{10}}}\right\}\right\}$

1.1 Our Contributions

In this paper, we consider the problem of single-node PageRank computation *on undirected graphs*. We propose a novel algorithm called *SetPush*, which achieves the $\tilde{O}\left(\min\left\{d_t, \sqrt{m}\right\}\right)$ query time complexity for the single-node PageRank computation under constant relative error and failure probability. Here m denotes the number of edges in the graph, d_t denotes the degree of the given target node t. Additionally, $\tilde{O}$ is a variant of the Big-Oh notation that ignores poly-logarithmic factors [10, 35, 36]. Detailed contributions achieved by this paper are summarized as below.

- **Theoretical Improvements.** We theoretically demonstrate the superiority of our *SetPush* over existing methods on undirected graphs. Specifically, in the last column of Table 1, we present the theoretical improvements of our *SetPush* over existing methods. In particular, the value of "Improvement" equals the query time complexity of a baseline method over that of our *SetPush*. Thus, the value of "Improvement" is the larger, the better. It's worth mentioning that the complexity results of FastPPR, BiPPR, LocalPush and our *SetPush* given in Table 1 are only applicable to undirected graphs, while the other complexities hold both on directed and undirected graphs. We observe that the expected time complexity of our *SetPush* is no worse than that of each baseline method listed in Table 1. Actually, except on a compete graph where the average node degree $d = n$, the time complexity of our *SetPush* is asymptotically better than that of every method listed in Table 1.
- **A Novel Push Operation.** The core of our *SetPush* is a novel push operation, which simultaneously mixes the deterministic backward push and randomized Monte-Carlo sampling in an atomic step. Benefit from this push operation, we cost $\Theta(d_u)$ time only at the node u with small d_u, and randomly sample a fraction of u's neighbors to push probability mass if d_u is large. As a result, we successfully remove the $O(d_u)$ term introduced by the vanilla push operation at node u, and achieve a superior time complexity over the baseline method.
- **Algorithm Development on Undirected Graphs.** Our *SetPush* algorithm is designed specifically on undirected graphs. We show that by making full use of the theoretical properties held by PageRank values on undirected graphs, we can achieve a better time complexity for single-node PageRank computation compared to existing methods on undirected graphs.

Table 2: Table of notations.

Notation	Description
$G = (V, E)$	undirected graph with vertex set V and edge set E
n, m	the numbers of nodes and edges in G
$N(u)$	the adjacency list of node u
$\mathbf{A}$	the adjacency matrix of G
d_u	the degree of node u
d	the average node degree of the graph
$d_{\max}$	the maximum node degree of the graph
$\mathbf{D}$	the diagonal degree matrix that $D_{uu} = d_u$
$\mathbf{P} = \mathbf{A}\mathbf{D}^{-1}$	the transitional probability matrix
α	the teleport probability that an α-discounted random walk terminates at each step
$\boldsymbol{\pi}(t), \hat{\boldsymbol{\pi}}(t)$	the true and estimated PageRank of node t.
$\boldsymbol{\pi}_t, \hat{\boldsymbol{\pi}}_t$	the true and estimated Personalized PageRank vectors with regard to node t.
c	constant relative error
$\tilde{O}$	the Big-Oh natation ignoring the log factors

2 PRELIMINARIES

This section introduces several basic concepts that are frequently adopted in the single-node PageRank computation. Table 2 shows the notations that are frequently used in this paper.

2.1 PageRank

Given an undirected and unweighted graph $G = (V, E)$ with n nodes and m edges, the PageRank vector $\boldsymbol{\pi}$ is an n-dimensional vector, which can be mathematically formulated as:

$$\boldsymbol{\pi} = (1-\alpha)\mathbf{A}\mathbf{D}^{-1} \cdot \boldsymbol{\pi} + \frac{\alpha}{n} \cdot \mathbf{1}. \tag{1}$$

Here $\mathbf{A}$ denotes the adjacency matrix of the graph, $\mathbf{D}$ is the diagonal degree matrix that $\mathbf{D}_{uu} = d_u$, $\mathbf{1} \in \mathbb{R}^n$ denotes an all-one vector, and α is a *constant* damping factor, which is strictly less than 1 (i.e., $\alpha \in (0, 1)$). For each node $t \in V$, we use $\boldsymbol{\pi}(t)$ to denote the PageRank value of node t. According to the definition formula given in Equation (1), the PageRank value of node t satisfies the following recurrence relation:

$$\boldsymbol{\pi}(t) = (1-\alpha) \sum_{u \in N(t)} \frac{\boldsymbol{\pi}(u)}{d_u} + \frac{\alpha}{n}, \tag{2}$$

where u is one of the neighbor of node t, and d_u denotes the degree of node u. In particular, Equation (2) also indicates a lower bound of any node's PageRank that $\boldsymbol{\pi}(t) \geq \frac{\alpha}{n}$ for each $t \in V$.

α-random walk. By the definition formula of PageRank vector $\boldsymbol{\pi}$ given in Equation (1), we can further derive:

$$\boldsymbol{\pi} = \left(\mathbf{I} - (1-\alpha)\mathbf{A}\mathbf{D}^{-1}\right)^{-1} \cdot \left(\frac{\alpha}{n} \cdot \mathbf{1}\right). \tag{3}$$

As pointed out in [25], Equation (3) can be solved using a power series expansion [4]:

$$\boldsymbol{\pi} = \sum_{i=0}^{\infty} \alpha(1-\alpha)^i \cdot (\mathbf{A}\mathbf{D}^{-1})^i \cdot \frac{1}{n} \cdot \mathbf{1}, \tag{4}$$

where $\boldsymbol{\pi}$ corresponds to a random walk probability distribution. Specifically, a random walk on the graph is a sequence of nodes $W = \{w_0, w_1, w_2, \ldots\}$ that the i-th step (i.e., the node w_i) in the walk is selected uniformly at random from the neighbor of node w_{i-1}. The PageRank value of node t equals to the probability that a so called *α-random walk* (or α-discounted random walks in some literature) [36, 37] simulated from a uniformly selected source node s terminates at node t. Note that in each step (e.g., currently at node u), an α-random walk:

- with probability $(1-\alpha)$, select a neighbor v uniformly at random from the adjacency list $N(u)$ of node u, and moves from u to v;
- with probability α, terminates at the current node u.

Therefore, the length L of an α-random walk is a geometrical random number following the geometric distribution $L \sim G(\alpha)$. The expectation of L is therefore a constant that $\mathrm{E}[L] = \frac{1}{\alpha}$.

Problem Definition. In this paper, we concern the problem of single-node PageRank computation. Specifically, given a target node t, a relative error parameter c, and a failure probability parameter p_f, we aim to derive a (c, p_f) approximation of $\boldsymbol{\pi}(t)$, which is formally defined as follows.

Definition 1 ((c, p_f)-Approximation of Single-Node PageRank). *Given a target node t in the graph $G = (V, E)$, $\hat{\boldsymbol{\pi}}(t)$ is an (c, p_f)-approximation of the single-node PageRank $\boldsymbol{\pi}(t)$ if*

$$|\hat{\boldsymbol{\pi}}(t) - \boldsymbol{\pi}(t)| \leq c \cdot \boldsymbol{\pi}(t)$$

holds with probability at least $1 - p_f$.

Note that in a line of research [10, 29, 36], c is set as a *constant* and thus is omitted in the Big-Oh notation. In this paper, we assume c is a constant following this convention. Additionally, we assume p_f is also a *constant* without loss of generality. It's worth mentioning that a constant failure probability p_f can be easily reduced to arbitrarily small with only adding a log factor to the running time by utilizing the Median-of-Mean trick [12].

2.2 Personalized PageRank

Apart from PageRank, the seminal paper [33] also propose a variant of PageRank, called Personalized PageRank (PPR), to evaluate the *personalized* centrality of graph vertices with respect to a given source node. The definition formula of PPR is analogous to that of PageRank except for the initial distribution:

$$\boldsymbol{\pi}_s = (1-\alpha)\mathbf{A}\mathbf{D}^{-1} \cdot \boldsymbol{\pi}_s + \alpha \boldsymbol{e}_s. \tag{5}$$

Specifically, $\boldsymbol{\pi}_s \in \mathcal{R}^n$ is called the single-source PPR vector, where $\boldsymbol{\pi}_s(t)$ denotes the PPR value of node t with respect to node s. $\boldsymbol{e}_s$ is an one-hot vector that $\boldsymbol{e}_s(s) = 1$ and $\boldsymbol{e}_s(u) = 0$ if $u \neq s$. Analogously, by applying the power series expansion [4], we can derive:

$$\boldsymbol{\pi}_s = \sum_{\ell=0}^{\infty} \alpha(1-\alpha)^\ell \left(\mathbf{A}\mathbf{D}^{-1}\right)^\ell \cdot \boldsymbol{e}_s. \tag{6}$$

Equation (6) provides a probabilistic interpretation on the PPR score. Specifically, the PPR value $\boldsymbol{\pi}_s(u)$ corresponds to the probability that an α-random walk generated from node s terminates at node u. Additionally, by comparing Equation (6) with Equation (4), we note that the PageRank score $\boldsymbol{\pi}(t)$ is actually an average over all $\boldsymbol{\pi}_u(t)$ for $\forall u \in V$:

$$\boldsymbol{\pi}(t) = \frac{1}{n} \cdot \sum_{s \in V} \boldsymbol{\pi}_s(t). \tag{7}$$

In particular, on undirected graphs, PPR vectors exhibit an underlying *reversibility property* that for any node-pair $(u, v) \in V^2$ [26]:

$$\boldsymbol{\pi}_u(v) \cdot d_u = \boldsymbol{\pi}_v(u) \cdot d_v. \tag{8}$$

ℓ-hop PPR. Given a source node s, a target node t and an integer $\ell \geq 0$, the ℓ-hop PPR $\boldsymbol{\pi}_s^{(\ell)}(t)$ corresponds to the probability that an α-random walk generated from node s terminates at node t exactly in its ℓ-th step. The ℓ-hop PPR vector $\boldsymbol{\pi}_s^{(\ell)}$ is defined as below.

$$\boldsymbol{\pi}_s^{(\ell)} = \alpha(1-\alpha)^\ell \cdot \left(\mathbf{A}\mathbf{D}^{-1}\right)^\ell \boldsymbol{e}_s. \tag{9}$$

By Equation (9) and Equation (4), we can thus derive $\boldsymbol{\pi}_s = \sum_{\ell=0}^{\infty} \boldsymbol{\pi}_s^{(\ell)}$. Moreover, the ℓ-hop PPR value $\boldsymbol{\pi}_s^{(\ell)}(u)$ admits the following recursive equation that for each node $v \in V$ and each integer $\ell \geq 1$:

$$\boldsymbol{\pi}_t^{(\ell+1)}(v) = \sum_{u \in N(v)} \frac{(1-\alpha)}{d_u} \cdot \boldsymbol{\pi}_t^{(\ell)}(u). \tag{10}$$

Moreover, the ℓ-hop PPR vector also exhibits the reversibility property on undirected graphs. More specifically, for every two nodes u, v in an undirected G and every $\ell \in \{0, 1, \ldots\}$, we have:

$$\boldsymbol{\pi}_s^{(\ell)}(t) \cdot d_s = \boldsymbol{\pi}_t^{(\ell)}(s) \cdot d_t. \tag{11}$$

3 ANALYSIS OF EXISTING METHODS

In this section, we present a brief review on existing approaches for single-node PageRank computation. Specifically, we classify existing methods into four categories: the power method [33], the Monte-Carlo method [17], the reverse exploration method [2, 28] and the hybrid method [10, 27, 29, 36]. Figure 2 provides a sketch to illustrate the differences among these methods.

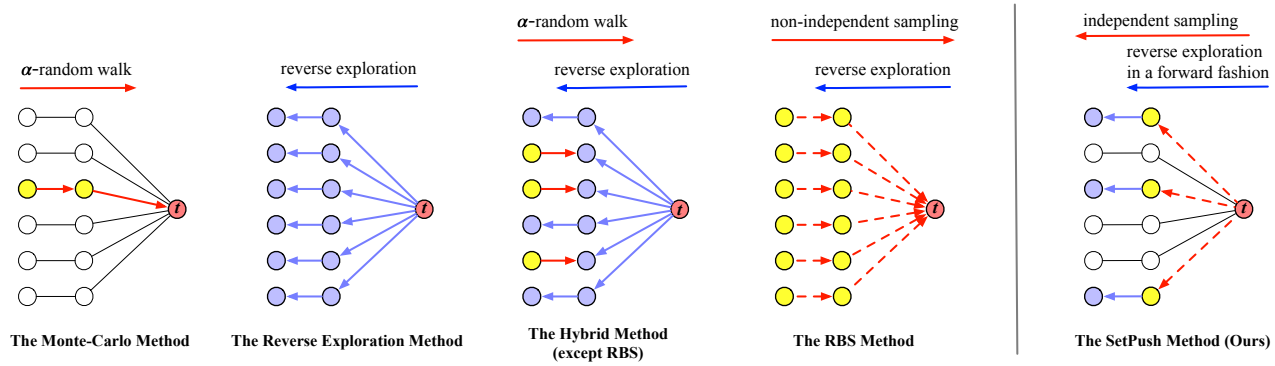

Figure 2: Comparison of existing methods.

3.1 The Power Method

The power method [33] is an iterative method for computing PageRank values of all nodes in the graph. It defines an n-dimensional vector $\hat{\boldsymbol{\pi}}$ as an approximation of the PageRank vector $\boldsymbol{\pi}$, where $\hat{\boldsymbol{\pi}}(t)$ is an estimate of node t's PageRank $\boldsymbol{\pi}(t)$. The power method initially sets $\hat{\boldsymbol{\pi}}$ as $\frac{1}{n} \cdot \mathbf{1}$, and iteratively updates $\hat{\boldsymbol{\pi}}$ according to the definition formula given in Equation (1) until $\hat{\boldsymbol{\pi}}$ merely converges. As demonstrated in [20], the convergence rate of the power method is given by $(1-\alpha)$. For the typical setting that $\alpha = 0.2$, the convergence rate of PageRank becomes 0.8, which turns out to be every fast even on large-scale graphs.

However, a major drawback of the power method is that the power method involves a multiplication between the transition matrix $\mathbf{P} = \mathbf{A}\mathbf{D}^{-1}$ and the PageRank vector $\boldsymbol{\pi}$ in each iteration. Note that $\mathbf{P}$ is an $n \times n$ matrix with m nonzero entries and $\boldsymbol{\pi}$ is an n-dimensional vector. Thus, the power method requires at least $O(m)$ time in each iteration, which is time-costly especially for single-node PageRank queries on large-scale graphs.

3.2 The Monte-Carlo Method

Recall that the PageRank score of node t equals the probability that an α-random walk simulated from a uniformly selected source node terminates at node t. Thus, the Monte-Carlo method [17] generates n_r α-random walks in the graph, where the source node of each walk is independently selected from V uniformly at random. Then the Monte-Carlo method computes $\frac{1}{n_r} \cdot \sum_{w=1}^{n_r} \mathcal{I}^{(w)}(t)$ as an estimate of $\boldsymbol{\pi}(t)$, where $\mathcal{I}^{(w)}(t)$ is an indicator variable that $\mathcal{I}^{w}(t) = 1$ if the w-th random walk terminates at node t. By the Chernoff bound, the number of α-random walks that is required to derive a (c, p_f)-approximation of $\boldsymbol{\pi}(t)$ can be bounded as $n_r = \tilde{O}\left(\frac{1}{\varepsilon^2}\right)$. Recall that the expected length L of an α-random walk is $\mathrm{E}[L] = \frac{1}{\alpha}$, which is a constant. Consequently, the expected time cost of the Monte-Carlo method for achieving the (c, p_f)-approximation of single-node PageRank is bounded by $O(n_r) = \tilde{O}(n)$.

3.3 The Reverse Exploration Method

Another line of research [2, 21, 28] computes single-node PageRank via reverse explorations. Specifically, given a target node t, this line of methods aim to estimate the contribution that each node makes to node t's PageRank. Specifically, as a well-known reverse exploration method, LocalPush [28] reversely explores the graph from the target node t to its ancestors, propagating the probability mass initially at the target node t to its neighbors step by step. To be more specific, the LocalPush method repeatedly conducts *backward push* operations, updating two variables $\boldsymbol{r}^b(v)$ and $\hat{\boldsymbol{\pi}}^b(v)$ for each node v in graph G during the query phase. In particular, $\boldsymbol{r}^b(v)$ is called the (reverse) *residue* of node v, which records the probability mass that is to be reversely pushed from node v to its ancestors. $\hat{\boldsymbol{\pi}}^b(v)$ is called the (reverse) *reserve* of v, which records the probability mass that has been received by node v so far. Initially, LocalPush sets $\boldsymbol{r}^b(v) = \hat{\boldsymbol{\pi}}^b(v) = 0$ for every $v \in V$ except $\boldsymbol{r}^b(t) = 1$. During the query phase, LocalPush repeatedly conducts the following backward push operations from all nodes v with $\boldsymbol{r}^b(v) \geq \varepsilon$. Specifically, in the backward push operation at node v, LocalPush updates $\hat{\boldsymbol{\pi}}^b(v)$ and $\boldsymbol{r}^b(v)$ as follows:

- convert α fraction of the probability mass currently at $\boldsymbol{r}^b(v)$ to its reserve: $\hat{\boldsymbol{\pi}}^b(v) \leftarrow \hat{\boldsymbol{\pi}}^b(v) + \alpha \cdot \boldsymbol{r}^b(v)$;
- reversely push the remained mass at $\boldsymbol{r}^b(v)$ to the neighbors of node v: for each $u \in N(v)$, $\boldsymbol{r}^b(u) \leftarrow \boldsymbol{r}^b(u) + (1-\alpha) \cdot \frac{\boldsymbol{r}^b(v)}{d_u}$;
- set $\boldsymbol{r}^b(v)$ as 0: $\boldsymbol{r}^b(v) \leftarrow 0$.

When no node in graph G has the residue that is larger than $\varepsilon \in (0, 1)$, the algorithm terminates. LocalPush then uses $\hat{\boldsymbol{\pi}}(t) = \frac{1}{n} \cdot \sum_{u \in V} \hat{\boldsymbol{\pi}}^b(u)$ as an estimate of $\boldsymbol{\pi}(t)$.

In particular, Lofgren et al. [28] prove that throughout the backward push process, $\hat{\boldsymbol{\pi}}^b(v)$ is always an underestimate of $\boldsymbol{\pi}_v(t)$ that $\boldsymbol{\pi}_v(t) - \hat{\boldsymbol{\pi}}^b(v) \leq \varepsilon$, where $\boldsymbol{\pi}_v(t)$ denotes the PPR of t (w.r.t node v), and $\hat{\boldsymbol{\pi}}^b(v)$ is the reserve of node v. Hence, by setting the push threshold $\varepsilon = \frac{c\alpha}{n}$, we can derive:

$$\boldsymbol{\pi}(t) - \hat{\boldsymbol{\pi}}(t) = \frac{1}{n} \sum_{v \in V} \left(\boldsymbol{\pi}_v(t) - \hat{\boldsymbol{\pi}}^b(v) \right) \leq \frac{1}{n} \sum_{v \in V} \frac{c\alpha}{n} \leq c\boldsymbol{\pi}(t) \quad (12)$$

when the LocalPush algorithm terminates. In the last inequality of Equation (12), we also adopt the lower bound $\boldsymbol{\pi}(t) \geq \frac{\alpha}{n}$, as shown in Equation (4). In other words, by setting $\varepsilon = \frac{c\alpha}{n}$, the estimate $\hat{\boldsymbol{\pi}}(t)$ derived by LocalPush is a (c, p_f)-approximation of $\boldsymbol{\pi}(t)$. Furthermore, Lofgren et al. [28] bound the worst-case time complexity of reverse exploration method as $\sum_{u \in V} \frac{\boldsymbol{\pi}_u(t) \cdot d_u}{\varepsilon}$. By plugging into $\varepsilon = \frac{c\alpha}{n}$ and the reversibility property $\boldsymbol{\pi}_u(t) \cdot d_u = \boldsymbol{\pi}_t(v) \cdot d_t$ as shown in Equation (11), we have:

$$\sum_{v \in V} \frac{\boldsymbol{\pi}_v(t) \cdot d_v \cdot n}{c\alpha} = \frac{n}{c\alpha} \cdot \left(\sum_{v \in V} \boldsymbol{\pi}_t(v) \cdot d_t \right) = \frac{n \cdot d_t}{c\alpha} = O\left(n \cdot d_t\right)$$

Note that the $O(n \cdot d_t)$ complexity may become $O(n^2)$ on some dense graphs where $d_t \to n$. To circumvent this problem, Lofgren and Goel [28] use a priority queue ordered by the residue $\boldsymbol{r}^b(v)$ of node v. Each time we pop off the node v with the greatest $\boldsymbol{r}^b(v)$ on the graph and conduct the backward push at node v. As a result, the worse-case time complexity of LocalPush is improved to $\tilde{O}(\min\{n \cdot d_t, m\})$ for deriving a (c, p_f)-approximation of $\boldsymbol{\pi}(t)$.

3.4 The Hybrid Method

Another set of papers [10, 27, 29, 36] prove some novel results by combining the Monte-Carlo method and the reverse exploration method together. The key idea is first proposed in FastPPR [29], which introduces a bi-directional approximation algorithm for single-node PageRank:

$$\boldsymbol{\pi}(t) = \sum_{v \in B(t)} \Pr\{RW(\alpha) = v\} \cdot \boldsymbol{\pi}_v(t). \tag{13}$$

Here $B(t)$ is a blanket set of the target node t that all α-random walks to node t pass through set $B(t)$. Additionally, $\Pr\{RW(\alpha) = v\}$ denotes the probability that node v is the first node in $B(t)$ hit by a randomly simulated α-random walk. FastPPR first invokes the reverse exploration method to estimate all the PPR values $\boldsymbol{\pi}_v(t)$ for $v \in V$. Then FastPPR simulates α-random walks to collect these estimators according to Equation (13). As a result, to achieve an ε-absolute error of $\boldsymbol{\pi}(t)$, FastPPR first allows an $\sqrt{\varepsilon}$ absolute error for each $\hat{\boldsymbol{\pi}}_v(t)$ derived in the reverse exploration phase, and only take $\frac{1}{\sqrt{\varepsilon}}$ α-random walks in the Monte-Carlo simulation phase. Thus, the query time complexity of FastPPR can be bounded by $\frac{1}{\alpha c^2} \cdot \sqrt{\frac{d_t \cdot n}{\alpha}} \cdot \sqrt{\frac{\log(1/p_f) \cdot \log(n/\alpha)}{\log(1/(1-\alpha))}} = \tilde{O}\left(\sqrt{n \cdot d_t}\right)$ for achieving a (c, p_f)-approximation of $\boldsymbol{\pi}(t)$. The result is subsequently improved by BiPPR [26, 27] to $\frac{1}{\alpha c} \cdot \sqrt{\frac{d_t \cdot n}{\alpha}} \cdot \sqrt{\log(1/p_f)} = \tilde{O}\left(\sqrt{n \cdot d_t}\right)$. Furthermore, Bressan et al. [10] proposed the SubgraphPush method, which optimizes the complexity result to $\tilde{O}\left(\min\left\{\frac{m^{2/3} d_{\max}^{1/3}}{d^{2/3}}, \frac{m^{4/5}}{d^{3/5}}\right\}\right)$. Here d and $d_{\max}$ denote the average and maximum degree of all the nodes in the graph, respectively.

The RBS Method. Reviewing the hybrid methods mentioned above, the Monte-Carlo sampling phase and the reverse exploration phase serve as two separate phases and are conducted sequentially. In comparison, a recent method, RBS [36], proposes to mix the two phases in a more flexible way. Specifically, the RBS method follows the framework of the reverse exploration, which reversely propagates the probability mass from the given target node t to its ancestors in the graph. The difference is, in each backward push step (e.g. at node v), the RBS method only deterministically pushes the probability mass at $\boldsymbol{r}^b(v)$ to a small fraction of v's neighbors (i.e., deterministically increase the residue of $u \in N(v)$ if the residue increment $\frac{(1-\alpha)\boldsymbol{r}^b(v)}{d_u} \geq \theta$, where θ is a threshold for deterministic push). For the other neighbors u, the RBS method generates a uniform random $rand \in (0, 1)$ and only updates the residues $\boldsymbol{r}^b(u)$ if $\frac{(1-\alpha)\boldsymbol{r}^b(v)}{d_u} \geq rand \cdot \theta$. By this means, the RBS method avoids to touch all neighbors, and successfully reduces an $O(d)$ gap between the time complexity of LocalPush [2] and the lower bound for single-target PPR queries. Here d denotes the average node degree in the graph. For the single-node PageRank computation, the expected time complexity of RBS can be bounded by $\tilde{O}(n)$ by setting $\theta = \frac{c^2 \cdot \boldsymbol{\pi}(t)}{12 \cdot \log_{1-\alpha}(c\alpha/2n)}$.

The theoretical insight introduced by RBS is encouraging, which enlightens us that we may flexibly mix the deterministic reverse exploration and the randomized Monte-Carlo sampling in each step, instead of separately performing the two phases one by one.

4 ALGORITHM

This section presents our *SetPush* algorithm. Before introducing the details, we first illustrate the reasons why existing methods are unable to achieve the $\tilde{O}\left(\min\left\{d_t, \sqrt{m}\right\}\right)$ time complexity for the single-node PageRank computation on undirected graphs.

4.1 Limitations of Existing Methods

- For the Monte-Carlo method, the lower bound of the query time complexity for deriving a (c, p_f)-approximation of $\boldsymbol{\pi}(t)$ is $\Omega\left(\frac{1}{\boldsymbol{\pi}(t)}\right)$. By the definition formula of PageRank, the initial probability distribution of simulating α-random walks is $\frac{1}{n} \cdot \mathbf{1}$. Therefore, in the worst-case scenario where $\boldsymbol{\pi}(t) = O\left(\frac{1}{n}\right)$ (e.g., the node v in Figure 1), the Monte-Carlo method needs to simulate at least $\Omega(n)$ α-random walks in order to hit node t once.
- For the reverse exploration method, we require at least $O(d_t)$ time to reversely push the probability mass initially at node t to all of its neighbors (i.e., the d_t neighbors). Consider the node u in Figure 1, where the neighborhood size of u is $O(n)$. When we conduct backward push operations from node u, the time cost has reached $\Omega(n)$ only after the first backward push operation.
- For the hybrid method, the above mentioned limitations still exist. Exceptions are the SubgraphPush [10] and RBS [36] methods.
  - The SubgraphPush method defines a blacklist to record all high-degree nodes in the graph. In the reverse exploration phase, the SubgraphPush method only performs the backward push operations from the nodes that are excluded from the blacklist. By this means, the SubgraphPush method effectively mitigates the limitations of the backward push operations as mentioned above. However, the SubgraphPush method still includes a Monte-Carlo sampling phase to simulate α-random walks from a uniformly selected source node. Thus, the lower bound of $\Omega(1/\boldsymbol{\pi}(t))$ for the query time complexity of the Monte-Carlo sampling methods still exists, which hinders the SubgraphPush method from achieving the $\tilde{O}\left(\min\left\{d_t, \sqrt{m}\right\}\right)$ time complexity for the single-node PageRank computation on undirected graphs.
  - For RBS, its major drawback comes from the sampling operation that RBS adopts in each backward push operation. Specifically, the sampling operation adopted in each backward push operation of RBS is non-independent. Consider the bad case scenario as shown in Figure 2. The residue increment $\frac{(1-\alpha)\boldsymbol{r}^b(v)}{d_u}$ of each neighbor $u \in N(t)$ is identical. As a result, for all neighbors $u \in N(t)$, the conditions to conduct a randomized push (i.e., $\frac{(1-\alpha)\boldsymbol{r}^b(v)}{d_u} \geq rand \cdot \theta$) are satisfied simultaneously, which, again, leads to the $O(d_t) = O(n)$ time cost in such bad case scenario.

In the following, we shall describe our *SetPush* in details and explain the superiority of our *SetPush* over existing methods. Specifically, we first define a concept called *truncated PageRank* in Section 4.2. Our *SetPush* is based on a $\left(\frac{c}{2}, p_f\right)$-approximation of the truncated PageRank. After that, in Section 4.3 and 4.4, we provide the high-level ideas and detailed algorithm structure of *SetPush*.

4.2 Truncated PageRank

Given a target node t in an undirected graph $G = (V, E)$, a constant damping factor $\alpha \in (0, 1)$, and a constant relative error c, we refer to $\bar{\boldsymbol{\pi}}(t)$ as the *truncated PageRank* of node t if

$$\bar{\boldsymbol{\pi}}(t) = \frac{1}{n} \cdot \sum_{s \in V} \sum_{\ell=0}^{L} \boldsymbol{\pi}_s^{(\ell)}(t), \tag{14}$$

where $L = \log_{1-\alpha} \frac{c\alpha}{2n} = O(\log n)$. Analogously, we call the n-dimensional vector $\bar{\boldsymbol{\pi}} = \frac{1}{n} \cdot \sum_{s \in V} \sum_{\ell=0}^{L} \boldsymbol{\pi}_s^{(\ell)}$ the truncated PageRank vector. By Equation (6), Equation (7) and Equation (9), we can further derive:

$$\boldsymbol{\pi} = \frac{1}{n} \cdot \sum_{s \in V} \sum_{\ell=0}^{\infty} \boldsymbol{\pi}_s^{(\ell)} = \bar{\boldsymbol{\pi}} + \frac{1}{n} \cdot \sum_{s \in V} \sum_{\ell=L+1}^{\infty} \alpha(1-\alpha)^\ell \cdot \left(\mathbf{A}\mathbf{D}^{-1}\right)^\ell \cdot \boldsymbol{e}_s.$$

Therefore, for every $t \in V$, we have:

$$\boldsymbol{\pi}(t) = \bar{\boldsymbol{\pi}}(t) + \frac{1}{n} \cdot \sum_{s \in V} \sum_{\ell=L+1}^{\infty} \alpha(1-\alpha)^\ell \cdot \boldsymbol{e}_t^\top \cdot \left(\mathbf{A}\mathbf{D}^{-1}\right)^\ell \cdot \boldsymbol{e}_s.$$

We note that for each $\ell \in \{0, 1, 2, \ldots\}$, $\left(\boldsymbol{e}_t^\top \cdot \left(\mathbf{A}\mathbf{D}^{-1}\right)^\ell \cdot \boldsymbol{e}_s\right) \in [0, 1]$. Thus, we have $\frac{1}{n} \cdot \sum_{s \in V} \boldsymbol{e}_t^\top \cdot \left(\mathbf{A}\mathbf{D}^{-1}\right)^\ell \cdot \boldsymbol{e}_s \leq 1$. As a consequence, we can derive $\boldsymbol{\pi}(t) \leq \bar{\boldsymbol{\pi}}(t) + \sum_{\ell=L+1}^{\infty} \alpha(1-\alpha)^\ell = \bar{\boldsymbol{\pi}}(t) + (1-\alpha)^{L+1}$. Recall that $L = \log_{1-\alpha} \frac{c\alpha}{2n}$. Then it follows:

$$\boldsymbol{\pi}(t) \leq \bar{\boldsymbol{\pi}}(t) + \frac{c}{2} \cdot \frac{\alpha}{n} \leq \bar{\boldsymbol{\pi}}(t) + \frac{c}{2} \cdot \boldsymbol{\pi}(t), \tag{15}$$

where we apply the lower bound of $\boldsymbol{\pi}(t)$ that $\boldsymbol{\pi}(t) \geq \frac{\alpha}{n}$ as shown in Equation (2). Furthermore, Lemma 1 implies that deriving a (c, p_f)-approximation of $\boldsymbol{\pi}(t)$ can be achieved by deriving a $(\frac{c}{2}, p_f)$-approximation of $\bar{\boldsymbol{\pi}}(t)$.

Lemma 1. *Given a target node t in the graph $G = (V, E)$, $\hat{\boldsymbol{\pi}}(t)$ is a (c, p_f)-approximation of node t's PageRank $\boldsymbol{\pi}(t)$ if*

$$|\hat{\boldsymbol{\pi}}(t) - \bar{\boldsymbol{\pi}}(t)| \leq \frac{c}{2} \cdot \boldsymbol{\pi}(t)$$

holds with probability at least $1 - p_f$.

Proof. For each node $t \in V$, we observe:

$$|\hat{\boldsymbol{\pi}}(t) - \boldsymbol{\pi}(t)| = |\hat{\boldsymbol{\pi}}(t) - \bar{\boldsymbol{\pi}}(t) + \bar{\boldsymbol{\pi}}(t) - \boldsymbol{\pi}(t)|$$
$$\leq |\hat{\boldsymbol{\pi}}(t) - \bar{\boldsymbol{\pi}}(t)| + |\bar{\boldsymbol{\pi}}(t) - \boldsymbol{\pi}(t)| \leq |\hat{\boldsymbol{\pi}}(t) - \bar{\boldsymbol{\pi}}(t)| + \frac{c}{2} \cdot \boldsymbol{\pi}(t),$$

where we plugging Equation (15) into the last inequality. Thus, if $|\hat{\boldsymbol{\pi}}(t) - \bar{\boldsymbol{\pi}}(t)| \leq \frac{c}{2} \cdot \boldsymbol{\pi}(t)$ holds with probability at least $1 - p_f$, $\hat{\boldsymbol{\pi}}(t)$ is a (c, p_f)-approximation of $\boldsymbol{\pi}(t)$, which proves the lemma. □

4.3 Key Idea of *SetPush*

Given an undirected graph $G = (V, E)$ and a target node t, our *SetPush* computes a (c, p_f)-approximation of node t's PageRank by deriving a $(c/2, p_f)$-approximation $\hat{\boldsymbol{\pi}}(t)$ of $\bar{\boldsymbol{\pi}}(t)$ following

$$\hat{\boldsymbol{\pi}}(t) = \frac{1}{n} \cdot \sum_{s \in V} \sum_{\ell=0}^{L} \frac{d_t}{d_s} \cdot \hat{\boldsymbol{\pi}}_t^{(\ell)}(s). \tag{16}$$

In particular, $\hat{\boldsymbol{\pi}}_t^{(\ell)}(s)$ is an unbiased estimator of the ℓ-hop PPR value $\boldsymbol{\pi}_t^{(\ell)}(s)$. To understand Equation (16), recall that $\boldsymbol{\pi}_t^{(\ell)}(s) \cdot d_t = \boldsymbol{\pi}_s^{(\ell)}(t) \cdot d_s$ as shown in Equation (11). Thus, if for each $s \in V$, $\hat{\boldsymbol{\pi}}_t^{(\ell)}(s)$ is an unbiased estimator of the ℓ-hop PPR value $\boldsymbol{\pi}_t^{(\ell)}(s)$, then $\frac{d_t}{d_s} \cdot \hat{\boldsymbol{\pi}}_t^{(\ell)}(s)$ is an unbiased estimator of $\hat{\boldsymbol{\pi}}_s^{(\ell)}(t)$. According to the definition formula of the truncated PageRank $\bar{\boldsymbol{\pi}}(t)$ as shown in Equation (14), $\hat{\boldsymbol{\pi}}(t)$ is therefore an unbiased estimator of $\boldsymbol{\pi}(t)$.

To compute $\hat{\boldsymbol{\pi}}_t^{(\ell)}(s)$, we maintain a variable called *ℓ-hop residue* $\boldsymbol{r}_t^{(\ell)}(u)$ for each node u in G. Initially, we set $\boldsymbol{r}_t^{(\ell)} = \mathbf{0}$ for $\forall \ell \in \{1, 2, \ldots, L\}$ and $\boldsymbol{r}_t^{(0)} = \boldsymbol{e}_t$, where $\mathbf{0}$ is an n-dimensional all zero vector. During the query phase, we repeatedly conduct the following steps to update $\boldsymbol{r}_t^{(\ell+1)}$ based on $\boldsymbol{r}_t^{(\ell)}$ by iterating ℓ from 0 to $L-1$:

- Pick a node u with nonzero $\boldsymbol{r}_t^{(\ell)}(u)$;
- If $(1-\alpha) \cdot \boldsymbol{r}_t^{(\ell)}(u) \geq \theta \cdot d_u$, we uniformly distribute $(1-\alpha) \cdot \boldsymbol{r}_t^{(\ell)}(u)$ to the $(\ell+1)$-hop residue $\boldsymbol{r}_t^{(\ell+1)}(v)$ of each $v \in N(u)$. To be more specific, for $\forall v \in N(u)$, $\boldsymbol{r}_t^{(\ell+1)}(v) \leftarrow \boldsymbol{r}_t^{(\ell+1)}(v) + \frac{(1-\alpha)}{d_u} \cdot \boldsymbol{r}_t^{(\ell)}(u)$. Note that $\theta \in (0, 1)$ is a tunable threshold and we provide a detailed analysis to the choice of θ in Section 5.
- Otherwise, we independently select some neighbors of u, and only distribute the probability mass at $\boldsymbol{r}_t^{(\ell)}(u)$ to those sampled neighbors. Notably, for each $v \in N(u)$, the expectation of $\boldsymbol{r}_t^{(\ell+1)}(v)$'s increment is still guaranteed to be $\frac{(1-\alpha)}{d_u} \cdot \boldsymbol{r}_t^{(\ell)}(u)$.

After all the L iterations have been processed, we return $\hat{\boldsymbol{\pi}}(t) = \frac{1}{n} \cdot \sum_{s \in V} \sum_{\ell=0}^{L} \frac{d_t}{d_s} \cdot \alpha \cdot \boldsymbol{r}_t^{(\ell)}(s)$ as an estimator of $\boldsymbol{\pi}(t)$.

As we shall demonstrate in Section 5, the ℓ-hop residue vector $\boldsymbol{r}_t^{(\ell)}$ is an unbiased estimate of $\frac{1}{\alpha} \cdot \boldsymbol{\pi}_t^{(\ell)}$. In other words, $\mathrm{E}\left[\boldsymbol{r}_t^{(\ell)}(u)\right] = \frac{1}{\alpha} \cdot \boldsymbol{\pi}_t^{(\ell)}(u)$ holds for each $u \in V$. To see this, we observe that $\boldsymbol{\pi}_t^{(0)} = \alpha \cdot \boldsymbol{e}_t$ holds by definition. Recall that we set $\boldsymbol{r}_t^{(0)} = \boldsymbol{e}_t$ as mentioned above. Therefore, $\mathrm{E}\left[\boldsymbol{r}_t^{(\ell)}\right] = \frac{1}{\alpha} \cdot \boldsymbol{\pi}_t^{(\ell)}$ holds when $\ell = 0$. Furthermore, let us assume $\boldsymbol{r}_t^{(\ell)} = \frac{1}{\alpha} \cdot \boldsymbol{\pi}_t^{(\ell)}$ holds for any $i \in [0, \ell]$. Then for each $v \in V$, the expectation of $\boldsymbol{r}_t^{(\ell+1)}(v)$ satisfies:

$$\mathrm{E}\left[\boldsymbol{r}_t^{(\ell+1)}(v)\right] = \sum_{u \in N(v)} \frac{(1-\alpha)}{d_u} \cdot \mathrm{E}\left[\boldsymbol{r}_t^{(\ell)}(u)\right] = \sum_{u \in N(v)} \frac{(1-\alpha)}{d_u} \cdot \frac{\boldsymbol{\pi}_t^{(\ell)}(u)}{\alpha}.$$

By Equation (10), we can therefore derive $\mathrm{E}\left[\boldsymbol{r}_t^{(\ell+1)}(v)\right] = \frac{1}{\alpha} \cdot \boldsymbol{\pi}_t^{(\ell+1)}(v)$. Consequently, for every $\ell \in \{1, \ldots, L\}$, $\mathrm{E}\left[\boldsymbol{r}_t^{(\ell)}\right] = \frac{1}{\alpha} \cdot \boldsymbol{\pi}_t^{(\ell)}$ holds by induction. The formal proof can be found in Section 5.

Furthermore, it can be proved that $\hat{\boldsymbol{\pi}}(t)$ is also an unbiased estimator of the truncated PageRank $\bar{\boldsymbol{\pi}}(t)$. Specifically, recall that $\hat{\boldsymbol{\pi}}(t) = \frac{1}{n} \cdot \sum_{s \in V} \sum_{\ell=0}^{L} \frac{d_t}{d_s} \cdot \alpha \cdot \boldsymbol{r}_t^{(\ell)}(s)$ according to Algorithm 1. By

applying the linearity of expectation, we can thus derive

$$\mathrm{E}\left[\hat{\boldsymbol{\pi}}(t)\right] = \frac{1}{n} \cdot \sum_{s \in V} \sum_{\ell=0}^{L} \frac{d_t}{d_s} \cdot \alpha \cdot \mathrm{E}\left[\boldsymbol{r}_t^{(\ell)}(s)\right] = \frac{1}{n} \cdot \sum_{s \in V} \sum_{\ell=0}^{L} \frac{d_t}{d_s} \cdot \boldsymbol{\pi}_t^{(\ell)}(s).$$

Recall that in Equation (11), we show that $\frac{d_t}{d_s} \cdot \boldsymbol{\pi}_t^{(\ell)}(s) = \boldsymbol{\pi}_s^{(\ell)}(t)$, following $\mathrm{E}\left[\hat{\boldsymbol{\pi}}(t)\right] = \frac{1}{n} \cdot \sum_{s \in V} \sum_{\ell=0}^{L} \boldsymbol{\pi}_s^{(\ell)}(t) = \bar{\boldsymbol{\pi}}(t)$.

Advantages of the Push Operation Adopted in *SetPush*. Note that the ℓ-hop residue $\boldsymbol{r}_t^{(\ell)}(u)$ defined above is similar in spirit to the one used in the vanilla backward push operation adopted in the reverse exploration method (see Section 3.3), but differs in two crucial aspects as described below.

- To distribute the probability mass maintained at $\boldsymbol{r}_t^{(\ell)}(u)$, the backward push operation (except in RBS [36]) touches every neighbor v of u to update the residue of v, which costs $O(d_u)$ deterministically. In comparison, for the node u with $(1-\alpha) \cdot \boldsymbol{r}_t^{(\ell)}(u) \leq \theta \cdot d_u$, we only select some neighbors $v \in N(u)$ to update $\boldsymbol{r}_t^{(\ell+1)}(v)$. Therefore, the time cost of each update process is only proportional to the size of the sampled outcomes. By this means, we successfully avoid the $O(d_u)$ term of time complexity introduced by the vanilla backward push.
- Compared to the RBS method, we independently sample the neighbors v from $N(u)$ to update $\boldsymbol{r}_t^{(\ell+1)}(v)$. As a consequence, the increment of $\boldsymbol{r}_t^{(\ell+1)}(v)$ for each $v \in V$ is independent with each other. In contrast, the sampling technique adopted in the RBS method [36] is non-independent, resulting in either large variance or expensive time cost. For example, consider the graph shown in Figure 2 with node t as the given target node. For the RBS method, the sampling condition of each $u \in N(t)$ is satisfied simultaneously, which costs either $O(n)$ time or unbounded approximation error. Instead, in *SetPush*, we can independently some $u \in N(t)$ to update $\boldsymbol{r}_t^{(\ell)}(u)$.

4.4 The *SetPush* Algorithm

Algorithm 1 illustrates the pseudocode of *SetPush*. Consider an undirected graph $G = (V, E)$, a target node t, a constant damping factor $\alpha \in (0, 1)$ and a threshold parameter $\theta \in (0, 1)$. Initially, we set $\boldsymbol{r}_t^{(0)} = \boldsymbol{e}_t$ and iteratively conduct the update process as described in Section 4.3 from $\ell = 0$ to $L-1$, where $L = \log_{1-\alpha} \frac{c\alpha}{2n}$. In particular, for the node u with $0 < (1-\alpha) \cdot \boldsymbol{r}_t^{(\ell)}(u) \leq \theta \cdot d_u$, we adopt a *geometric sampling operation* to independently select neighbors v from $N(u)$. Specifically, we independently sample every $v \in N(u)$ with probability $p^* = \frac{(1-\alpha) \cdot \boldsymbol{r}_t^{(\ell)}(u)}{d_u \cdot \theta}$. For each sampled $v \in N(u)$, we increase the residue $\boldsymbol{r}_t^{(\ell+1)}(v)$ by θ. By this means, the expectation of $\boldsymbol{r}_t^{(\ell+1)}(v)$'s increment is still $\frac{(1-\alpha)}{d_u} \cdot \boldsymbol{r}_t^{(\ell)}(u)$. It's worth noting that we aim to complete the above described sampling process using the time of $O(d_u \cdot p^*)$. In other words, we require the expected time cost of the above described sampling process is asymptotically the same to the expected size of the sampling outcomes (i.e., the expected number of u's neighbors that are successfully sampled). To achieve this goal, we define a variable *idx* for referring to the index of u's neighbor in $N(u)$ that is successfully sampled. Initially, we set *idx* as 0. Moreover, we define a geometric random number rg,

Algorithm 1: The *SetPush* Algorithm

Input: Undirected graph $G = (V, E)$, target node $t \in V$, constant damping factor α, threshold θ
Output: Estimator of $\boldsymbol{\pi}(t)$

Initialize two n-dimensional vectors $\boldsymbol{r}_t^{(0)} \leftarrow \boldsymbol{e}_t$ and $\hat{\boldsymbol{\pi}}_t \leftarrow \alpha \boldsymbol{e}_t$;
$L \leftarrow \log_{1-\alpha} \frac{c\alpha}{2n}$;
for ℓ *from* 0 *to* $L-1$ **do**
  Initialize an n-dimensional vector $\boldsymbol{r}_t^{(\ell+1)} \leftarrow \boldsymbol{0}$;
  for *each* $u \in V$ *with nonzero* $\boldsymbol{r}_t^{(\ell)}(u)$ **do**
    if $(1-\alpha) \cdot \boldsymbol{r}_t^{(\ell)}(u) \geq \theta \cdot d_u$ **then**
      for *each* $v \in N(u)$ **do**
        $\boldsymbol{r}_t^{(\ell+1)}(v) \leftarrow \boldsymbol{r}_t^{(\ell+1)}(v) + \frac{(1-\alpha)}{d_u} \cdot \boldsymbol{r}_t^{(\ell)}(u)$;
    else
      Let $idx \leftarrow 0$, and $p^* \leftarrow \frac{(1-\alpha) \cdot \boldsymbol{r}_t^{(\ell)}(u)}{d_u \cdot \theta}$;
      while *true* **do**
        Generate a geometrical random $rg \sim G(p^*)$;
        $idx \leftarrow idx + rg$;
        if $idx > d_u$ **then**
          break;
        Let v denote the idx-th node in $N(u)$;
        $\boldsymbol{r}_t^{(\ell+1)}(v) \leftarrow \boldsymbol{r}_t^{(\ell+1)}(v) + \theta$;
  Clear $\boldsymbol{r}_t^{(\ell)}$;
  $\hat{\boldsymbol{\pi}}_t \leftarrow \hat{\boldsymbol{\pi}}_t + \alpha \cdot \boldsymbol{r}_t^{(\ell+1)}$;
$\hat{\boldsymbol{\pi}}(t) \leftarrow \frac{1}{n} \cdot \sum_{s \in V} \frac{d_t}{d_s} \cdot \hat{\boldsymbol{\pi}}_t(s)$;
return $\hat{\boldsymbol{\pi}}(t)$ as an estimator of $\boldsymbol{\pi}(t)$;

and repeatedly generate rg according to the geometric distribution $G(p^*)$. According to [11, 15], a geometric random number can be generated in $O(1)$ time. We repeatedly generate $rg \sim G(p^*)$, update $idx \leftarrow idx + rg$ and increase the residue $\boldsymbol{r}_t^{(\ell+1)}(v)$ of the idx-th neighbor v in $N(u)$ by θ, until $idx > d_u$.

To understand the sampling process mentioned above, recall that a geometric random number $rg \sim G(p^*)$ indicates the number of Bernoulli trials needed to get one success, where each Bernoulli trial has two Boolean-valued outcomes: success (with probability p^*) and failure (with probability $1-p^*$). Therefore, by generating $rg \sim G(p^*)$, we are able to derive the index of the first sampled node in $N(u)$, using only $O(1)$ time. We iteratively generate $rg \sim G(p^*)$ to derive the index of the next sampled node from the index of the last sampled neighbor (recorded by *idx*). By this means, we are able to independently select each neighbor v from $N(u)$ with probability p^* using only $O(d_u \cdot p^*) = O\left(\frac{(1-\alpha) \cdot \boldsymbol{r}_t^{(\ell)}(u)}{\theta}\right)$ time in expectation. By carefully setting the value of θ (see Section 5 for details), the expected time cost of *SetPush* can be consequently bounded by $O\left(\min\left\{d_t, \sqrt{m}\right\}\right)$. Additionally, after the ℓ-th iteration ($\forall \ell \in \{0, 1, \ldots, L-1\}$), we clear the ℓ-hop residue vector $\boldsymbol{r}_t^{(\ell)}$ to save memory. Finally, we return $\hat{\boldsymbol{\pi}}(t) = \frac{1}{n} \cdot \sum_{s \in V} \sum_{\ell=0}^{L} \frac{d_t}{d_s} \cdot \alpha \boldsymbol{r}_t^{(\ell)}(s)$ as the estimator of $\boldsymbol{\pi}(t)$.

5 THEORETICAL ANALYSIS

In this section, we analyze the theoretical properties of our *SetPush*.

5.1 Correctness

Recall that we have presented some intuitions on $\mathrm{E}\left[\boldsymbol{r}_t^{(\ell)}(u)\right] = \frac{1}{\alpha} \cdot \boldsymbol{\pi}_t^{(\ell)}(u)$ and $\mathrm{E}\left[\hat{\boldsymbol{\pi}}(t)\right] = \bar{\boldsymbol{\pi}}(t)$ in Section 4.3. The following Lemmas further provide formal proofs on these intuitions. Due to the page limit, we defer the proofs of Lemma 2 and Lemma 3 to our Technical Report [1].

Lemma 2. *For each $\ell \in \{0, 1, \ldots, L\}$ The residue vector $\boldsymbol{r}_t^{(\ell)}$ obtained in Algorithm 1 is an unbiased estimator of $\frac{1}{\alpha} \cdot \boldsymbol{\pi}_t^{(\ell)}$, such that for each $v \in V$,*

$$\mathrm{E}\left[\boldsymbol{r}_t^{(\ell)}(v)\right] = \frac{1}{\alpha} \cdot \boldsymbol{\pi}_t^{(\ell)}(v).$$

Based on Lemma 2, we are able to prove that Algorithm 1 returns an unbiased estimator of the truncated PageRank $\bar{\boldsymbol{\pi}}(t)$.

Lemma 3. *Algorithm 1 returns an unbiased estimator $\hat{\boldsymbol{\pi}}(t)$ of the truncated PageRank score of node t. Specifically,* $\mathrm{E}[\hat{\boldsymbol{\pi}}(t)] = \bar{\boldsymbol{\pi}}(t)$.

Up to now, we have proved that $\hat{\boldsymbol{\pi}}(t)$ is an unbiased estimator of the truncated PageRank $\boldsymbol{\pi}(t)$. Next, we shall bound the variance of $\hat{\boldsymbol{\pi}}(t)$ and utilize the following Chebyshev Inequality [30] to bound the failure probability for deriving a $(\frac{c}{2}, p_f)$-approximation of $\bar{\boldsymbol{\pi}}(t)$.

Fact 1 (Chebyshev's Inequality [30]). *Let X denote a random variable. For any real number $\varepsilon > 0$,* $\Pr\{|X - \mathrm{E}[X]| \geq \varepsilon\} \leq \frac{\mathrm{Var}[X]}{\varepsilon^2}$.

5.2 Variance Analysis

We claim that the variance of $\hat{\boldsymbol{\pi}}(t)$ can be bounded by $\frac{L\theta d_t}{n} \cdot \boldsymbol{\pi}(t)$, which is formally demonstrated in Theorem 1.

Theorem 1 (Variance). *The variance of the estimator $\hat{\boldsymbol{\pi}}(t)$ returned by Algorithm 1 can be bounded as* $\mathrm{Var}[\hat{\boldsymbol{\pi}}(t)] \leq \frac{L\theta d_t}{n} \cdot \boldsymbol{\pi}(t)$.

To prove Theorem 1, we need several technical lemmas. Specifically, in Lemma 4, we bound the variance of $\boldsymbol{r}_t^{(\ell+1)}(v)$ conditioned on $\boldsymbol{r}_t^{(\ell)}$ that is derived in the ℓ-th iteration.

Lemma 4. *For each node $v \in V$ and each $\ell \in \{0, 1, \ldots, L-1\}$, the variance of $\boldsymbol{r}_t^{(\ell+1)}(v)$ can be bounded as*

$$\mathrm{Var}\left[\boldsymbol{r}_t^{(\ell+1)}(v) \,\middle|\, \boldsymbol{r}_t^{(\ell)}\right] \leq \sum_{u \in N(v)} \theta \cdot \frac{(1-\alpha) \cdot \boldsymbol{r}_t^{(\ell)}(u)}{d_u},$$

where $\mathrm{Var}\left[\boldsymbol{r}_t^{(\ell+1)}(v) \,\middle|\, \boldsymbol{r}_t^{(\ell)}\right]$ denotes the variance of $\boldsymbol{r}_t^{(\ell+1)}(v)$ conditioned on the value of $\boldsymbol{r}_t^{(\ell)}$ that has been derived in the ℓ-th iteration.

In the second step, we prove:

Lemma 5. *The variance of the estimator $\hat{\boldsymbol{\pi}}(t)$ obtained by Algorithm 1 can be computed as:*

$$\mathrm{Var}\left[\hat{\boldsymbol{\pi}}(t)\right] = \frac{\alpha^2}{n^2} \cdot \mathrm{Var}\left[\sum_{\ell=0}^{L} \sum_{s \in V} \frac{d_t}{d_s} \cdot \boldsymbol{r}_t^{(\ell)}(s)\right]$$

$$= \frac{\alpha^2}{n^2} \cdot \sum_{\ell=0}^{L-2} \mathrm{E}\left[\mathrm{Var}\left[\sum_{v \in V}\left(\sum_{s \in V} \frac{d_t}{d_s} \cdot \sum_{i=0}^{L-\ell-1} \frac{\boldsymbol{\pi}_v^{(i)}(s)}{\alpha}\right) \cdot \boldsymbol{r}_t^{(\ell+1)}(v) \,\middle|\, \boldsymbol{r}_t^{(\ell)}\right]\right].$$

To prove Lemma 5, recall that $\hat{\boldsymbol{\pi}}(t) = \frac{1}{n} \sum_{s \in V} \frac{d_t}{d_s} \cdot \hat{\boldsymbol{\pi}}_t(s)$, and $\hat{\boldsymbol{\pi}}_t(s) = \sum_{\ell=0}^{L} \alpha \boldsymbol{r}_t^{(\ell)}(s)$ according to Algorithm 1. Thus, the variance of $\hat{\boldsymbol{\pi}}(t)$ derived by Algorithm 1 can be computed as:

$$\mathrm{Var}[\hat{\boldsymbol{\pi}}(t)] = \mathrm{Var}\left[\frac{1}{n} \cdot \sum_{s \in V} \frac{d_t}{d_s} \cdot \sum_{\ell=0}^{L} \alpha \cdot \boldsymbol{r}_t^{(\ell)}(s)\right] = \frac{\alpha^2}{n^2} \cdot \mathrm{Var}\left[\sum_{s \in V} \frac{d_t}{d_s} \cdot \sum_{\ell=0}^{L} \boldsymbol{r}_t^{(\ell)}(s)\right],$$

For the second equality in Lemma 5, we can prove it by repeatedly applying the law of total variance. Details of the law of total variance are given as below.

Fact 2 (Law of Total Variance [40]). *For two random variables X and Y, the law of total variance states:*

$$\mathrm{Var}[Y] = \mathrm{E}[\mathrm{Var}[Y \mid X]] + \mathrm{Var}[\mathrm{E}[Y \mid X]]$$

holds if the two variables X and Y are on the same probability space and the variance of Y is finite.

Furthermore, we plug the variance bound derived in Lemma 4 into Lemma 5, which follows Lemma 6.

Lemma 6. *For all $\ell \in [0, L]$, the residue vectors $\boldsymbol{r}^{(\ell)}$ obtained by Algorithm 1 in the ℓ-th iterations satisfy:*

$$\sum_{\ell=1}^{L-1} \mathrm{E}\left[\mathrm{Var}\left[\sum_{v \in V}\left(\sum_{s \in V} \frac{d_t}{d_s} \cdot \sum_{i=0}^{L-\ell} \frac{\boldsymbol{\pi}_v^{(i)}(s)}{\alpha}\right) \cdot \boldsymbol{r}_t^{(\ell)}(v) \,\middle|\, \boldsymbol{r}_t^{(\ell-1)}\right]\right] \leq \frac{L\theta d_t \cdot n\boldsymbol{\pi}(t)}{\alpha^2}.$$

Finally, by putting Lemma 5 and 6 together, we can conclude that $\mathrm{Var}[\hat{\boldsymbol{\pi}}(t)] \leq \frac{L \cdot \theta \cdot d_t}{n^2} \cdot n\hat{\boldsymbol{\pi}}(t)$, following Theorem 1. Detailed proofs of the above theorem and lemmas can be found in the Technical Report [1] due to the page limit of the main text.

5.3 Time Cost

In the following, we analyze the expected time cost of the *SetPush* algorithm. Moreover, Theorem 2 provides the theoretical guarantees of the *SetPush* algorithm for achieving a (c, p_f)-approximation of the single-node PageRank.

Lemma 7. *The expected time cost of Algorithm 1 can be bounded by* $\frac{1}{\alpha\theta} = O\left(\frac{1}{\theta}\right)$.

Proof. Let $Cost^{(\ell+1)}(u, v)$ denote the time cost of increasing $\boldsymbol{r}_t^{(\ell+1)}(v)$ during the update process conducted at node u with nonzero $\boldsymbol{r}_t^{(\ell)}(u)$. According to Algorithm 1, $Cost^{(\ell+1)}(u, v) = 1$ holds deterministically if $\frac{(1-\alpha)}{d_u} \cdot \boldsymbol{r}_t^{(\ell)}(u) \geq \theta$. On the other hand, if $\frac{(1-\alpha)}{d_u} \cdot \boldsymbol{r}_t^{(\ell)}(u) < \theta$, $Cost^{(\ell+1)}(u, v) = 1$ (i.e., pushing probability mass from node u to v) holds with probability $\frac{(1-\alpha)}{d_u \cdot \theta} \cdot \boldsymbol{r}_t^{(\ell)}(u)$, or $Cost^{(\ell+1)}(u, v) = 0$ holds with probability $1 - \frac{(1-\alpha)}{d_u \cdot \theta} \cdot \boldsymbol{r}_t^{(\ell)}(u)$. Thus, given the ℓ-hop residue vector $\boldsymbol{r}_t^{(\ell)}$, the expectation of $Cost^{(\ell+1)}(u, v)$ can be bounded as:

$$\mathrm{E}\left[Cost^{(\ell+1)}(u, v) \,\middle|\, \boldsymbol{r}_t^{(\ell)}\right] \leq 1 \cdot \frac{(1-\alpha)}{d_u \cdot \theta} \cdot \boldsymbol{r}_t^{(\ell)}(u).$$

Furthermore, let $Cost^{(\ell+1)}$ denote the time cost of updating the $(\ell+1)$-hop residue vector $\boldsymbol{r}_t^{(\ell+1)}$ based on the ℓ-hop residue vector

$\boldsymbol{r}_t^{(\ell)}$. Then we have $Cost^{(\ell+1)} = \sum_{(u,v)\in E} Cost^{(\ell+1)}(u,v)$. It follows:

$$\mathrm{E}\left[Cost^{(\ell+1)} \mid \boldsymbol{r}_t^{(\ell)}\right] = \sum_{(u,v)\in E} \mathrm{E}\left[Cost^{(\ell+1)}(u,v) \mid \boldsymbol{r}_t^{(\ell)}\right] = \sum_{(u,v)\in E} \frac{(1-\alpha)}{d_u\cdot\theta}\cdot \boldsymbol{r}_t^{(\ell)}(u).$$

By the property of expectation, we further have:

$$\mathrm{E}\left[Cost^{(\ell+1)}\right] = \mathrm{E}\left[\mathrm{E}\left[Cost^{(\ell+1)} \mid \boldsymbol{r}_t^{(\ell)}\right]\right] = \sum_{(u,v)\in E} \frac{(1-\alpha)}{d_u\cdot\theta}\cdot \mathrm{E}\left[\boldsymbol{r}_t^{(\ell)}(u)\right]$$

$$= \frac{1}{\alpha\theta}\cdot \sum_{v\in V}\sum_{u\in N(v)} \frac{(1-\alpha)}{d_u}\cdot \boldsymbol{\pi}_t^{(\ell)}(u) = \frac{1}{\alpha\theta}\cdot \sum_{v\in V} \boldsymbol{\pi}_t^{(\ell+1)}(v),$$

where we apply Lemma 2 in the third equality given above. We also apply Equation (10) in the last equality as shown above. Furthermore, let $Cost = \sum_{\ell=0}^{L-1} Cost^{(\ell+1)}$ denote the total time cost of Algorithm 1. Thus, we can derive:

$$\mathrm{E}\left[Cost\right] = \sum_{\ell=0}^{L-1} \mathrm{E}\left[Cost^{(\ell+1)}\right] = \frac{1}{\alpha\theta}\cdot \sum_{v\in V}\sum_{\ell=0}^{L-1} \boldsymbol{\pi}_t^{(\ell+1)}(v) \le \frac{1}{\theta} = O\left(\frac{1}{\theta}\right),$$

by applying $\sum_{\ell=0}^{L-1} \boldsymbol{\pi}_t^{(\ell+1)}(v) \le \boldsymbol{\pi}_t(v)$, and $\sum_{v\in V} \boldsymbol{\pi}_t^{(\ell+1)}(v) = \alpha$. Therefore, the lemma follows. □

In the end, we employ the bound of variance $\mathrm{Var}\left[\hat{\boldsymbol{\pi}}(t)\right]$ derived in Theorem 1 to the Chebyshev's Inequality given in Fact 1, to derive an appropriate setting of the threshold θ.

Theorem 2. *By setting* $\theta = \max\left\{\frac{\alpha c^2}{12L\cdot d_t}, \frac{\alpha c^2}{12L}\cdot\sqrt{\frac{2(1-\alpha)}{m}}\right\}$, *Algorithm 1 returns a* (c, p_f)*-approximation* $\hat{\boldsymbol{\pi}}(t)$ *of* $\boldsymbol{\pi}(t)$, *such that* $|\boldsymbol{\pi}(t) - \hat{\boldsymbol{\pi}}(t)| \le c\cdot\boldsymbol{\pi}(t)$ *holds with constant probability. The expected time cost of Algorithm 1 is bounded by*

$$\frac{12\cdot\left(\log_{1-\alpha}\frac{c\alpha}{2n}\right)}{\alpha^2c^2}\cdot\min\left\{d_t, \sqrt{\frac{m}{2(1-\alpha)}}\right\} = \tilde{O}\left(\min\left\{d_t, \sqrt{m}\right\}\right).$$

Proof. Recall that the variance of $\hat{\boldsymbol{\pi}}(t)$ obtained by Algorithm 1 is bounded by $\frac{L\cdot\theta\cdot d_t}{n}\cdot\hat{\boldsymbol{\pi}}(t)$ as shown in Theorem 1. Plugging into the Chebyshev's Inequality, we can further derive:

$$\Pr\left\{\hat{\boldsymbol{\pi}}(t) - \bar{\boldsymbol{\pi}}(t) \ge \frac{c}{2}\cdot\boldsymbol{\pi}(t)\right\} \le \frac{4\cdot\mathrm{Var}\left[\hat{\boldsymbol{\pi}}(t)\right]}{c^2\cdot(\boldsymbol{\pi}(t))^2} \le \frac{4L\theta d_t}{c^2\cdot n\boldsymbol{\pi}(t)}.$$

Thus, by setting $\theta = \frac{c^2\cdot p_f\cdot n\boldsymbol{\pi}(t)}{4Ld_t}$, $\hat{\boldsymbol{\pi}}(t) - \bar{\boldsymbol{\pi}}(t) \le \frac{c}{2}\cdot\boldsymbol{\pi}(t)$ holds with probability at least p_f. In particular, we note $\frac{c^2\cdot p_f\cdot n\boldsymbol{\pi}(t)}{4Ld_t} \ge \frac{\alpha c^2\cdot p_f}{4Ld_t}$ based on the fact that $\boldsymbol{\pi}(t) \ge \frac{\alpha}{n}$ as illustrated in Equation (2). If we set $\theta = \frac{\alpha c^2\cdot p_f}{4Ld_t}$, then according to Lemma 7, the expected time cost of Algorithm 1 can be bounded by $\frac{1}{\alpha\theta} = \frac{4Ld_t}{\alpha^2c^2\cdot p_f} = \tilde{O}(d_t)$, where α, c, p_f are all constants, and $L = \log_{1-\alpha}\frac{c\alpha}{2n}$ (see Section 4.2 for the details of setting L). Moreover, as we shall prove below, $\frac{n\boldsymbol{\pi}(t)}{d_t} \ge \alpha\cdot\sqrt{\frac{2(1-\alpha)}{m}}$ holds for any $t \in V$. Thus, by setting $\theta = \frac{\alpha c^2\cdot p_f}{4L}\cdot\sqrt{\frac{2(1-\alpha)}{m}}$, the expected time cost of Algorithm 1 is bounded by $\frac{1}{\alpha\theta} = \frac{4L}{\alpha^2c^2\cdot p_f}\cdot\sqrt{\frac{m}{2(1-\alpha)}} = \tilde{O}(\sqrt{m})$.

Table 3: Datasets

Dataset	n	m	m/n
Youtube(YT)	1,138,499	5,980,886	5.25
IndoChina (IC)	7,414,768	301,969,638	40.73
Orkut-Links (OL)	3,072,441	234,369,798	76.28
Friendster (FR)	68,349,466	3,623,698,684	53.02

Now we present the proof of $\frac{n\boldsymbol{\pi}(t)}{d_t} \ge \alpha\cdot\sqrt{\frac{2(1-\alpha)}{m}}$. By Equation (2), we have:

$$\boldsymbol{\pi}(t) \ge (1-\alpha)\sum_{u\in N(t)}\frac{\boldsymbol{\pi}(u)}{d_u} + \frac{\alpha}{n} \ge (1-\alpha)\sum_{u\in N(t)}\frac{1}{d_u}\cdot\frac{\alpha}{n} + \frac{\alpha}{n}. \quad (17)$$

We note $\sum_{u\in N(t)}\frac{1}{d_u} \ge \frac{d_t^2}{2m}$ since $\left(\sum_{u\in N(t)}\frac{1}{d_u}\right)\cdot\left(\sum_{u\in N(t)} d_u\right) \ge \left(\sum_{u\in N(t)} 1\right)^2 = d_t^2$ holds by the Cauchy-Schwarz Inequality [34]. Plugging into Inequality (17), we can further derive:

$$\boldsymbol{\pi}(t) \ge \frac{\alpha}{n}\cdot\left(\frac{(1-\alpha)d_t^2}{2m} + 1\right) = \frac{\alpha d_t}{n}\cdot\left(\frac{(1-\alpha)d_t + \frac{2m}{d_t}}{2m}\right) \ge \frac{\alpha d_t}{n}\cdot\sqrt{\frac{2(1-\alpha)}{m}},$$

where we apply the fact that $(1-\alpha)d_t + \frac{2m}{d_t} \ge 2\cdot\sqrt{(1-\alpha)2m}$ by the AM-GM Inequality. Consequently, $\frac{n\boldsymbol{\pi}(t)}{d_t} \ge \alpha\cdot\sqrt{\frac{2(1-\alpha)}{m}}$ holds for each $t \in V$, and the theorem follows. □

6 EXPERIMENTS

This section presents the empirical results of *SetPush*. All experiments are conducted on a machine with an Intel(R) Xeon(R) Gold 6126@2.60GHz CPU and 500GB memory with the Linux OS. We implement all algorithms in C++ compiled by g++ with the O3 optimization turned on.

Datasets. We use four large-scale real-world datasets in the experiments [1] [2], including Youtube (YT), IndoChina (IC), Orkut-Links (OL) and Friendster (FR). The Youtube, Orkut-Links and Friendster datasets are all originated from social networks, where the nodes in the graph correspond to the users in the website, and edges indicates friendship between users. Additionally, the IndoChina is a web dataset for the country domains in Indochina. We summarize the statistics of all the datasets in Table 3.

Query Sets. We generate two sets of query nodes, denoted as Q_1 and Q_2, in the experiments. First, for the Q_1 query set, we select 10 nodes from the graph's vertex set V uniformly at random. For the second query set Q_2, we select 10 query nodes from V according to the node degree distribution. The larger the node's degree is, the more likely the node is selected into Q_2. Note that the PageRank distribution of a real-world network is experimentally observed to follow the power-law distribution [5, 27, 38, 39]. In particular, the power-law exponent of the PageRank distribution is the same as that of the degree distribution of the network. Therefore, by sampling query nodes according to the degree distribution, we are more likely to obtain the query nodes with relatively large PageRank scores.

Parameters. We compare our *SetPush* against five competitors: MC [17], LocalPush [28], FastPPR [29], RBS [36] and SubgraphPush [10]. Among them, MC is a Monte-Carlo method. LocalPush

[1] http://snap.stanford.edu/data
[2] http://law.di.unimi.it/datasets.php

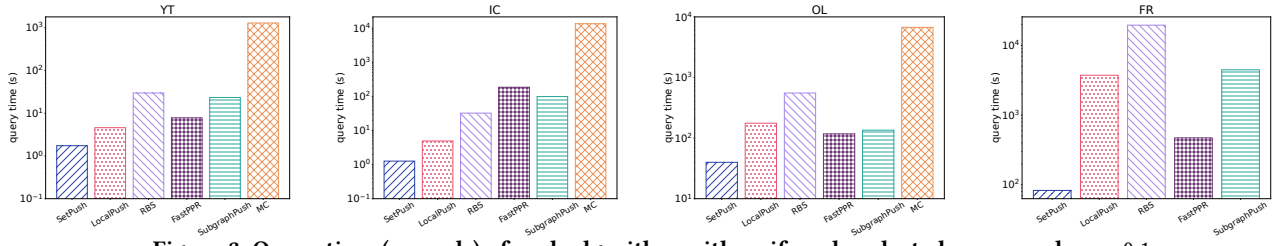

Figure 3: Query time (seconds) of each algorithm with uniformly selected query node, $c = 0.1$

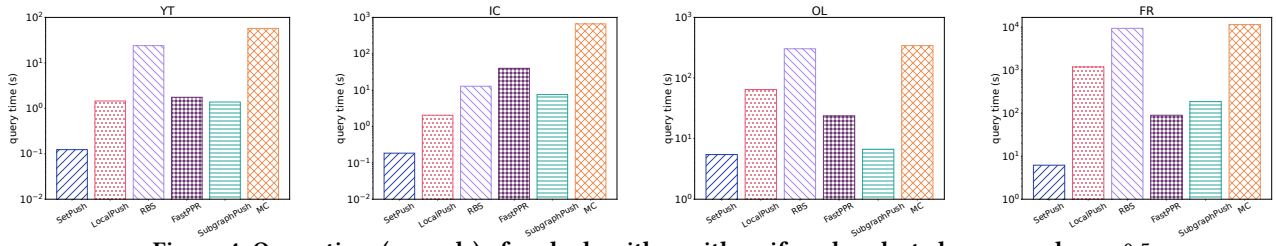

Figure 4: Query time (seconds) of each algorithm with uniformly selected query node, $c = 0.5$

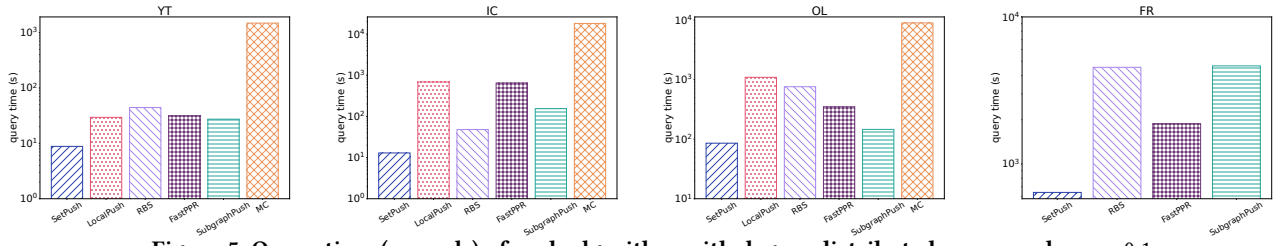

Figure 5: Query time (seconds) of each algorithm with degree distributed query nodes, $c = 0.1$

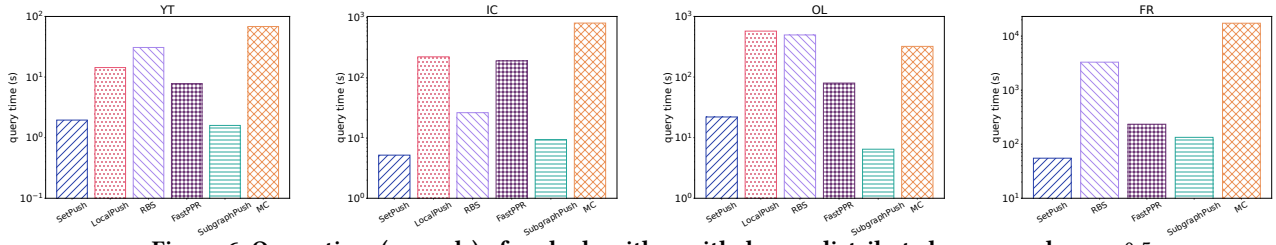

Figure 6: Query time (seconds) of each algorithm with degree distributed query nodes, $c = 0.5$

is a reverse exploration method. FastPPR [29], RBS [36] and SubgraphPush [10] are all hybrid methods. We set the parameters of these competitors strictly according to the theoretical analysis. Specifically, for the MC method [17], it has one parameter n_r, the number of α-random walks. We set $n_r = \frac{\frac{2}{3}c+2}{c^2 \cdot \boldsymbol{\pi}(t)} \cdot \ln \frac{1}{p_f}$ according to the analysis. The LocalPush method [28] has one parameter: the push threshold ε. We set $\varepsilon = \frac{c\alpha}{n}$. The FastPPR method has two parameters: the push threshold $r_{\max}$ and the number of random walks n_r. We set $r_{\max} = c \cdot \sqrt{\frac{\alpha d_t \cdot \log(1/(1-\alpha))}{n \cdot \log(1/p_f) \cdot \log(n/\alpha)}}$, and $n_r = \frac{45 \log_{1-\alpha}(c\alpha/2n)}{c} \cdot \sqrt{\frac{n \cdot d_t \cdot \log(1/(1-\alpha)) \log(2/p_f)}{\alpha \cdot \log(n/\alpha)}}$ according to the descriptions in FastPPR [29]. For RBS, recall that RBS can achieve the $\tilde{O}(n)$ time complexity by setting the threshold $\theta = \frac{c^2 \cdot \boldsymbol{\pi}(t)}{12 \cdot \log_{1-\alpha}(c\alpha/2n)}$. However, we do not know the real value of $\boldsymbol{\pi}(t)$ in advance. Thus, the value of $\boldsymbol{\pi}(t)$ can be only in place of the lower bound $\frac{\alpha}{n}$ of $\boldsymbol{\pi}(t)$ as indicated in Equation (2). Thus, in the experiments of RBS, we set $\theta = \frac{c^2\alpha}{12n \cdot \log_{1-\alpha}(c\alpha/2n)}$. For the SubgraphPush method, it has three parameters: the number of random walks n_r, the number of subgraphs k, and the maximum iteration number L. We set $n_r = \min\left\{n^{\frac{2}{3}} \cdot d_{\max}^{1/3} \cdot \left(\ln \frac{n}{p_f}\right)^{\frac{1}{3}} \cdot \left(\ln \frac{1}{p_f}\right)^{\frac{1}{3}} \cdot c^{-\frac{4}{3}}, n^{\frac{4}{5}} d^{\frac{1}{5}} \cdot \left(\ln \frac{n}{p_f}\right)^{\frac{1}{5}} \cdot \left(\ln \frac{1}{p_f}\right)^{\frac{2}{5}} \cdot c^{-\frac{6}{5}}\right\}$, $k = \frac{n \cdot \log 1/p_f}{c^2 \cdot n_r}$, and $L = \ln(c/n)$ following [10]. In all experiments, we set the failure probability $p_f = 0.1$, the relative error parameter $c = 0.1$, and the damping factor $\alpha = 0.2$ unless otherwise specified.

Average Overall Query Time. We first compare the empirical query time of all methods. Specifically, for each method, we issue one single-node PageRank query for each query node in the Q_1 query set, and report the average query time of each method over all the query nodes in Q_1 in Figure 3 and Figure 4. In particular, we set the relative error parameter $c = 0.1$ and $c = 0.5$ in Figure 3 and Figure 4, respectively. From Figure 3 and Figure 4, we observe that our *SetPush* consistently outperforms other competitors, which demonstrates the superiority of our *SetPush*. It's worth mentioning that we omit the MC method on the FR dataset in Figure 3 since the query time of MC on the FR dataset exceeds one day.

Moreover, in Figure 5 and Figure 6, we report the average query time of each method over all the query nodes in the query set

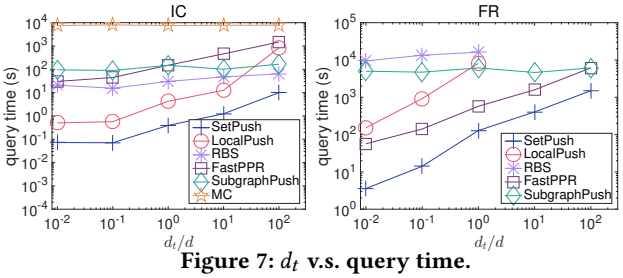

Figure 7: d_t v.s. query time.

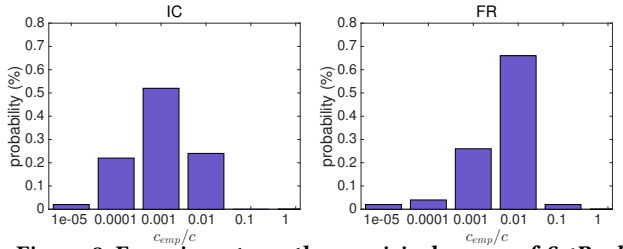

Figure 8: Experiments on the empirical errors of *SetPush*.

Q_2. We omit the LocalPush method in both Figure 5 and Figure 6, and the MC method in Figure 5 because the query time of these methods exceed one day. We note that our *SetPush* still consistently outperforms other competitors when $c = 0.1$. When $c = 0.5$, the empirical query time of our *SetPush* outperforms other competitors (except the SubgraphPush method) by up to an order of magnitude on all datasets. However, on the YT and OL datasets, the SubgraphPush method slightly outperforms our *SetPush*. We attribute the superiority of SubgraphPush as shown in Figure 6 to the blacklist trick adopted in the SubgraphPush method. In Figure 7, we report the increment of the query time of each method with increasing d_t and fixed $c = 0.1$. We observe that our *SetPush* can consistently outperform SubgraphPush on all datasets. This demonstrates the superiority and robustness of our *SetPush*.

d_t v.s. Average Overall Query Time. In Figure 7, we show the trade-off lines between d_t (i.e., the degree of the target node t) and the empirical query time. We leverage such experiments to observe the relationship between the query time of each method and the value of d_t. Specifically, we partition the vertex set V into five subsets V_1, V_2, V_3, V_4, V_5, such that the average node degrees d_1, d_2, d_3, d_4, d_5 of V_1, V_2, V_3, V_4, V_5 satisfy $d_1 \geq 100d$, $d_2 \in [10d, 100d)$, $d_3 \in [d, 10d)$, $d_4 \in [0.1d, d)$, and $d_5 \in [0.01d, 0.1d)$, respectively, where d denotes the average node degree in the graph G. In each subset (i.e., $V_1, \ldots, V_5$), we select five query nodes uniformly at random, and report the average query time of each method over the five query nodes. We omit LocalPush and RBS on the FR dataset when $d_t/d \geq 10$ because their query time exceeds one day. We set $c = 0.1$ and $p_f = 0.1$ in these experiments. From Figure 7, we note that our *SetPush* consistently outperforms all baseline methods on all datasets for all query sets. In particular, for law-degree query nodes, our *SetPush* achieves $10\times \sim 1000\times$ improvements on the query time over existing methods. For high-degree query nodes, the superiority of *SetPush* is gradually weakened, but still exists. Additionally, we observe:

- The query time of the Monte-Carlo method, RBS, and the SubgraphPush method nearly remain unchanged with the increment of d_t. This concurs with our analysis that the three methods do not include d_t in their complexity results.
- The query time of FastPPR and BiPPR increase slowly with the increment of d_t, while the query time of our *SetPush* and LocalPush grows linearly to d_t. This concurs with our analysis that the time complexities of FastPPR and BiPPR are both $\tilde{O}\left(\sqrt{n \cdot d_t}\right)$, while the time complexities of *SetPush* and LocalPush both have a linear dependence on d_t.

Empirical Errors of *SetPush*. In Figure 8, we evaluate the empirical error of our *SetPush*. Specifically, we adopt the power method [33] with the maximum iteration times $L = 100$ to compute the ground truth of PageRank. Furthermore, on each dataset, we fix the relative error parameter $c = 0.1$ and run our *SetPush* for each query node in the set Q_1. Then we compute the empirical relative error c_{emp} for each query node following $c_{emp} = \frac{|\hat{\boldsymbol{\pi}}(t) - \boldsymbol{\pi}(t)|}{\boldsymbol{\pi}(t)}$. We report the average of the values $\left(\frac{c_{emp}}{c}\right)$ over all query nodes in Figure 8. Note that $\left(\frac{c_{emp}}{c}\right) \leq 1$ implies that the empirical relative error of *SetPush* meets the requirement of the (c, p_f)-approximation of $\boldsymbol{\pi}(t)$. From Figure 8, we observe that the empirical relative errors of *SetPush* on all datasets are consistently smaller than c. In particular, on the IC datasets, the empirical relative errors of *SetPush* are smaller than c by up to two orders of magnitude. This demonstrates the correctness and query efficiency of our *SetPush*.

7 CONCLUSION

In this paper, we study the problem of single-node PageRank computation on undirected graphs. We propose a novel method, *SetPush*, which achieves the $\tilde{O}\left(\min\left\{d_t, \sqrt{m}\right\}\right)$ expected time complexity for estimating the target node t's PageRank with constant relative error and constant success probability. We prove that this is the best result among existing methods on undirected graphs. We also empirically demonstrate the effectiveness of *SetPush* on large-scale real-world datasets. For the future work, we note that the lower bound for the problem of single-node PageRank computation on undirected graphs is still unclear. Since we have already achieved the complexity bound $\tilde{O}\left(\min\left\{d_t, \sqrt{m}\right\}\right)$, a natural question is whether this complexity matches the lower bound for the problem.

ACKNOWLEDGMENTS

This research was supported in part by National Natural Science Foundation of China (No. U2241212, No. 61972401, No. 61932001, No. 61832017), by the major key project of PCL (PCL2021A12), by Beijing Natural Science Foundation (No. 4222028), by Beijing Outstanding Young Scientist Program No.BJJWZYJH012019100020098, by Alibaba Group through Alibaba Innovative Research Program, and by Huawei-Renmin University joint program on Information Retrieval. Hanzhi Wang was also supported by the Outstanding Innovative Talents Cultivation Funded Programs 2020 of Renmin University of China. We also wish to acknowledge the support provided by Engineering Research Center of Next-Generation Intelligent Search and Recommendation, Ministry of Education, and the fund for building world-class universities (disciplines) of Renmin University of China. Additionally, we acknowledge the support from Intelligent Social Governance Interdisciplinary Platform, Major Innovation & Planning Interdisciplinary Platform for the "Double-First Class" Initiative, Public Policy and Decision-making Research Lab, Public Computing Cloud, Renmin University of China.

REFERENCES

[1] [n.d.]. http://arxiv.org/abs/2307.13162.

[2] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S Mirrokni, and Shang-Hua Teng. 2007. Local computation of PageRank contributions. In *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 150–165.

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

[4] 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.

[5] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. 2010. Fast incremental and personalized pagerank. *arXiv preprint arXiv:1006.2880* (2010).

[6] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. 2004. Objectrank: Authority-based keyword search in databases. In *VLDB*, Vol. 4. 564–575.

[7] Ziv Bar-Yossef and Li-Tal Mashiach. 2008. Local approximation of pagerank and reverse pagerank. In *Proceedings of the 17th ACM conference on Information and knowledge management*. 279–288.

[8] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling Graph Neural Networks with Approximate PageRank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA.

[9] Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, Aristides Gionis, and Sebastiano Vigna. 2008. The query-flow graph: model and applications. In *Proceedings of the 17th ACM conference on Information and knowledge management*. 609–618.

[10] Marco Bressan, Enoch Peserico, and Luca Pretto. 2018. Sublinear algorithms for local graph centrality estimation. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 709–718.

[11] Karl Bringmann and Konstantinos Panagiotou. 2012. Efficient sampling methods for discrete distributions. In *International colloquium on automata, languages, and programming*. Springer, 133–144.

[12] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*. Springer, 693–703.

[13] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*. PMLR, 1725–1735.

[14] Fan Chung. 2007. The heat kernel as the pagerank of a graph. *Proceedings of the National Academy of Sciences* 104, 50 (2007), 19735–19740.

[15] Luc Devroye. 2006. Nonuniform random variate generation. *Handbooks in operations research and management science* 13 (2006), 83–121.

[16] Dániel Fogaras. 2003. Where to start browsing the web?. In *International Workshop on Innovative Internet Community Systems*. Springer, 65–79.

[17] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. 2005. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics* 2, 3 (2005), 333–358.

[18] David F Gleich. 2015. PageRank beyond the Web. *siam REVIEW* 57, 3 (2015), 321–363.

[19] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. 2013. Wtf: The who to follow service at twitter. In *Proceedings of the 22nd international conference on World Wide Web*. 505–514.

[20] Taher Haveliwala and Sepandar Kamvar. 2003. *The second eigenvalue of the Google matrix*. Technical Report. Stanford.

[21] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*. 271–279.

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

[23] Kyle Kloster and David F Gleich. 2014. Heat kernel based community detection. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1386–1395.

[24] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a social network or a news media?. In *Proceedings of the 19th international conference on World wide web*. 591–600.

[25] Peter Lofgren. 2015. *EFFICIENT ALGORITHMS FOR PERSONALIZED PAGERANK*. Ph.D. Dissertation. STANFORD UNIVERSITY.

[26] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2015. Bidirectional pagerank estimation: From average-case to worst-case. In *Algorithms and Models for the Web Graph: 12th International Workshop, WAW 2015, Eindhoven, The Netherlands, December 10-11, 2015, Proceedings 12*. Springer, 164–176.

[27] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2016. Personalized pagerank estimation and search: A bidirectional approach. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. 163–172.

[28] Peter Lofgren and Ashish Goel. 2013. Personalized pagerank to a target node. *arXiv preprint arXiv:1304.4658* (2013).

[29] Peter A Lofgren, Siddhartha Banerjee, Ashish Goel, and C Seshadhri. 2014. Fast-ppr: Scaling personalized pagerank estimation for large graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1436–1445.

[30] Michael Mitzenmacher and Eli Upfal. 2017. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press.

[31] Barbara Logan Mooney, L René Corrales, and Aurora E Clark. 2012. MoleculaR-networks: An integrated graph theoretic and data mining tool to explore solvent organization in molecular simulation. *Journal of computational chemistry* 33, 8 (2012), 853–860.

[32] Julie L Morrison, Rainer Breitling, Desmond J Higham, and David R Gilbert. 2005. GeneRank: using search engine technology for the analysis of microarray experiments. *BMC bioinformatics* 6, 1 (2005), 1–14.

[33] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: bringing order to the web. (1999).

[34] J Michael Steele. 2004. *The Cauchy-Schwarz master class: an introduction to the art of mathematical inequalities*. Cambridge University Press.

[35] Shang-Hua Teng et al. 2016. Scalable algorithms for data and network analysis. *Foundations and Trends® in Theoretical Computer Science* 12, 1–2 (2016), 1–274.

[36] Hanzhi Wang, Zhewei Wei, Junhao Gan, Sibo Wang, and Zengfeng Huang. 2020. Personalized pagerank to a target node, revisited. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 657–667.

[37] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: simple and effective approximate single-source personalized pagerank. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 505–514.

[38] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Yu Liu, Xiaoyong Du, and Ji-Rong Wen. 2019. Prsim: Sublinear time simrank computation on large power-law graphs. In *Proceedings of the 2019 International Conference on Management of Data*. 1042–1059.

[39] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. 2018. Topppr: top-k personalized pagerank queries with precision guarantees on large graphs. In *Proceedings of the 2018 International Conference on Management of Data*. 441–456.

[40] Neil A Weiss. 2005. A course in probability. 2005. , 385–386 pages.

[41] Wenpu Xing and Ali Ghorbani. 2004. Weighted pagerank algorithm. In *Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004.* IEEE, 305–314.

[42] Renchi Yang, Xiaokui Xiao, Zhewei Wei, Sourav S Bhowmick, Jun Zhao, and Rong-Hua Li. 2019. Efficient estimation of heat kernel pagerank for local clustering. In *Proceedings of the 2019 International Conference on Management of Data*. 1339–1356.

[43] Xi-Nian Zuo, Ross Ehmke, Maarten Mennes, Davide Imperati, F Xavier Castellanos, Olaf Sporns, and Michael P Milham. 2012. Network centrality in the human functional connectome. *Cerebral cortex* 22, 8 (2012), 1862–1875.