BIRD: Efficient Approximation of Bidirectional Hidden Personalized PageRank

Haoyu Liu
haoyu.liu@ntu.edu.sg
Nanyang Technological University
Singapore

Siqiang Luo[*]
siqiang.luo@@ntu.edu.sg
Nanyang Technological University
Singapore

ABSTRACT

In bipartite graph analysis, similarity measures play a pivotal role in various applications. Among existing metrics, the Bidirectional Hidden Personalized PageRank (BHPP) stands out for its superior query quality. However, the computational expense of BHPP remains a bottleneck. Existing approximation methods either demand significant matrix storage or incur prohibitive time costs. For example, current state-of-the-art methods require over 3 hours to process a single-source BHPP query on the real-world bipartite graph *Orkut*, which contains approximately 3×10^8 edges.

We introduce BIRD, a novel algorithm designed for answering single-source BHPP queries on weighted bipartite graphs. Through meticulous theoretical analysis, we demonstrate that BIRD significantly improves time complexity to $\widetilde{O}(n)$, as compared to the previous best one, $\widetilde{O}(m)$, under typical relative error setting and constant failure probability. (n, m denote the number of nodes and edges respectively.) Extensive experiments confirm that BIRD outperforms existing baselines by orders of magnitude in large-scale bipartite graphs. Notably, our proposed method accomplishes a single-source BHPP query on *Orkut* using merely 7 minutes.

PVLDB Reference Format:
Haoyu Liu and Siqiang Luo. BIRD: Efficient Approximation of Bidirectional Hidden Personalized PageRank. PVLDB, 17(9): 2255 - 2268, 2024.
doi:10.14778/3665844.3665855
PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/hyLiu-777/BIRD.

1 INTRODUCTION

In the vast domain of graph theory, bipartite graph is a ubiquitous structure that garners significant interest from both academic and industrial communities. A bipartite graph G consists of two distinct sets of nodes, U and V, and edges only exist between nodes from different sets. The fundamental task — *similarity measurement* in bipartite graphs, aims to identify nodes within U that are similar to a given node u based on a specific metric, and finds various real-world applications including E-commerce advertising [8, 11, 19], recommendation systems [26, 31, 32], and biomedical analysis [17, 51].

Despite its wide applicability, similarity measurement on bipartite graphs remains relatively under-explored compared to general graphs. Traditional metrics such as the Jaccard's coefficient [28] and Pearson's correlation coefficient [53] are not directly applicable or produce sub-optimal results on the intricate structure of bipartite graphs [54]. Meanwhile, measurements designed for general graphs, including Personalized PageRank (PPR) [24], SimRank [30], and others [18, 19, 61], either fail to capture the unique properties with inferior performance in mining tasks, or entail significant computational overheads. This underscores the pressing need for an effective bipartite similarity measurement that can provide precise measures for analysis and decisions in real-world applications.

In a recent study [64], the *Bidirectional Hidden Personalized PageRank* (BHPP), a strengthened variant of HPP [20], is shown to be effective in bipartite similarity measurement. Specifically, BHPP is defined based on the Personalized PageRank (PPR) [30, 38] on the graph $\widehat{G}$ constructed from the bipartite graph G with only nodes in U. Given two nodes $u_i, u_j \in U$, the BHPP value $\mathcal{B}(u_i, u_j)$ on G equals the summation of forward and reverse PPR value, i.e. $\boldsymbol{\pi}(u_i, u_j) + \boldsymbol{\pi}(u_j, u_i)$ over $\widehat{G}$. Inheriting the nature of similarity measurement, BHPP can be applied widely in real-world bipartite applications. For example, in query rewriting on sponsored search of BING [19], we can use BHPP to find similar queries according to a specific user's query submitted to the search engine. Also, in recommendation systems such as Amazon [6], similar items can be searched by the single-source BHPP similarities and thus generate item-based recommendations for users. Besides, in drug-target bipartite graph [17], new drug-target interactions can be predicted by computing the BHPP similarity for a given set of interested drugs. Upon our evaluation in Sec 7, BHPP achieves superior results against seven competitive measurements in above two scenarios. This demonstrates its superiority in bipartite domain.

While BHPP excels in effectiveness, its computational aspects can present challenges when scaling to larger graphs. Prior methods for HPP computation require up to $O(n_u^2)$ (n_u denotes the number of nodes in U) space cost for the materialization of $\widehat{G}$, which is prohibitive for large graphs. As such, advanced techniques for PPR computation on general graphs cannot be applied to solve the problem efficiently, posing a great technical challenge. Additionally, the localpush [9] can be modified to answer the reverse HPP query, leading to a technique called selectpush. Further combining techniques such as the power method [49] or the Monte Carlo [21] for forward HPP calculation, the BHPP query can be approximated in an overall time of $O(\frac{m}{\epsilon})$, where m denotes the number of edges and ϵ is the result estimation error threshold. However, such time complexity is unsatisfying especially when calculating relatively high-precision results with $\epsilon = O(\frac{1}{|n_u|})$. To overcome the linear reliance on the

[*]Siqiang Luo is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 9 ISSN 2150-8097.
doi:10.14778/3665844.3665855

error requirement, Yang [64] proposed an approximation method, namely ABHPP to speed up the computation process by utilizing a sequential push strategy after a sufficient period of selectpush, which further reduces the time cost to $O(m \log \frac{1}{\epsilon})$. Nonetheless, current methods are hard to break the $\widetilde{O}(m)$ complexity bound, which is rather inefficient for massive graphs with billion scale edges. For example, to answer a single-source BHPP query on *Orkut* with roughly 3×10^8 edges, it runs approximately 3 hours to meet the 10^{-6} accuracy requirement.

In this paper, we consider the problem of approximating the single-source BHPP query computation on weighted undirected bipartite graphs[1]. We propose a novel algorithm called BIRD, which breaks the $\widetilde{O}(m)$ barrier to achieve a superior time complexity bound in $\widetilde{O}(n)$ for deriving an approximation of single-source BHPP query within constant relative error and constant failure probability. Note that n, m denote the number of nodes, and edges respectively and $\widetilde{O}$ is a variant of the Big-Oh notation which ignores the polylogarithmic factors. Compared with the state-of-the-art (SOTA) method ABHPP which uses 3 hours to finish computation over the Orkut dataset, our BIRD algorithm finishes in merely 7 minutes. We summarize our contributions in the following:

- **Theoretical Improvements**. Through meticulous theoretical analysis, our proposed algorithm BIRD significantly improves query time complexity over the state-of-the-art BHPP algorithms. Under a typical relative error setting, BIRD breaks the $\widetilde{O}(m)$ complexity barrier of the existing algorithms and achieves a lower $\widetilde{O}(n)$ complexity for estimating the single-source BHPP query on weighted, undirected bipartite graphs.
- **Innovative Algorithmic Design**: Our BIRD algorithm successfully combines a carefully designed weight-dependent probability mass propagation technique with a novel iterative reuse strategy for these propagation results, to achieve the upmost computation efficiency while securing the theoretically provable result quality.
- **Empirical Validation**: We implement BIRD over eight real-world datasets. The empirical results indicate that our algorithm achieves orders of magnitude speed-up over all baselines on massive bipartite graphs, i.e. up to 50× acceleration on the large-scale dataset *Orkut* with roughly 3×10^8 edges.

2 PRELIMINARIES

2.1 Notations

We mostly follow the notations in [18, 20, 64]. Consider a bipartite graph $G = (U \cup V, E)$, where nodes consist of two disjoint node sets $U = \{u_1, u_2, \cdots, u_{n_u}\}$ and $V = \{v_1, v_2, \cdots, v_{n_v}\}$ with number n_u and n_v respectively. Each edge $e = (u, v, w(u, v)) \in E$ represents an undirected connection between the node $u \in U$ and $v \in V$ with positive weight $w(u, v)$. N_u denotes the connected neighbors of node u and $d_u = |N_u|$ denotes node u's degree. $N_u^{(2)}$ represents node u's 2-hop neighbors. m equals the number of edges. w_u is defined as the sum of weights in edges incident to node u:

$$w_u = \sum_{v \in N_u} w(u, v). \tag{1}$$

We use $\mathbf{U} \in \mathbb{R}^{n_u \times n_v}$ and $\mathbf{V} \in \mathbb{R}^{n_v \times n_u}$ to represent the forward and backward transition matrices. For each node $u \in U$ and $v \in V$:

$$\mathbf{U}(u, v) = \frac{w(u,v)}{w_u}, \text{and } \mathbf{V}(v, u) = \frac{w(v,u)}{w_v}. \tag{2}$$

The hidden transition matrix [18] for node set $\mathbf{U}$ is defined as $\mathbf{P} = \mathbf{U} \cdot \mathbf{V} \in \mathbb{R}^{n_u \times n_u}$, where each (u_i, u_j) entry is calculated upon:

$$\mathbf{P}(u_i, u_j) = \sum_{v \in N_{u_i} \cap N_{u_j}} \mathbf{U}(u_i, v) \cdot \mathbf{V}(v, u_j). \tag{3}$$

Note that the number of non-zero entries of $\mathbf{P}$ can be up to $O(n_u^2)$, which is prohibitive for computing over large-scale graphs.

2.2 Hidden Personalized PageRank

In contrast to the classic Personalized PageRanks which is defined on general graphs, *Hidden Personalized PageRank (HPP)* [18, 20] is a measure for bipartite graphs. Consider a bipartite graph G and two nodes $u, u_i \in U$, the *Hidden Personalized PageRank (HPP)* $\boldsymbol{\pi}(u, u_i)$ of node u_i to u is defined as the probability that an α-decay random walk starts from node u stops at node u_i. Given the decay probability $\alpha < 1$, at each step, such a random walk initializing from u either terminates at current node, i.e. u_j, with α probability, or steps to its any neighbor $u_k \in N_{u_j}$ based on the transition probability $\mathbf{P}(u_j, u_k)$. The HPP value $\boldsymbol{\pi}(u, u_i)$ can be calculated from [12] as:

$$\boldsymbol{\pi}(u, u_i) = \sum_{\ell=0}^{\infty} \alpha \cdot (1 - \alpha)^{\ell} \cdot \mathbf{P}^{\ell}(u, u_i). \tag{4}$$

ℓ-hop HPP. With an integer $\ell \geq 0$, the ℓ-hop HPP $\boldsymbol{\pi}^{(\ell)}(u, u_i)$ corresponds to the probability that an α-decay random walk started from u terminates at node u_i using exactly ℓ steps:

$$\boldsymbol{\pi}^{(\ell)}(u, u_i) = \alpha(1 - \alpha)^{\ell} \cdot \mathbf{P}^{\ell}(u, u_i). \tag{5}$$

holds for each integer $\ell \geq 1$ and each node $u \in U$. Combined with Equation (4), we can easily derive that $\boldsymbol{\pi}(u, u_i) = \sum_{\ell=0}^{\infty} \boldsymbol{\pi}^{(\ell)}(u, u_i)$.

2.3 Problem Definition

To remedy the bias of HPP, Yang [64] models the similarity between nodes u and u_i by *Bidirectional Hidden Personalized PageRank* (BHPP), defined as:

$$\mathcal{B}_{\boldsymbol{u}}(u_i) = \boldsymbol{\pi}(u, u_i) + \boldsymbol{\pi}(u_i, u), \tag{6}$$

where the measurement considers a symmetrical relation between node u and u_i. In this paper, we focus on designing efficient approximation solutions for single-source BHPP queries, defined as:

(c, p_f)-approximation of single-source BHPP query. Given a bipartite graph $G = (U \cup V, E)$, a relative error c, and a source node $u \in U$, a (c, p_f)-approximation of BHPP returns estimated BHPP value $\widehat{\mathcal{B}}_{\boldsymbol{u}}(u_i)$ for each node $\forall u_i \in U$, such that for any $\mathcal{B}_{\boldsymbol{u}}(u_i) \geq \delta$,

$$|\mathcal{B}_{\boldsymbol{u}}(u_i) - \widehat{\mathcal{B}}_{\boldsymbol{u}}(u_i)| \leq c \cdot \mathcal{B}_{\boldsymbol{u}}(u_i) \tag{7}$$

holds with probability at least $1 - p_f$.

We denote the single-source *Forward HPP* (FHPP) vector and *Reverse HPP* (RHPP) vector as $\{\boldsymbol{\pi}(u, u_i) | u_i \in U\}$ and $\{\boldsymbol{\pi}(u_i, u) | u_i \in U\}$ respectively given the source node u in following discussions.

3 EXISTING TECHNIQUES

In this section, we first analyze representative techniques tailored to single-direction HPP computation. Then, we explain the trade-offs of sota approximation algorithm ABHPP in answering the single-source BHPP query. Table 1 shows the time complexity comparison.

[1] Following conventional settings in previous works [18–20, 64].

Table 1: Complexity comparison from baselines to our BIRD method answering the single-source BHPP query on the undirected bipartite graph with constant relative error and failure probability. δ decides the BHPP target scope (i.e., larger than δ) and the typical setting determines $\delta = 1/n$, where n denotes the number of target nodes (n_u or n_v), following settings in [40, 43, 58, 59].

Methods	Time Complexity On Reverse, Forward and Bidirectional HPP				Improvements Over Baselines	
	Full-Scope			Typical	Full-Scope	Typical
	Reverse	Forward	Bidirectional	Bidirectional	Bidirectional	Bidirectional
MCSP [9, 21]	$\widetilde{O}(\frac{m}{\delta})$	$\widetilde{O}(\frac{1}{\delta})$	$\widetilde{O}(\frac{m}{\delta} + \frac{1}{\delta})$	$\widetilde{O}(nm + n)$	$\widetilde{O}(\max\{\frac{1}{\delta}, \frac{n \cdot m}{n_u + n_v}\})$	$\widetilde{O}(m)$
BPI [49]	$\widetilde{O}(m)$	$\widetilde{O}(m)$	$\widetilde{O}(m)$	$\widetilde{O}(m)$	$\widetilde{O}(\max\{1, \frac{n \cdot \delta \cdot m}{n_u + n_v}\})$	$\widetilde{O}(m/n)$
PISP [9, 49]	$\widetilde{O}(\frac{m}{\delta})$	$\widetilde{O}(m)$	$\widetilde{O}(\frac{m}{\delta} + m)$	$\widetilde{O}(nm + m)$	$\widetilde{O}(\max\{\frac{1}{\delta}, \frac{n \cdot m}{n_u + n_v}\})$	$\widetilde{O}(m)$
ABHPP [64]	$\widetilde{O}(m)$	$\widetilde{O}(m)$	$\widetilde{O}(m)$	$\widetilde{O}(m)$	$\widetilde{O}(\max\{1, \frac{n \cdot \delta \cdot m}{n_u + n_v}\})$	$\widetilde{O}(m/n)$
BIRD (ours)	$\widetilde{O}(min\{\frac{n_u + n_v}{n \cdot \delta}, m\})$			$\widetilde{O}(n)$	—	—

3.1 Classical Methods

The Monte-Carlo Approach. The FHPP vector $\boldsymbol{\pi}(u, u_i)$, from source node u to any node $u_i \in U$, equals the probability that an α-decay random walk started from node u terminates at node u_i. Thus, the ***Monte-Carlo*** (MC) approach [21] simulates n_r number of random walks from node u, and then returns the fraction of walks ending at u_i as the estimator. According to [21], it requires $n_r = O\left(\frac{2(1+c/3)\cdot \ln(1/p_f)}{c^2 \cdot \delta}\right)$ to derive a (c, p_f)-approximation of FHPP. As such, MC is bounded by $O(n_r/\alpha) = \widetilde{O}(\frac{1}{\delta})$. While, as shown in [40, 59], MC is inefficient due to large amount of random walks.

The PowerIteration Method. The *PowerIteration* (PI) method [49] is an iterative method that approximates the FHPP vector $\boldsymbol{\pi}(u, \cdot) = \{\boldsymbol{\pi}(u, u_i) | u_i \in U\}$ with the following linear formula:

$$[\boldsymbol{\pi}(u, \cdot) - \alpha \cdot \mathbf{e_u}] / (1 - \alpha) = \boldsymbol{\pi}(u, \cdot) \cdot \mathbf{P} = (\boldsymbol{\pi}(u, \cdot) \cdot \mathbf{U}) \cdot \mathbf{V}, \tag{8}$$

where the one-hot vector $\boldsymbol{e_u} \in \mathcal{R}^{1 \times n_u}$ only has value 1 at the u-th position and 0 otherwise. According to [25], PI realizes ϵ absolute error bound for each $\boldsymbol{\pi}(u, u_i)$ after $L = \log_{\frac{1}{1-\alpha}} \frac{1}{\epsilon}$ iterations While during each iteration, it requires two matrix-vector multiplications which cost at least $O(m)$ time. Therefore, it requires in total $O(m \cdot \log \frac{1}{\epsilon})$ cost, which is rather ineffective on massive bipartite graphs.

The Select Push. The *SelectPush* (SP) method [9] approximates the RHPP query. Generally it performs as a deterministic version of MC in a reverse manner which recursively pushes non-zero *residues* along edges through a graph traversal of G starting from node u, where the *residues* can be considered as the portion of random walks that are still alive. Specifically, Algorithm 1 first initializes the residue vector $\boldsymbol{r_u}(\cdot) \in \mathcal{R}^{|U \cup V|}$ that equals to $\mathbf{e_u}$ as well as a reserve vector $\{\boldsymbol{\pi}(u, u_i) = 0 | u_i \in U\}$ for estimating RHPP. Next, it iteratively pushes the residues of selected nodes, obeying the ϵ, to their neighbors. During each iteration, for any node $u_i \in U$ with large residue, its neighbor $v_j \in N_{u_i}$ will receive a residue increment of amount $(1 - \alpha) \cdot w(v, u) \cdot \boldsymbol{r_u}(u_i)/w_v$ and the reserve vector raises its estimation by $\alpha \cdot \boldsymbol{r_u}(u_i)$. After all neighbors are processed, the residue $\boldsymbol{r_u}(u_i)$ dismisses. Subsequently, all non-zero residues of nodes in V will be pushed back to U losslessly. The iteration terminates until all residues in U are less than ϵ. SP can approximate RHPP query in $O(\frac{m}{\epsilon})$ time, which grows linearly to graph edges, hindering it from running efficiently on large graphs.

Algorithm 1: SP

```
   Input: Bipartite graph $G$, Target node $u$, Error tolerance $\epsilon$ and
          Decay factor $\alpha$.
   Output: $\{\boldsymbol{\pi}(u_i, u) | u_i \in U\}, r(\cdot)$.
1  $\boldsymbol{\pi}(u_i, u) \leftarrow 0 \ \forall u_i \in U$;
2  $r(u) \leftarrow 1; r(x) \leftarrow 0 \ \forall x \in U \cup V$ and $x \neq u$;
3  // SelectPush
4  while $\exists u_i \in U \ s.t. r(u_i) > \epsilon$ do
5      for $u_i \in U \ s.t. \ r(u_i) > \epsilon$ do
6          for $v \in N_{u_i}$ do
7              $r(v) \leftarrow r(v) + (1 - \alpha) \cdot \frac{w(v,u)}{w_v} \cdot r(u_i)$;
8          $\boldsymbol{\pi}(u_i, u) \leftarrow \boldsymbol{\pi}(u_i, u) + \alpha \cdot r(u_i)$;
9          $r(u_i) \leftarrow 0$;
10     for $v \in V s.t. \ r(v) > 0$ do
11         for $u_j \in N_v$ do
12             $r(u_j) \leftarrow r(u_j) + \frac{w(u_j,v)}{w_{u_j}} \cdot r(v)$;
13         $r(v) \leftarrow 0$;
14 return $\{\boldsymbol{\pi}(u_i, u) | u_i \in U\}$;
```

3.2 Sequential Select Push

The ***S****equential* ***S****elect* ***P****ush* (SSP) method [64] overcomes the efficiency issues of SP. To explain, consider a node $v \in V$ connecting to 1000 neighbors $u_1 - u_{1000}$ in U. According to lines 4-12 in Algorithm 1, node v first (i) receives residues from the selected neighbors, and then (ii) conducts 1000 edge-push operations (we refer to *edge-push operation* as a push computation alongside an edge and *push operation* as the total push computation on a selected node.) to $u_1 - u_{1000}$ along each edge. After some iterations, only a few neighbors of v would be selected as residues of the majority are (slightly) less than

ϵ. As such to further deplete a certain amount of node v's residue, SP requires numerous iterations and each will involve at least 1000 edge-push operations. To alleviate this, SSP uses a cost recorder n_p to track the number of edge-push operations. It performs SP first when n_p is small. Upon n_p exceeding a predefined threshold, SSP will instantly switch to the *sequential push*, i.e. push each node in U with non-zero residues, which is shown to be efficient to aggregate residues from node v's neighbors in one batch before pushing back. Thus, SSP removes the linear dependency on ϵ in SP and runs in $O(m\cdot\log(\frac{1}{\epsilon})) = \widetilde{O}(m)$ time.

3.3 Baselines Answering BHPP Queries

Classical methods can be combined directly to answer the (c, p_f)-approximation of the single-source BHPP query.

PISP. A straight way to answer a (c, p_f)-approximation BHPP query is by summing up the results of a $(c/2, p_f)$-approximation FHPP query and a $(c/2, p_f)$-approximation RHPP query for each node. This motivates us to combine PI and SP, dubbed as PISP. To satisfy the approximation requirement of the query, PI needs to conduct $O\left(\log_{\frac{1}{1-\alpha}}(\frac{1}{\epsilon})\right) = O\left(\log_{\frac{1}{1-\alpha}}(\frac{2}{c\cdot\delta})\right)$ iterations and the error threshold of SP needs to be set as $\epsilon = \frac{c\cdot\delta}{2}$. This results in a total time complexity of $O\left(m\cdot\log_{\frac{1}{1-\alpha}}(\frac{2}{c\cdot\delta}) + m\cdot\frac{2}{c\cdot\delta}\right) = \widetilde{O}(\frac{m}{\delta} + m)$.

MCSP. Similarly, we can answer the $(c/2, p_f)$-approximation FHPP query by simulating $n_r = O\left(\frac{8(1+c/6)\cdot ln(1/p_f)}{c^2\cdot\delta}\right)$ random walks and calculate the $(c/2, p_f)$-approximation RHPP query results by SP with $\epsilon = \frac{c\cdot\delta}{2}$, which derives a total time cost in $\widetilde{O}(\frac{m}{\delta} + \frac{1}{\delta})$.

BPI. Different from PISP and MCSP which answer the BHPP query separately, we propose *Bidirectional PowerIteration* (BPI) method to re-use the (c, p_f)-approximation FHPP query of PI to compute the BHPP results. Specifically, the HPP value exhibits an underlying *reversibility property* that for any node-pair $(u, u_i) \in U \times U$ [64], it holds $w_u \cdot \boldsymbol{\pi}(u, u_i) = w_{u_i} \cdot \boldsymbol{\pi}(u_i, u)$. As such, each RHPP value $\boldsymbol{\pi}(u_i, u)$ on source node u for node $u_i \in U$ can be calculated as $\boldsymbol{\pi}(u_i, u) = w_u/w_{u_i} \cdot \boldsymbol{\pi}(u, u_i)$. Thus, the error is amplified by at most $n_a = min_{u_i \in U} w_u/w_{u_i}$ times. By adjusting the error threshold to $\epsilon = c \cdot \delta/n_a$ in answering the FHPP query, BPI can answer single-source BHPP query in $O\left(m\cdot\log_{\frac{1}{1-\alpha}}(n_a/c\cdot\delta)\right) = \widetilde{O}(m)$ time.

ABHPP [64]. Combining SSP and PI, the ABHPP method can answer the single-source BHPP query in $\widetilde{O}(m)$ time. It utilizes SSP to compute the RHPP vector and then calculates the FHPP vector by re-using the former to reduce the iteration number in PI. In specific, ABHPP has one parameter ϵ and returns BHPP vector with at most ϵ additive error. By setting $\epsilon = c\cdot\delta$, it answers the (c, p_f)-approximation single-source BHPP query in $O(m\cdot\log\frac{1}{c\cdot\delta}) = \widetilde{O}(m)$.

4 THE BIRD ALGORITHM

In this section, we propose ***Bi**directional **D**iscrete Push* (BIRD), an efficient approximation algorithm for single-source BHPP query.

4.1 High-level Ideas

As discussed in Sec 3.2, the ABHPP algorithm spiritually strives to make every push operation "effective" according to the residue of the selected node. This is realized by combining the selective and sequential push strategies, where the former selects nodes in U with large residues to distribute non-trivial probability mass, and the latter tends to aggregate as much residues as possible to nodes in V by sequentially pushing nodes in U with non-zero residues. However, such a push strategy overlooks an important problem that may still largely impact the efficiency: *does the neighbors receive meaningful residue increments in each push operation?*

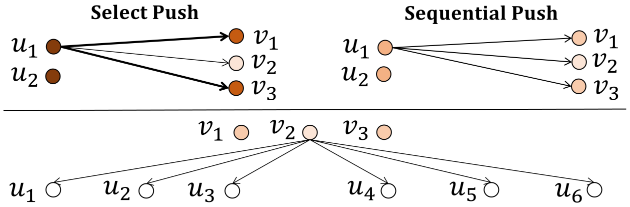

Figure 1: Efficiency bad cases in ABHPP method. Tiny residue increments transferred to node v_2 need to be transferred back to every neighbor of v_2, resulting in unnecessary overheads.

Recall that in Algorithm 1 (line 6), when pushing a node $u \in U$, each neighbor $v \in N_u$ will receive an increment of $(1-\alpha)\cdot\frac{w(v,u)}{w_v}\cdot r(u)$. We observe that such increments can be potentially redundant as demonstrated in Figure 1. When the edge proportion $\frac{w(v,u)}{w_v}$ is small in the select push or the residue value $r(u)$ is tiny in the sequential strategy, the total probability increment transported to v becomes trivial, i.e. contributing negligible difference to the overall quality but still occupies push computation cost. On the contrary, our BIRD method ensures that the push operation is "effective" according to the increment of neighbors that each push will transport a significant amount of residues. Specifically, we propose a novel strategy called *push discretization unit*, tailored to weighted graphs, by either adding the increment until reaching a predefined error threshold, i.e. θ, or ignoring it without performing a edge-push operation. In this way, each edge-push operation will at least transport θ probability mass, making the overall push process more effective. Moreover, the extra error brought by ignoring pushes over small increments will be bounded based on our algorithm design.

Besides, the ABHPP algorithm along with several classical methods answers the BHPP query in a separate manner, which first answers the RHPP query and then conducts power iterations to compute FHPP results, lacking an efficient optimization of reusing the shared intermediate results to reduce superfluous computation costs. We alleviate this problem by first deriving a concept called *truncated BHPP*, which is built upon the summation of each ℓ-hop BHPP value. Then in each iteration, we calculate the ℓ-hop FHPP value by fully reusing the estimation of the ℓ-hop RHPP vector with carefully analyzed query quality guarantee. Such bidirectional design further allows us to effectively reuse the intermediate estimations in each iteration to reduce the computation redundancy.

Putting the above ideas together, our BIRD algorithms obtain a superior theoretical time complexity of $\widetilde{O}(n)$ in the typical setting, and achieve orders of magnitude speed-up upon evaluation on real-world datasets compared with the state-of-the-art approaches. In following content, for ease of discussions, we will introduce the details of BIRD with $\delta = \frac{1}{n_u}$ by default unless otherwise specified. We will relax this assumption in section 5.4 for smaller δ.

4.2 Truncated BHPP

In this section, we derive the concept of *Truncated BHPP*, which performs an essential role in designing our algorithm. Note that for any node pair $(u, u_i) \in U^2$, the HPP value $\boldsymbol{\pi}(u, u_i)$ can be decomposed into the summation of its ℓ-hop units:

$$\boldsymbol{\pi}(u, u_i) = \textstyle\sum_{\ell=0}^{\infty} \boldsymbol{\pi}^{(\ell)}(u, u_i), \tag{9}$$

where $\boldsymbol{\pi}^{(\ell)}(u, u_i)$ is the probability that an α-decay random walk starts from u terminates at u_i using exactly ℓ steps. For different ℓ, such events are mutually exclusive, and thus we can compute the original HPP value as above. We then define the *ℓ-hop BHPP* of node pair (u, u_i) as $\mathcal{B}_{\boldsymbol{u}}^{(\boldsymbol{\ell})}(u_i) = \boldsymbol{\pi}^{(\ell)}(u, u_i) + \boldsymbol{\pi}^{(\ell)}(u_i, u)$. Accordingly, we can define the BHPP query based on a summation format that

$$\mathcal{B}_{\boldsymbol{u}}(u_i) = \textstyle\sum_{\ell=0}^{\infty} \mathcal{B}_{\boldsymbol{u}}^{(\boldsymbol{\ell})}(u_i) = \sum_{\ell=0}^{\infty} \left[\boldsymbol{\pi}^{(\ell)}(u, u_i) + \boldsymbol{\pi}^{(\ell)}(u_i, u)\right]. \tag{10}$$

Given a node pair (u, u_i) in an undirected bipartite graph G, a restart probability α, and a constant relative error c, we derive the concept *Truncated BHPP* corresponding to $\overline{\mathcal{B}}_{\boldsymbol{u}}(u_i)$ that

$$\overline{\mathcal{B}}_{\boldsymbol{u}}(u_i) = \textstyle\sum_{\ell=0}^{L} \mathcal{B}_{\boldsymbol{u}}^{(\boldsymbol{\ell})}(u_i), \tag{11}$$

where $L = \log_{1-\alpha} \frac{c}{4n_u} = O(\log n_u) = \widetilde{O}(1)$. Next, we present Lemma 1 to show that deriving a (c, p_f)-approximation of the single-source BHPP vector $\mathcal{B}_{\boldsymbol{u}}(\cdot)$ can be achieved by deriving a $(c/2, p_f)$-approximation of the truncated BHPP vector $\overline{\mathcal{B}}_{\boldsymbol{u}}(\cdot)$.

LEMMA 1. *Given source node u in the graph G, $\widehat{\mathcal{B}}(u, \cdot)$ is a (c, p_f)-approximation of node u's single-source BHPP vector $\mathcal{B}_{\boldsymbol{u}}(\cdot)$ if*

$$|\widehat{\mathcal{B}}_{\boldsymbol{u}}(u_i) - \overline{\mathcal{B}}_{\boldsymbol{u}}(u_i)| \le \frac{c}{2} \cdot \overline{\mathcal{B}}_{\boldsymbol{u}}(u_i)$$

holds with at least $1 - p_f$ probability for $\forall u_i \in U$ s.t. $\overline{\mathcal{B}}_{\boldsymbol{u}}(u_i) > \frac{1}{2n_u}$.

PROOF. According to Equation (10),

$$\begin{aligned}\mathcal{B}_{\boldsymbol{u}}(u_i) &= \overline{\mathcal{B}}_{\boldsymbol{u}}(u_i) + \textstyle\sum_{\ell=L+1}^{\infty} \left[\boldsymbol{\pi}^{(\ell)}(u, u_i) + \boldsymbol{\pi}^{(\ell)}(u_i, u)\right] \\ &= \overline{\mathcal{B}}_{\boldsymbol{u}}(u_i) + \textstyle\sum_{\ell=L+1}^{\infty} \alpha(1-\alpha)^{\ell} \cdot \left[\mathbf{P}^{\ell}(u_i, u_j) + \mathbf{P}^{\ell}(u_j, u_i)\right] \\ &\le \overline{\mathcal{B}}_{\boldsymbol{u}}(u_i) + 2 \cdot \textstyle\sum_{\ell=L+1}^{\infty} \alpha(1-\alpha)^{\ell} = \overline{\mathcal{B}}_{\boldsymbol{u}}(u_i) + 2(1-\alpha)^{L}.\end{aligned}$$

We conclude $0 \le \mathcal{B}_{\boldsymbol{u}}(u_i) - \overline{\mathcal{B}}_{\boldsymbol{u}}(u_i) \le \frac{c}{2n_u}$ holds. Consequently,

$$\begin{aligned}|\mathcal{B}_{\boldsymbol{u}}(u_i) - \widehat{\mathcal{B}}_{\boldsymbol{u}}(u_i)| &\le |\mathcal{B}_{\boldsymbol{u}}(u_i) - \overline{\mathcal{B}}_{\boldsymbol{u}}(u_i)| + |\overline{\mathcal{B}}_{\boldsymbol{u}}(u_i) - \widehat{\mathcal{B}}_{\boldsymbol{u}}(u_i)| \\ &\le \frac{c}{2n_u} + \frac{c}{2} \cdot \overline{\mathcal{B}}_{\boldsymbol{u}}(u_i) \le \frac{c}{2n_u} + \frac{c}{2} \cdot \mathcal{B}_{\boldsymbol{u}}(u_i) \le c\mathcal{B}_{\boldsymbol{u}}(u_i)\end{aligned}$$

holds for any $\forall \mathcal{B}_{\boldsymbol{u}}(u_i) \ge \frac{1}{n_u}$, which completes the proof. □

4.3 Push Discretization Unit

In previous sections, we have shown some intuitions on the importance of each push operation in transporting sufficient amount of residue. Here, we present the novel technique named *Push Discretization Unit* (PDU) to achieve our goal. In baseline methods, when conducting edge-push operations on a selected node u_i, each neighbor $v_j \in N_{u_i}$ will be transferred a residue increment $X(u_i, v_j)$ alongside the edge (u_i, v_j) proportional to $w(v_j, u_i) \cdot r_u(u_i)/w_v$ (lines 4-8 in Algorithm 1). This sort of deterministic push undiminishedly requires $O(d(u_i))$ computation cost for each selected node, regardless of the transported increment $X(u_i, v_j)$'s magnitude. Even when $X(u_i, v_j)$ is small enough that the probability mass contributed by transferring to v_j is negligible, v_j is still pushed equally as other neighbors. Subsequently, another push operation will happen on node v_j. Each node in V with non-zero residue will be selected to recycle the probability portion, which further includes $O(d(v_j))$ computation overhead (lines 9-12). This hinders them from breaking the the $\widetilde{O}(m)$ bound.

Algorithm 2: PDU

```
  Input: Residue vector $\boldsymbol{r}(\cdot)$, Reserve vector $\boldsymbol{\pi}(\cdot)$, Candidate node $c$,
         Error parameter $\theta$ and Decay factor $\alpha$.
  Output: Updated reserve vector $\boldsymbol{\pi}(\cdot)$.
1 $X(c,\cdot) \leftarrow \{\frac{w(c_i,c)\cdot(1-\alpha)\cdot r(c)}{w_{c_i}} | c_i \in N_c\}$  // Reverse
2           or $\{\frac{w(c_i,c)\cdot(1-\alpha)\cdot r(c)}{w_c} | c_i \in N_c\}$;  // Forward
3 for each $c_i \in N_c$ s.t. $X(c, c_i) \ge \theta$ do
4     $\boldsymbol{\pi}(c_i) \leftarrow \boldsymbol{\pi}(c_i) + X(c, c_i)$;
5 $p \leftarrow rand(0, 1)$;
6 for each $c_i \in N_c$ s.t. $1 > \frac{X(c,c_i)}{\theta} \ge p$ do
7         $\boldsymbol{\pi}(c_i) \leftarrow \boldsymbol{\pi}(c_i) + \theta$   // Reverse
8     or $\boldsymbol{\pi}(c\ ) \leftarrow \boldsymbol{\pi}(c_i) + \theta$;  // Forward
9 return $\boldsymbol{\pi}(\cdot)$
```

On the contrary, the PDU technique decides the edge-push operation from a candidate node c (c can be any node in node set U or V) to its neighbors $\forall c_i \in N_c$ to happen or not based on the magnitude of the increment to each neighbor, rather than the size of the residue $r(c)$ itself. Specifically in Algorithm 2, given a candidate node c, the residue vector $r(\cdot)$, a threshold parameter θ and the decay factor α, we compare the value of the increment with θ. For example, in the reverse manner, when the increment equals $w(c_i, c) \cdot (1-\alpha) \cdot r(c)/w_{c_i} > \theta$, we consider it an effective push transportation already and we add it into the reserve variable $\boldsymbol{\pi}(c_i)$ (lines 4-5), similar as that in SSP. However, if $w(c_i, c) \cdot (1-\alpha) \cdot r(c)/w_{c_i} \le \theta$, we no longer conduct the same deterministic strategy. Instead, we construct a new increment variable $X_d(c, c_i)$ obeying a scaled binomial distribution based on the increment value shown below:

$$\mathbb{X}(c, c_i) = \begin{cases} \theta\,, & \text{with } p = \frac{w(c_i,c)\cdot(1-\alpha)\cdot r(c)}{w_{c_i}\cdot\theta}; \\ 0\,, & \text{with } p = 1 - \frac{w(c_i,c)\cdot(1-\alpha)\cdot r(c)}{w_{c_i}\cdot\theta}. \end{cases} \tag{12}$$

Next, we discretize the original increment into 0 (not to push) or θ (push the threshold value) according to $\mathbb{X}(c, c_i)$. To achieve the above procedure, we generate a random variable $\boldsymbol{p}$ following the standard uniform distribution and conduct edge-push operation when $\frac{w(c_i,c)\cdot(1-\alpha)\cdot r(c)}{w_{c_i}\cdot\theta} \ge p$ (lines 5-7). Such discretization procedure will ensure that $X_d(c, c_i)$ is a conditional unbiased estimator of $X(c, c_i)$, which will assist our correctness analysis in Sec 5.1 later. Lemma 2 formally presents the unbiaseness property below.

LEMMA 2. *The discrete increment $X_d(c, c_i)$ is conditional unbiased of $X(c, c_i)$ as $X(c, c_i) = \mathbb{E}\left[X_d(c, c_i) | r(\cdot)\right]$ in Algorithm 2.*

PROOF. $\mathbb{E}\left[X_d(c, c_i) | r(\cdot)\right] = \theta \cdot \frac{w(c_i,c)\cdot(1-\alpha)\cdot r(c)}{w_{c_i}\cdot\theta} = X(c, c_i)$. □

By conducting PDU, those tiny increments $\frac{w(c_i,c)\cdot(1-\alpha)\cdot r(c)}{w_{c_i}}$ resulting from either the small weight of the edge (c_i, c) or the ignorable residues from the candidate node c will be either ignored (thus reduce the push cost), or be assigned a larger value θ. In this

Algorithm 3: BIRD

```
   Input: Bipartite graph $G$, Source node $u$, Decay factor $\alpha$, Error
          parameter $\theta$, Parameter $\gamma$.
   Output: $\{\widehat{\mathcal{B}}_{\boldsymbol{u}}(u_i)|\ u_i \in U\}$.
1  Initialize $\widehat{\mathcal{B}}_{\boldsymbol{u}}(\cdot) \leftarrow 2\alpha e_u$; $\boldsymbol{r}_{\boldsymbol{r}}^{(0)}(\cdot), \boldsymbol{r}_{\boldsymbol{f}}^{(0)}(\cdot) \leftarrow e_u$; $L \leftarrow \log_{1-\alpha} \frac{c}{4 \cdot n_u}$;
2  $U_\gamma \leftarrow \{s \mid s \in U\ s.t.\ \frac{w_s}{w_u} \leq \gamma\}, V_I \leftarrow \bigcup_{s \in U \setminus U_\gamma} N_s$;
3  for $\ell = 0$ to $L - 1$ do
4      Initialize $\boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(\cdot) \in \mathcal{R}^{n_u}$ and $\boldsymbol{r}_{\boldsymbol{v}}(\boldsymbol{\cdot}) \in \mathcal{R}^{n_v} \leftarrow 0$;            ↑
5      for each $u_i \in U\ s.t.\ r_r^{(\ell)}(u_i) > 0$ do                                        ¦
6          $\boldsymbol{r}_{\boldsymbol{v}}(\cdot) \leftarrow$ PDU$\left(\ \boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(\cdot),\ \boldsymbol{r}_{\boldsymbol{v}}(\cdot),\ u_i,\ \theta\ , \alpha\ \right)$;          ¦ Phase-r
7      for each $v_j \in V\ s.t.\ \boldsymbol{r}_{\boldsymbol{v}}(v_j) > 0$ do                                       ¦
8          $\boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(\cdot) \leftarrow$ PDU$\left(\ \boldsymbol{r}_{\boldsymbol{v}}(\cdot),\ \boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(\cdot),\ v_j,\ \theta\ , 0\right)$;      ↓
9      Initialize $\boldsymbol{r}_{\boldsymbol{f}}^{(\ell+1)}(\cdot) \in \mathcal{R}^{n_u}$ and $\boldsymbol{r}_{\boldsymbol{v}} \in \mathcal{R}^{n_v} \leftarrow 0$;              ↑
10     for each $u_i \in U_\gamma\ s.t.\ \boldsymbol{r}_{\boldsymbol{f}}^{(\ell)}(u_i) > 0$ do                                     ¦
11         $\boldsymbol{r}_{\boldsymbol{f}}^{(\ell+1)}(u_i) \leftarrow \frac{w_{u_i}}{w_u} \cdot \boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(u_i)$;  // Re-use                   ¦
12     for each $v_j \in V_I$ do                                                    ¦ Phase-f
13         $\boldsymbol{r}_{\boldsymbol{v}}(v_j) \leftarrow$ PDU$\left(\ \boldsymbol{r}_{\boldsymbol{f}}^{\ell}(\cdot),\ \boldsymbol{r}_{\boldsymbol{v}}(\cdot),\ v_j,\ \theta\ , \alpha\ \right)$;          ¦
14     for each $u_j \in U \setminus U_\gamma$ do                                               ¦
15         $\boldsymbol{r}_{\boldsymbol{f}}^{(\ell+1)}(u_i) \leftarrow$ PDU$\left(\ \boldsymbol{r}_{\boldsymbol{v}}(\cdot),\ \boldsymbol{r}_{\boldsymbol{f}}^{(\ell+1)}(\cdot),\ u_j,\ \theta\ , 0\right)$;    ↓
16     $\widehat{\mathcal{B}}(u, \cdot) \leftarrow \widehat{\mathcal{B}}(u, \cdot) + \alpha \cdot \left[\boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(\cdot) + \boldsymbol{r}_{\boldsymbol{f}}^{(\ell+1)}(\cdot)\right]$;
17 return $\{\widehat{\mathcal{B}}(u, u_i)|\ u_i \in U\}$
```

way, each edge-push operation empirically happened will distribute at least θ amount of probability mass, which overcomes the efficiency limitations in SSP where trivial increments happen. Note that $\theta \in (0, 1)$ is a tuneable threshold and the choice of θ is carefully analyzed in Sec 5 to ensure the push quality. We then build up our BIRD combining PDU technique and the *truncated BHPP* concept.

Note that the idea of incorporating randomness in each deterministic push for general unweighted graphs is first introduced in [56], i.e. RBS. Our PDU technique modifies and generalizes the approach described in [56] by integrating both weight dependency and pushing directions. In particular, RBS is known for approximating single-target PPR on general unweighted graphs, while our PDU technique supports both single-source and single-target HPP queries on weighted bipartite graphs. This adaptation better tailors our push strategies for answering bidirectional queries specifically in weighted scenarios. We emphasize that the addition of *weight* considerations, especially when coupled with our estimator *re-using design* in the main BIRD algorithm, brings in additional difficulty in ensuring the query quality, i.e. bounding the variance of estimators. Nonetheless, we employ carefully-designed procedures and provide rigorous theoretical analysis following to address these challenges.

4.4 The BIRD Procedure

Given a source node u, BIRD computes a (c, p_f)-approximation of $\mathcal{B}(u, \cdot)$ by deriving a $(c/2, p_f)$-approximation $\widehat{\mathcal{B}}_{\boldsymbol{u}}(\cdot)$ of $\overline{\mathcal{B}}_{\boldsymbol{u}}(\cdot)$ based on the following:

$$\widehat{\mathcal{B}}_{\boldsymbol{u}}(\cdot) = \textstyle\sum_{\ell=0}^{L} \widehat{\mathcal{B}}_{\boldsymbol{u}}^{(\ell)}(\cdot) = \sum_{\ell=0}^{L} \left[\widehat{\boldsymbol{\pi}}^{(\ell)}(u, \cdot) + \widehat{\boldsymbol{\pi}}^{(\ell)}(\cdot, u)\right], \qquad (13)$$

where $\widehat{\mathcal{B}}_{\boldsymbol{u}}^{(\ell)}(\cdot)$ acts as an unbiased estimator of each ℓ-hop BHPP vector and meets the requirement of $(c/2, p_f)$-approximation. Thus, the estimator $\widehat{\mathcal{B}}_{\boldsymbol{u}}(\cdot)$ follows as a (c, p_f)-approximation of $\mathcal{B}(u, \cdot)$. Algorithm 3 illustrates the pseudo-code of the BIRD algorithm. In BIRD, we maintain two variables, the *reverse ℓ-hop residue* $r_{\boldsymbol{r}}^{(\ell)}(\cdot)$ and the *forward ℓ-hop residue* $r_{\boldsymbol{f}}^{(\ell)}(\cdot)$ for each node in set U to update our estimator. At the end of (ℓ-1)-th iteration, the on-hand residue vectors $r_{\boldsymbol{r}}^{(\ell)}(\cdot)$ and $r_{\boldsymbol{f}}^{(\ell)}(\cdot)$ will be added to our global reserve vector, acting as a role of unbiased estimation of the ℓ-hop BHPP vector (line 19). Later, the new residue vector $r_{\boldsymbol{r}}^{(\ell+1)}(\cdot)$ and $r_{\boldsymbol{f}}^{(\ell+1)}(\cdot)$ will be updated by conducting the bidirectional push procedures consisting of two phases: the reverse phase phase-r and the forward phase phase-f, respectively. By repeating the above updating steps, the global reserve vector $\widehat{\mathcal{B}}_{\boldsymbol{u}}^{(\ell)}(\cdot)$ is accumulated with each ℓ-hop BHPP estimation until the iteration number exceeding the predefined amount L by truncating the BHPP vector. Initially, $r_{\boldsymbol{r}}^{(\ell)}(\cdot), r_{\boldsymbol{f}}^{(\ell)}(\cdot)$ are set to $\mathbf{0}$ for $\forall \ell \in \{1, 2, ..., L\}$ and $r_{\boldsymbol{r}}^{(0)}(\cdot), r_{\boldsymbol{f}}^{(0)}(\cdot)$ are set to $\mathbf{e_u}$. Then in each main iteration, we repeatedly conduct phase-r and phase-f to update $r_{\boldsymbol{r}}^{(\ell+1)}(\cdot), r_{\boldsymbol{f}}^{(\ell+1)}(\cdot)$ based on $r_{\boldsymbol{r}}^{(\ell)}(\cdot), r_{\boldsymbol{f}}^{(\ell)}(\cdot)$ by enumerating ℓ.

Reverse Phase (Phase-r). The Phase-r (lines 5-9) aims at computing the reverse part of $\widehat{\mathcal{B}}_{\boldsymbol{u}}^{(\ell)}(\cdot)$ as $\widehat{\boldsymbol{\pi}}^{(\ell)}(\cdot, u)$ in Equation 13. Intuitively, it simulates the graph traversal process in a reverse manner first from node set U to V and then back to U from V. The residue on a node can be considered as the fraction of random walks which are still alive. Under our PDU technique, such proportion will be transported to neighbors only when the probability mass alongside the edge is non-trivial. On the contrary, the deterministic push process in ABHPP ignores this factor. Specifically, phase-r performs the following steps to achieve an unbiased estimation of the ℓ-hop vector $\boldsymbol{\pi}^{(\ell)}(\cdot, u)$:

1. Initialize a residue vector $\boldsymbol{r}_{\boldsymbol{v}}(\cdot) \leftarrow \mathbf{0}$, for each node in set V;
2. Pick up all candidate nodes $u_i \in U$ with non-zero $r_{\boldsymbol{r}}^{(\ell)}(u_i)$;
3. Update residue vector $\boldsymbol{r}_{\boldsymbol{v}}(\cdot)$ by running PDU in reverse direction with inputs: residue vector $\boldsymbol{r}(\boldsymbol{\cdot}) = \boldsymbol{r}_{\boldsymbol{v}}(\cdot)$, reserve vector $\boldsymbol{\pi}(\boldsymbol{\cdot}) = \boldsymbol{r}_{\boldsymbol{v}}(\cdot)$, candidate node $\boldsymbol{c} = u_i$, error parameter $\boldsymbol{\theta}$ and decay factor $\boldsymbol{\alpha} = \alpha$ denoted as PDU($\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(\cdot),\ \boldsymbol{r}_{\boldsymbol{v}}(\cdot),\ u_i,\ \theta\ , \alpha$).
4. Pick up all candidate nodes $v_j \in V$ with non-zero $\boldsymbol{r}_{\boldsymbol{v}}(v_j)$ and update $\boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(\cdot)$ by running PDU($\boldsymbol{r}_{\boldsymbol{v}}(\cdot),\ \boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(\cdot),\ v_j,\ \theta\ , 0$).

Note that the traversal direction differs in phase-r and phase-f in Algorithm 2, adapting to their own unique recursive property. In the reverse phase, the increment $X(c, c_i) = \frac{w(c_i, c) \cdot (1-\alpha) \cdot r(c)}{w_{c_i}}$ can be considered as $r(c)$ proportion of α-decay random walks travels from c_i to c (line 1) and vice versa for the forward phase (line 2).

Running Example 1. A phase-r running example is displayed in Figure 2. Assume at the beginning of the ℓ-th iteration, we have the reverse residue vector $r_{\boldsymbol{r}}^{(\ell)}(\cdot)$ with non-zero values on node set $\{u\}$. Then we first conduct PDU on node u in the reverse direction and increments are only effective towards neighbor v_1, where increments of v_2, v_3 are discretized into 0 (no push happens). Thus, one edge-push computation is empirically happened and only node v_1 has non-zero residue in for vector $r_v(\cdot)$ updated in V. In turn, another PDU runs on node v_1 and updates $r_{\boldsymbol{r}}^{(\ell+1)}(\cdot)$, generating non-zero values on nodes $\{u, u_2, u_5\}$. The whole procedure only takes

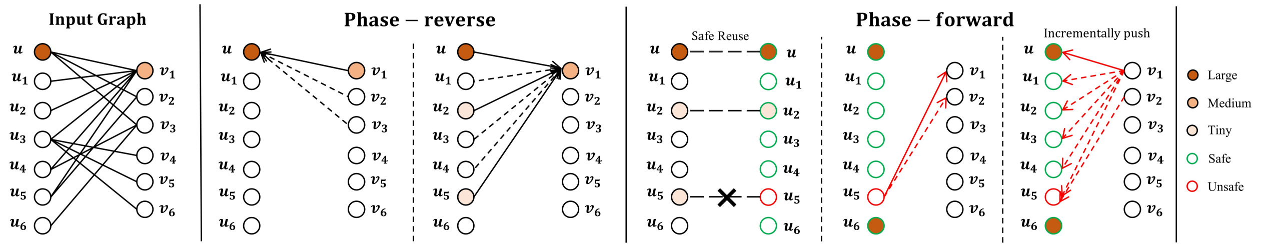

Figure 2: A running example of our BIRD Algorithm at the ℓ-th iteration.

$O(1) + O(3) = O(4)$ computations, compared to the SSP which reaches every neighbor, taking $O(3) + O(6 + 2 + 3) = O(14)$ steps.

Forward Phase (Phase-f). Upon termination of phase-r, we have accomplished the computation of $r_r^{(\ell)}(\cdot), r_f^{(\ell)}(\cdot)$ and $r_r^{(\ell+1)}(\cdot)$ residue vectors in the ℓ-th iteration. Our next goal is to estimate the forward part of $\widehat{\mathcal{B}}_u^{(\ell)}(\cdot)$ as $\widehat{\pi}^{(\ell)}(\cdot, u)$ by updating the $r_f^{(\ell+1)}(\cdot)$. Note that the PDU technique allows us to conduct either forward or reverse push directions, thus a straightforward way is to perform the same procedure as that in *Phase-f* but change the push direction of PDU to forward to compute it independently. However, this naive solution unavoidably, though provides a workout, at least doubles the overhead by adding a scale-similar computation overhead in phase-f. To further optimizing the redundancy issue, we propose to re-use the computation results on hand, i.e. the reverse BHPP vector, for estimating $\widehat{\pi}^{(\ell)}(\cdot, u)$. We then propose the *reversibility property* of the ℓ-hop BHPP vector in following Lemma 3, which motivates us to design an efficient re-using procedure. The detail is presented as follow:

Lemma 3. *(Proof in Sec 9) Given an undirected bipartite graph G, for any two nodes u_i, u_j in U and $\ell \geq 1$, the ℓ-hop HPP satisfies*

$$\pi^{(\ell)}(u_i, u_j) \cdot w_{u_i} = \pi^{(\ell)}(u_j, u_i) \cdot w_{u_j}.$$

Lemma 3 hints that we may reuse the residue vector $r_r^{(\ell+1)}(\cdot)$ obtained in phase-r to estimate $r_f^{(\ell+1)}(u_i)$ by setting $r_f^{(\ell+1)}(u_i) = \frac{w_{u_i}}{w_u} \cdot r_r^{(\ell+1)}(u_i)$ for each $u_i \in U$. Then, the forward residue vector will consequently be an unbiased estimation that

$$\mathbb{E}\left[r_f^{(\ell+1)}(u_i)\right] = \mathbb{E}\left[\frac{w_{u_i}}{w_u} \cdot r_r^{(\ell+1)}(u_i)\right] = \frac{w_{u_i}}{w_u} \cdot \pi^{(\ell)}(u_i, u) = \pi^{(\ell)}(u, u_i),$$

if the reverse vector possess the unbiased property (proved latter in Sec 5.1). However, although the direct re-using of the reverse vector for all nodes in U is convenient, it may result in inaccurate query results. Recall that we have evolved randomness in the PDU technique, thus the residue vector $r_r^{(\ell+1)}(\cdot)$ computed is a random variable. When using samples of random variables for estimations, we also expect its variance to be small or at least controllable. Based on the previous re-using setting, for $\forall u_i \in U$, we have

$$\mathbb{V}\mathrm{ar}\left[r_f^{(\ell+1)}(u_i)\right] = \mathbb{V}\mathrm{ar}\left[\frac{w_{u_i}}{w_u} \cdot r_r^{(\ell+1)}(u_i)\right] = (\frac{w_{u_i}}{w_u})^2 \cdot \mathbb{V}\mathrm{ar}\left[r_r^{(\ell+1)}(u_i)\right].$$

This scale of variance depends on two factors: the weight of the re-using node u_i and the variance of the estimator $r_r^{(\ell+1)}(u_i)$. Luckily, we can ensure the quality of $r_r^{(\ell+1)}(u_i)$ by bounding its variance as $\mathbb{V}\mathrm{ar}\left[r_r^{(\ell+1)}(u_i)\right] \leq O(\theta)$, which is formally analyzed in Sec 5.2 later. However, it's unrealistic to assume an universal distribution for node weights. Even if we can guarantee the variance of each $r_r^{(\ell+1)}(u_i)$, the variance of the most risky node, i.e. the node with the largest weight, can reach $\left(\max_{u_i}(\frac{w_{u_i}}{w_u})^2 \cdot \theta\right)$. Lacking any prior knowledge of G's weights, such estimation is inapplicable since the maximum weight can be extremely large and the source node weight can be small, leading to unstable results.

To entirely guarantee the quality in re-using the estimator, we derive a new node set $U_\gamma \subseteq U$ called the *safe set*, by introducing another parameter γ to bound the variance of re-using and play an overall efficiency-quality trade-off. Specifically, given the parameter $\gamma > 0$, we first construct the node set $U_\gamma \leftarrow \{s \mid s \in U \ s.t. \ \frac{w_s}{w_u} \leq \gamma\}$ (line 3). The set U_γ contains *safe nodes* in U where we can directly re-use the reverse residue for updating the forward residue vector (lines 11-12). For such *safe* nodes, their re-used estimators' variance can be bounded by $\gamma^2 \cdot \theta$, providing an intermediary to guarantee the final estimation quality. For example, setting $\gamma = 1$ ensures the variance quality of reusing is no larger than the on-hand calculations. The choose of γ is analyzed later in Sec 5.3.

For the remaining nodes in $U \setminus U_\gamma$, we incrementally conduct DPUs to simulate a forward graph traversal step. We construct node set $V_I \leftarrow \bigcup_{s \in U \setminus U_\gamma} N_s$, which delimits the scope of nodes in V that are possible to participate in the computation process when utilizing $r_f^{(\ell)}(\cdot)$ for the results of all the *unsafe* nodes in $U \setminus U_\gamma$. Note that the variance of *unsafe* nodes will exceed $\gamma^2 \cdot \theta$ by a direct reuse, so that we conduct extra computation steps to eliminate the issues. Next, by only evolving the nodes which are related to the unsafe nodes, we manipulate the phase-f by gathering residues for nodes in V_I (lines 13-14) and then fetching those residues towards unsafe nodes to update the corresponding residue (lines 15-16). In this way, we can finish the computation $r_f^{(\ell+1)}(\cdot)$ in a *safe* yet non-wasteful way. It is worth mentioning that phase-f requires much less computation overhead than phase-r by only updating a subset of U in the residue vector and re-using the reverse estimations to fulfill the others. Finally at the end of each iteration, we accumulate the α proportion of $r_r^{(\ell+1)}(\cdot)$ and $r_f^{(\ell+1)}(\cdot)$ to the our target estimator $\widehat{\mathcal{B}}_u(\cdot)$. After iterating all $\ell \leq L$, we return $\widehat{\mathcal{B}}_u(\cdot)$ as estimator to the $(c/2, p_f)$-approximation of $\overline{\mathcal{B}}_u(\cdot)$.

Running Example 2. We display a toy running example in Figure 2 of our BIRD algorithm with one iteration. After the phase-r of the ℓ-th iteration, we already have on-hand $\ell + 1$ residue vector $r_r^{(\ell+1)}(\cdot)$ with non-zero values on node set $\{u, u_2, u_5\}$. Assume $r_f^{(\ell)}(\cdot)$ has non-zero values on node set $\{u, u_6\}$ and the $r_f^{(\ell+1)}(\cdot)$ vector are all zeros. Then in phase-f, we build up the *safe set* $U_\gamma = \{u, u_1 - u_4, u_6\}$ and re-use the non-zero forward residue vector $r_r^{(\ell+1)}(\cdot)$ on $\{u, u_2\} \subseteq U_\gamma$

to directly compute $\boldsymbol{r}_{\boldsymbol{f}}^{(\ell+1)}(u)$ and $\boldsymbol{r}_{\boldsymbol{f}}^{(\ell+1)}(u_2)$. For the *unsafe* node u_5, we fetch for the possible nodes in U which need to conduct incremental push computations. As such, node u with non-zero residue $\boldsymbol{r}_{\boldsymbol{f}}^{(\ell)}(u)$ is selected to conduct the similar PDU process as in the phase-r. Note that node u_6, which is supposed to be pushed also with non-zero residue, is excluded since for now it does not evolve in the computation paths for updating u_5's residue.

5 THEORETICAL ANALYSIS

In this section, we present theoretical properties of the BIRD, regarding correctness in Sec 5.1, result accuracy in Sec 5.2, efficiency in Sec 5.3, and discussing parameter δ in Sec 5.4.

5.1 Correctness

Lemma 2 ensures that the expectation of each discretized increment $\mathbb{E}[X_d(c, c_i)]$ equals to the original increment $X(c, c_i)$. This hints that our final estimator based on them is unbiased as well.

Theorem 1. *The estimator $\widehat{\mathcal{B}}_{\boldsymbol{u}}(\cdot)$ returned by Algorithm 3 is unbiased that $\mathbb{E}[\widehat{\mathcal{B}}(u,\cdot)] = \overline{\mathcal{B}}_{\boldsymbol{u}}(\cdot)$.*

Proof. We break down the overall prove of Theorem 1 by deriving several following technical lemmas:

Lemma 4. *Given source node u and for each $\ell \geq 0$, the temporary residue $\boldsymbol{r}_{\boldsymbol{v}}(\cdot)$ in Alg. 3* phase-r *on node set V holds that for $\forall v_j \in V$,*

$$\mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{v}}(v_j) | \boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(\cdot)\right] = \textstyle\sum_{u_i \in N_{v_j}} (1-\alpha) \cdot \frac{w(v_j,u_i)}{w_{v_j}} \cdot r_r^{(\ell)}(u_i).$$

Lemma 4 is directly implied when combining Lemma 2 with the linearity of expectation based on $\boldsymbol{r}_{\boldsymbol{v}}(v_j) = \sum_{(u_i,v_j)\in E} X_d(u_i, v_j)$. This benefits from our unbiased design in DPU that when conducting pushing from set U to V, the temporary residue vector on set V remains unbiased on expectation. With similar property on pushing back from V to U, we further derive Lemma 5 to show that our estimators calculated each iteration in Algorithm 3 are unbiased.

Lemma 5. *(Proof in Sec 9) Given source node u and $\forall \ell \geq 0$, the reverse and forward ℓ-hop residues $\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(\cdot), \boldsymbol{r}_{\boldsymbol{f}}^{(\ell)}(\cdot)$ are unbiased estimators of $\boldsymbol{\pi}^{(\ell)}(u,\cdot)/\alpha, \boldsymbol{\pi}^{(\ell)}(\cdot,u)/\alpha$ respectively such that $\forall u_i \in U$,*

$$\mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(u_i)\right] = \frac{\boldsymbol{\pi}^{(\ell)}(u_i,u)}{\alpha}, \quad \mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{f}}^{(\ell)}(u_i)\right] = \frac{\boldsymbol{\pi}^{(\ell)}(u,u_i)}{\alpha}.$$

Equipped with Lemma 5 and the definition of $\overline{\mathcal{B}}_{\boldsymbol{u}}(\cdot)$, we can prove that $\widehat{\mathcal{B}}_{\boldsymbol{u}}(\cdot)$ returned by Algorithm 3 is an unbiased estimator of the truncated Bidirectional HPP $\overline{\mathcal{B}}_{\boldsymbol{u}}(\cdot)$ of node u. □

Now, we have ensured the correctness of Algorithm 3 on expectation. To further guarantee the estimation quality, we shall bound the variance of $\widehat{\mathcal{B}}_{\boldsymbol{u}}^{(\ell)}(\cdot)$ for each $\ell \geq 0$, which will assist in assuring the failure probability for the final approximation. We utilize the following Chebyshev Inequality to bound the failure probability.

Fact 1 (The Chebyshev's Inequality [45]). *Denote X a random variable. For any real number $\epsilon > 0$, $\mathbb{P}[|X - \mathbb{E}[X]| \geq \epsilon] \leq \frac{\mathbb{V}\mathrm{ar}[X]}{\epsilon^2}$.*

5.2 Approximation Quality Guarantee

We first showcase our main theorem as follows in this section.

Theorem 2. *Given source node u, we claim that the estimator $\widehat{\mathcal{B}}_{\boldsymbol{u}}(\cdot)$ returned by Algorithm 3 is a (c, p_f)-approximation of the single-source BHPP vector $\mathcal{B}_{\boldsymbol{u}}(\cdot)$, such that for each $\mathcal{B}_{\boldsymbol{u}}(u_i) \geq \frac{1}{n_u}$,*

$$\mathbb{P}\left[|\widehat{\mathcal{B}}_{\boldsymbol{u}}(u_i) - \mathcal{B}_{\boldsymbol{u}}(u_i)| \geq c \cdot \mathcal{B}_{\boldsymbol{u}}(u_i)\right] \leq p_f.$$

Proof. Theorem 2 provides us evidence that our BIRD Algorithm will return quality guaranteed estimations. Note that the estimation is conducted iteration by iteration, we consider the following lemma to first bound the variance of each ℓ-hop BHPP estimator $\widehat{\mathcal{B}}_{\boldsymbol{u}}^{(\ell)}(\cdot)$.

Lemma 6. *(Proof in Sec 9) Given source node u, for each $\forall u_i \in U$ and $\ell \geq 0$, the variance of $\widehat{\mathcal{B}}_{\boldsymbol{u}}^{(\ell)}(u_i)$ can be bounded as*

$$\mathbb{V}\mathrm{ar}\left[\widehat{\mathcal{B}}_{\boldsymbol{u}}^{(\ell)}(\cdot)\right] \leq \max((1+\gamma)^2, 2) \cdot 2\theta \cdot \mathcal{B}_{\boldsymbol{u}}^{(\ell)}(u_i).$$

Based on the variance bound and the Chebyshev's Ineqality, we are then able to bound the failure probability of $\widehat{\mathcal{B}}_{\boldsymbol{u}}^{(\ell)}(\cdot)$ that

$$\mathbb{P}\left[|\widehat{\mathcal{B}}^{(\ell)}(u,u_i) - \mathcal{B}_{\boldsymbol{u}}^{(\ell)}(u_i)| \geq \sqrt{6\theta \cdot \gamma_m \cdot \mathcal{B}_{\boldsymbol{u}}^{(\ell)}(u_i)}\right] \leq \frac{1}{3}, \tag{14}$$

where $\gamma_m = \max((1+\gamma)^2, 2)$. The above inequality implies a $c/2$-relative error for all $\mathcal{B}_{\boldsymbol{u}}^{(\ell)}(u_i) \geq \frac{24\cdot\gamma_m\cdot\theta}{c^2}$. Since $\theta \leq \frac{c^2 \mathcal{B}_{\boldsymbol{u}}^{(\ell)}(u_i)}{24\cdot\gamma_m}$ and consequently $24 \cdot \gamma_m \cdot \theta \cdot \mathcal{B}_{\boldsymbol{u}}^{(\ell)}(u_i) \leq (c/2 \cdot \mathcal{B}_{\boldsymbol{u}}^{(\ell)}(u_i))^2$. It then follows

$$\mathbb{P}\left[|\widehat{\mathcal{B}}^{(\ell)}(u,u_i) - \mathcal{B}_{\boldsymbol{u}}^{(\ell)}(u_i)| \geq c/2 \cdot \mathcal{B}_{\boldsymbol{u}}^{(\ell)}(u_i)\right] \leq \frac{1}{3}$$

for all $\mathcal{B}_{\boldsymbol{u}}^{(\ell)}(\cdot) \geq \frac{24\cdot\gamma_m\cdot\theta}{c^2}$ with constant probability. By setting $\theta = \frac{c^2}{48\cdot L\cdot\gamma_m\cdot n_u}$, we can obtain a $(c/2, 1/3)$-approximation for all $\mathcal{B}_{\boldsymbol{u}}^{(\ell)}(u_i) \geq \frac{1}{2Ln_u}$. According to Equation (11), we can therefore derive a $c/2$ relative error guarantee for our estimator $\widehat{\mathcal{B}}_{\boldsymbol{u}}(u_i)$ for all $\overline{\mathcal{B}}_{\boldsymbol{u}}(u_i) \geq \frac{1}{2n_u}$. Note that we can apply the Median-of-Mean trick [14] to reduce the failure probability to arbitrarily small by only adding a log factor to the running time. For example, by taking the median of $\log n$ independent copies of $\widehat{\mathcal{B}}_{\boldsymbol{u}}(u_i)$ as the final estimator, the failure probability is brought from $1/3$ to $1/n^2$. By further applying the union bound to n source nodes $u \in U$, and each $\ell \geq 0$, the failure probability will become $1/n$. Therefore, we can conclude that $\widehat{\mathcal{B}}(u,\cdot)$ is a $(c/2, p_f)$-approximation of $\overline{\mathcal{B}}(u,\cdot)$.

Supposition 1. *Given source node u and for $\forall u_i \in U$, $\widehat{\mathcal{B}}_{\boldsymbol{u}}(u_i)$ is a $(c/2, p_f)$-approximation for all $\overline{\mathcal{B}}_{\boldsymbol{u}}(u_i) \geq \frac{1}{2n_u}$ satisfying*

$$\mathbb{P}\left[|\widehat{\mathcal{B}}_{\boldsymbol{u}}(u_i) - \overline{\mathcal{B}}_{\boldsymbol{u}}(u_i)| \geq c/2 \cdot \overline{\mathcal{B}}_{\boldsymbol{u}}(u_i)\right] \leq p_f.$$

With Supposition 1 and Lemma 1, we conquer Theorem 2. □

By guaranteeing the quality of approximations, we obtain the value of our error parameter $\theta = \frac{c^2}{48\cdot L\cdot\gamma_m\cdot n_u}$. Intuitively, larger θ incorporates more PDUs, and thus less push operations by discrediting increments, leading to faster computations. Next, we show that the exact time cost can be bounded as the reciprocal of θ.

5.3 Efficiency Analysis

Now we present the expected time cost of our BIRD algorithm.

Theorem 3. *The expected time cost of Algorithm 3 can be computed as $(\frac{n_u+n_v}{\alpha n_u}) \cdot \frac{1}{\theta}$. By setting $\theta = \frac{c^2}{48\cdot L\cdot\gamma_m\cdot n_u}$ and choosing $\gamma_m = 2$, the expected time cost of Algorithm 3 is bounded by $\widetilde{O}(n_u + n_v)$.*

Proof. Here we denote $\mathbb{C}_r^{(\ell)}(u_i, v_j)$ and $\mathbb{C}_r^{(\ell)}(v_i, u_j)$ the cost for edge-push operation from U to V and V to U respectively in the phase-r on the ℓ-th iteration in Algorithm 3. We first derive the following two inequalities (Proof in Sec 9):

$$\begin{aligned} &\sum_{\ell=0}^{L-1} \sum_{(u,v)\in E} \mathbb{E}\left[\mathbb{C}_r^{(\ell)}(v,u)\right] \le \frac{1}{\alpha\theta} \cdot \sum_{u_j \in U} \boldsymbol{\pi}(u_j, u); \\ &\sum_{\ell=0}^{L-1} \sum_{(u,v)\in E} \mathbb{E}\left[\mathbb{C}_r^{(\ell)}(u,v)\right] \le \frac{(1-\alpha)}{\alpha\theta} \cdot \sum_{v_j \in V} \mathbb{P}\left[v_j \to u\right]. \end{aligned} \tag{15}$$

Next, by choosing node $u \in U$ uniformly, the total expected time cost in phase-r for source node u can be bounded as

$$\begin{aligned} \mathbb{E}\left[\mathbb{C}_r(u)\right] &\le \sum_{\ell=0}^{L-1} \sum_{(u,v)\in E} \mathbb{E}\left[\mathbb{C}_r^{(\ell)}(v,u)\right] + \mathbb{E}\left[\mathbb{C}_r^{(\ell)}(u,v)\right] \\ &\le \frac{1}{(\alpha\theta)\cdot n_u} \cdot \sum_{u\in U}\left[\sum_{v_j\in V} \mathbb{P}[v_j \to u] + \sum_{u_j \in U} \boldsymbol{\pi}(u_j,u)\right]. \\ &\le \frac{1}{(\alpha\theta)\cdot n_u} \cdot \left[\sum_{v_j\in V}\sum_{u\in U} P(v_j \to u) + \sum_{u_j\in U}\sum_{u\in U} \boldsymbol{\pi}(u_j,u)\right]. \\ &\le \frac{1}{(\alpha\theta)\cdot n_u} \cdot [n_v + n_u] = (\frac{n_u + n_v}{\alpha n_u}) \cdot \frac{1}{\theta}. \end{aligned}$$

For phase-f, the bound is the same and we omit the details. Note that phase-f re-uses the results in phase-r, more efficient than computing another PDU from scratch. Thus, Theorem 3 holds. □

5.4 Discussion on Settings of δ

In Theorem 3, BIRD achieves time complexity in $\widetilde{O}(n_u + n_v) = \widetilde{O}(\frac{n_u+n_v}{n_u} \cdot \frac{1}{\delta})$. This is achieved by setting $\theta = \frac{c^2 \cdot \delta}{48 \cdot L \cdot \gamma_m}$ and following typical setting $\delta = \frac{1}{n_u}$ in [40, 43, 58, 59]. When considering a full scope range that choosing $\delta << \frac{1}{n_u}$, the previous bound is not applicable. Meanwhile, we secure that our BIRD algorithm finishes in at most $\widetilde{O}(m)$ time when $\delta << \frac{1}{n_u}$, by setting $\theta = 0$. In this way, PDU conducts no push discretization steps and BIRD degenerates into a bidirectional variant of SSP, which adopts a deterministic sequential push strategy. This can be verified that in each iteration, the total residue will be reduced by α proportion as all non-zero residues will be pushed. With at most $\log_{1-\alpha}(c \cdot \delta)$ iterations, the (c, p_f) approximation achieves in $O(\log_{1-\alpha}(c \cdot \delta)) \cdot m = \widetilde{O}(m)$ time. For $\forall \delta \ge 0$ as a full scope, BIRD achieves an overall $\widetilde{O}(\min\{\frac{n_u+n_v}{n \cdot \delta}, m\})$ time complexity, no worse than ABHPP.

6 EXTRA RELATED WORK AND DISCUSSION

PPR computation is relevant to our work as BHPP can be defined in view of PPR on graph $\widehat{G}$ by materializing $\mathbf{P} = \mathbf{U} \cdot \mathbf{V}$. On decades, PPR has been studied widely in recent studies and the majority focus on speeding and scaling up the single-source [12, 21, 34, 35, 39, 44, 59, 63, 66], single-target [41, 42, 55, 56, 58] and top-k [22, 41, 43, 59, 62] PPR queries on general graphs. Among them, [12, 21, 22, 39] utilize a large number of random walks to estimate PPR and [10, 42] use deterministic graph traversal to compute exact PPR vectors. Subsequent works [41, 44, 58, 58, 59, 62, 63] tend to improve the scalability and efficiency in PPR computations by combining the deterministic graph traversal with Monte-Carlo in separate phases and several recent studies [23, 55, 56] explore to perform the deterministic graph traversal and the randomized Monte-Carlo method in an atomic step. As the single-source BHPP query on G can be approximated by answering both the single-source and single-target PPR on $\widehat{G}$, we summarize the theoretical results in Table 2 for a comparison of SOTA PPR techniques (RBS [56], FORA [59] and SpeedPPR [63]). In a word, PDU achieves the best time and space complexity on approximating both the single-source and single-target HPP. Inspired by RBS, PDU technique also incorporates randomness in each deterministic push simultaneously. The difference is, our whole BIRD algorithm focuses on weighted bipartite graph scenarios which requires a weight-dependent push strategy for answering BHPP queries. This adaptation brings in additional analytical difficulties especially when combined with estimator reusing techniques, requiring adapted proofs based on that in RBS. Other recent works focus on PPR computation with dynamic graphs [27, 33, 46, 48, 67] and its potential applications in graph neural network [36, 37].

Table 2: Comparison from PDU to other sota PPRs.

Technique	Single-Source	Single-Target	Space	Time
RBS [56]	×	✓	$\widetilde{O}(n_u^2)$	$\widetilde{O}(n)$
FORA [59]	✓	×	$\widetilde{O}(n_u^2)$	$\widetilde{O}(\sqrt{m \cdot n})$
SpeedPPR [63]	✓	×	$\widetilde{O}(n_u^2)$	$\widetilde{O}(m)$
PDU (ours)	✓	✓	$\widetilde{O}(m)$	$\widetilde{O}(n)$

Table 3: Statistics of eight real-world bipartite graphs.

Dataset	n_u	n_v	m	m/n	Type
Avito [7]	27,736	16,589	67,028	2.4	weighted
MovieLens [2]	6,040	3,706	1,000,209	165.6	weighted
Amazon-Games [6]	826,767	50,210	1,324,753	1.6	weighted
KDDCup [5]	255,170	1,848,114	2,766,393	10.8	weighted
Last.fm [4]	359,349	160,168	17,559,530	48.9	weighted
AOL [3]	4,811,647	1,632,788	10,741,953	2.2	weighted
Netflix [47]	480,189	17,770	100,480,507	209.2	unweighted
Orkut [52]	2,783,196	8,730,857	327,037,487	117.5	weighted

7 EXPERIMENTS

This section experimentally evaluates BIRD algorithm against state-of-the-art methods. We utilize original code of ABHPP for fair comparisons. All methods run on a single CPU core and are implemented in C++ and compiled by g++ 9.4.0 with -O3 optimization, and all experiments are conducted on a Linux machine with an Intel(R) Xeon(R) Gold 6326 CPU @ 2.90GHz CPU and 256GB RAM.

Datasets. We include eight real-world bipartite graphs commonly used in previous studies [15, 38, 57, 60, 65]. Table 3 summarizes the statistic of each dataset. In specific, *Avito, KDDCup*, and *AOL* are three click graphs that contain a set of queries in U and a set of URLs in V. *MovieLens, Last.fm,* and *Amazon-Games* are three user-item graphs that contain a set of users in U and a set of items in V. *Netflix* is a user-movie rating bipartite-graph and *Orkut* is a user-to-group affiliation bipartite network. Note that due to the space limits, part of the experimental results on graph *MovieLens*

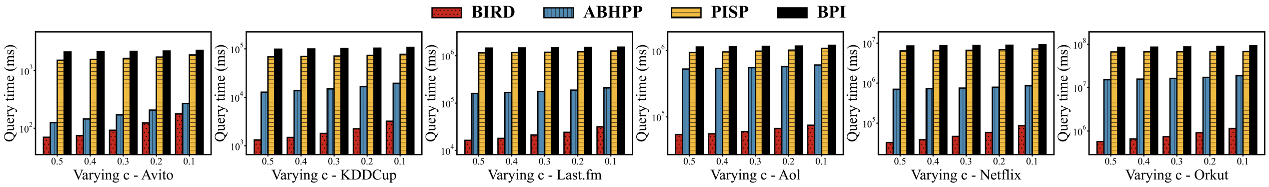

Figure 3: Query time varying relative error c.

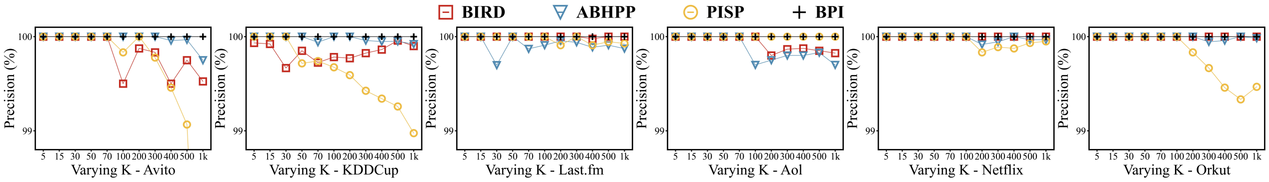

Figure 4: Query top-K precision varying K.

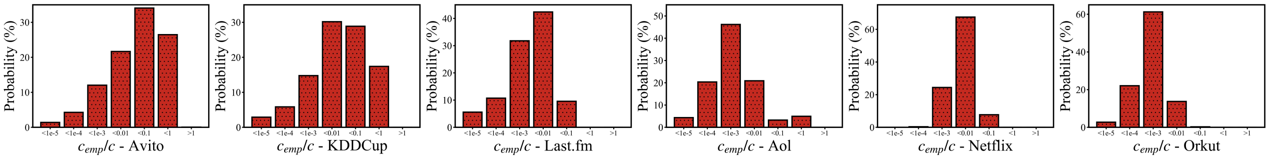

Figure 5: Empirical relative error distributions of BIRD.

and *Amazon-Games* are omitted on the main content without loss of generality, Interested readers can view such results in [1].

Baselines and settings. We compare our BIRD method to the three most competitive methods: BPI [49], PISP [9] and ABHPP [64]. To achieve a (c, p_f)-approximation of the BHPP query, we set the parameters in strict accordance with the theoretical analysis in Sec 3.3. Specifically, the BPI method has one parameter ϵ, the ℓ_1 error threshold. We set $\epsilon = \min_{u_i \in U} \frac{w_u}{w_{u_i}} \cdot \frac{c}{n_u}$ given the query node u and graph based on the analysis. PISP method is a combination of the PI and SP, we equally set $\epsilon_f = \epsilon_r = \frac{c}{2n_u}$ for both threshold. The ABHPP method has one overall error parameter ϵ and it self-decides the forward and reverse error. Besides, we set $\epsilon = \frac{c}{n_u}$ accordingly. We set the damping factor $\alpha = 0.15$, the failure probability $p_f = 0.1$ and the relative error parameter $c = 0.1$ by default if not specified.

Empirical Query Time. We evaluate the average execution time for each method using a query set namely Q containing randomly 100 selected source nodes. We vary the relative error parameter c, taking values from the $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. The results are presented in Figure 3. Based on the results, our BIRD method consistently delivers exceptional efficiency, outpacing all other methods across eight diverse datasets and various relative error settings. Specifically, the speed-up is often by an order of magnitude over the SOTA method ABHPP and reaches approximately 100× compared to BPI and PISP. Notably, when $c = 0.5$, BIRD achieves an acceleration around 50× compared against ABHPP on dataset *Netflix*. This underscores the robustness and superior capabilities of BIRD.

Empirical Precision and Relative Error. We verify the result quality by the top-K *Precision* and relative error distribution of BHPP queries following previous work on evaluating empirical relative errors [56, 59]. Specifically, given the ground truth node set U_k and approximate one $\widehat{U}_k$, the *Precision* equals $|U_k \cap \widehat{U}_k|/|U_k|$ as the proportion of nodes in $\widehat{U}_k$ which coincides in the ground truth results U_k. For obtaining the actual results, we run BPI for 100 iterations on each query and vary the number of top-K in Figure 4. Generally, BIRD, ABHPP and BPI achieve high quality results on all graphs with various K settings consistently while PISP show slightly diminished accuracy on datasets *Avito* and *KDDCup* with large K settings. This indicates that it may need tighter error threshold to achieve more accurate query results, resulting in more time costs. Besides, we compute the empirical relative errors for each query source node u as $c_{cmp} \in \{[\widehat{\mathcal{B}}_{\boldsymbol{u}}(u_i) - \mathcal{B}_{\boldsymbol{u}}(u_i)]/\mathcal{B}_{\boldsymbol{u}}(u_i) | \forall u_i \in U\}$. Then, we report the distribution of the values c_{cmp}/c across all nodes in Figure 5. Note that all $c_{cmp}/c < 1$ indicates the empirical error of the BIRD meets the (c, p_f) approximation requirement. Upon observation, we claim that the empirical values c_{cmp}/c are consistently smaller than 1 across all datasets. Particularly, on graph *Last.fm*, *Netflix* and *Orkut*, the empirical relative errors are smaller than c by up to two order of magnitude. This demonstrates the correctness and query efficacy of our BIRD.

Varying Failure Probability p_f. With a default failure probability of $p_f = 0.1$, our BIRD algorithm demonstrates high performance. Though reducing p_f to a vanishingly small value, i.e., $O(\frac{1}{n^2})$, introduces an $O(\log n)$ computational overhead, as detailed in Section 5.2. To evaluate the algorithm's robustness, we tested a range of failure probabilities $p_f \in \{0.1, 0.01, 0.001, 1e^{-4}, 1e^{-5}, 1e^{-6}\}$, with results shown in Figure 7 for the *Netflix* and *Orkut* datasets. While the running times for algorithms ABHPP, BPI, and PISP remain stable, BIRD shows a slight decrease in efficiency as the failure probability p_f approaches zero. Nonetheless, BIRD consistently outperforms ABHPP by an order of magnitude, demonstrating its robustness.

Evaluation on Reuse Strategy . Recall that in BIRD, we introduced the reuse strategy to expedite the computation process in the forward-phase by utilizing the intermediate results from the Reverse-phase. To demonstrate its effectiveness, we conducted an experiment by removing the reuse component and analyzing

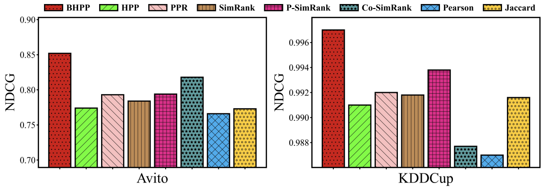

Figure 6: Query rewriting performance comparison.

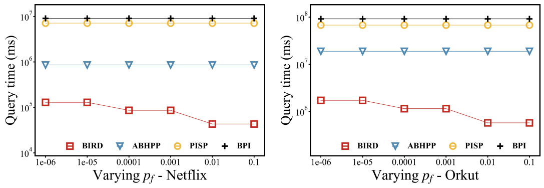

Figure 7: Query time varying failure probability p_f.

the time cost breakdown, as shown in Table 4, for our four primary datasets: *Last.fm*, *AOL*, *Netflix*, and *Orkut*. The results indicate that BIRD with the **reuse** strategy significantly reduces the computation overhead in the forward-phase, saving approximately 70% of the push operations across all datasets. This results in an average speed-up of about 3.5× for the forward-phase cost, leading to an overall computation acceleration of approximately 1.6×. These findings highlight the efficacy of our reuse strategy in leveraging intermediate results to significantly accelerate the computation.

Real-World Applications. To thoroughly assess BHPP's performance, we further conduct experiments in the domains of query rewriting and item recommendation, employing metrics as Normalized Discounted Cumulative Gain (NDCG) [29] and F1-score, respectively, following the settings in [11, 13, 64]. We compare BHPP against seven other competitive similarity measures, categorized as follows: (1) Simple methods, including Pearson's correlation coefficient [53] and Jaccard's coefficient [28]; (2) General similarity measurements, such as naive Personalized PageRank (PPR) [24], SimRank [30], and CoSimRank [61]; and (3) Bipartite similarity measurements, including HPP [18] and P-SimRank [19]. The performance comparison results are presented in Figure 6 for query rewriting and in Table 5 for item recommendation. From Figure 6, it is evident that BHPP consistently outperforms all other similarity measures on the two click graphs. Notably, BHPP shows a significant improvement of 3.0% over the second-best method, Co-SimRank, on the *Avito* dataset. Similarly, in the item recommendation task, as shown in Table 5, BHPP consistently achieves the highest F1-score across four user-item bipartite graphs. Its performance is particularly dominant on the *MovieLens* dataset, where it achieves a remarkable improvement of approximately 5.2% over the naive PPR. These results imply that BHPP, as a rising bipartite variant of naive PPR, is more adept at capturing the properties of bipartite graphs and demonstrates superior effectiveness as a similarity measurement specialized for the bipartite domain. Consequently, we posit that BHPP represents a significant enhancement to the entire family of PPR-related algorithms, and our scalability improvement is also valuable for applications on massive graphs.

Limitation. Though BIRD scales up well on massive graphs, we also notice that its acceleration on the small scale graph *Avito* is only 2 ∼ 3 times, not as sophisticated as that on large datasets. Meanwhile, its top-K precision is slightly falling behind the optimal interval, compared to the perfect performance achieved by BPI. This may remind users that when the graph scale is small with tolerable query time, i.e. within a second, a preferable alternative is to choose the classical method with impeccable precision.

Table 4: Evaluation on the reuse strategy. We separately show break down of Forward-phase (Fwd.) and Overall (All.).

Dataset	Last.fm		AOL		Netflix		Orkut	
	Fwd.	All.	Fwd.	All.	Fwd.	All.	Fwd.	All.
w **reuse**	5.4	20.3	8.5	44.8	17.6	58.5	195.3	755
w/o **reuse**	16.6	31.5	38.6	74.9	45.1	86.0	587.3	1147
Speed up	3.1×	1.6×	4.6×	1.7×	2.6×	1.5×	3.0×	1.53×

Table 5: Item recommendation performance. The best score is marked in bold and the runner-up one is underlined.

Similarity Measurement	F1-Score@k			
	DBLP	MovieLens	Last.fm	Amazon-Games
BHPP (by BIRD)	**0.165**	**0.337**	**0.266**	**0.213**
HPP [18]	0.139	0.187	0.258	0.169
PPR [24]	0.147	0.285	0.237	0.162
SimRank [30]	0.15	0.206	0.198	0.101
P-SimRank [19]	0.127	0.188	0.187	0.108
CoSimRank [61]	0.114	0.158	0.253	0.136
Pearson [53]	0.037	0.087	0.108	0.049
Jaccard [28]	0.157	0.226	0.245	0.07

8 CONCLUSION

In this paper, we introduce BIRD to address the problem of approximating single-source BHPP queries on weighted undirected bipartite graphs. Our method achieves the superior time complexity in $\widetilde{O}(n)$ with typical settings. BIRD incorporates a novel technique called PDU to fit with the bidirectional reusing based procedure to optimize the computational efficiency. Empirical evaluations demonstrate that BIRD achieves orders of magnitude speed-up.

9 PROOFS

Lemma 3. Recall that $\boldsymbol{\pi}^{(\ell)}(u_i, u_j) = \alpha(1-\alpha)^{\ell} \cdot \mathbf{P}^{\ell}(u_i, u_j)$, which suggests that the lemma can be proved by deriving for $\forall \ell \geq 1$, Equation $\mathbf{P}^{\ell}(u_i, u_j)/w_{u_j} = \mathbf{P}^{\ell}(u_j, u_i)/w_{u_i}$. By induction for $\ell = 1$,

$$\begin{aligned}\frac{\mathbf{P}(u_i,u_j)}{w_{u_j}} &= \frac{1}{w_{u_j}} \sum\nolimits_{v \in N_{u_i} \cap N_{u_j}} \frac{w(u_i,v)}{w_{u_i}} \cdot \frac{w(v,u_j)}{w_v} \\ &= \frac{1}{w_{u_i}} \sum\nolimits_{v \in N_{u_i} \cap N_{u_j}} \frac{w(u_j,v)}{w_{u_j}} \cdot \frac{w(v,u_i)}{w_v} = \frac{\mathbf{P}(u_j,u_i)}{w_{u_i}}\end{aligned}$$

is valid. Assume that above Equation holds for ℓ, then in $\ell + 1$,

$$\begin{aligned}\frac{\mathbf{P}^{\ell+1}(u_i,u_j)}{w_{u_j}} &= \sum\nolimits_{u \in N^{(2)}_{u_j}} \frac{\mathbf{P}^{\ell}(u_i,u) \cdot \mathbf{P}^{\ell}(u,u_j)}{w_{u_j}} = \sum\nolimits_{u \in N^{(2)}_{u_j}} \frac{\mathbf{P}^{\ell}(u_i,u) \cdot \mathbf{P}(u_j,u)}{w_u} \\ &= \sum\nolimits_{u \in N^{(2)}_{u_j}} \frac{\mathbf{P}^{\ell}(u,u_i) \cdot \mathbf{P}(u_j,u)}{w_{u_i}} = \frac{\mathbf{P}^{\ell+1}(u_j,u_i)}{w_{u_i}}.\end{aligned}$$

Therefore, the lemma holds by completing the math induction.

Fact 2 (Law of Total Variance [16]). *Given two random variables* X *and* Y, $\mathbb{V}\mathrm{ar}[X] = \mathbb{E}[\mathbb{V}\mathrm{ar}[X \mid Y]] + \mathbb{V}\mathrm{ar}[\mathbb{E}[X \mid Y]]$ *holds.*

Lemma 5. Recall in the ℓ-th iteration, the reverse residue vector $\boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(\cdot)$ in `phase-r` receives discretized increments from n_v, which in turn come from the ℓ-th vector $\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(\cdot)$. For $\forall u_j \in U$,

$$\begin{aligned}
&\mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(u_j)\right] = \mathbb{E}\left[\mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(u_i)|\boldsymbol{r}_{\boldsymbol{v}}(\cdot)\right]\right] \\
&= \sum_{(v_i,u_j)\in E} \mathbb{E}\left[X_d(v_i,u_j)|\boldsymbol{r}_{\boldsymbol{v}}(\cdot)\right] = \sum_{v_i\in N_{u_j}} \frac{w(u_j,v_i)}{w_{v_i}} \cdot \mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{v}}(v_i)\right] \\
&= \sum_{v_i\in N_{u_j}} \frac{w(u_j,v_i)}{w_{v_i}} \cdot \mathbb{E}\left[\mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{v}}(v_i)|\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(\cdot)\right]\right] \quad \textbf{(By Lemma 4)} \\
&= \sum_{v_i\in N_{u_j}} \frac{w(u_j,v_i)}{w_{v_i}} \sum_{u_i\in N_{v_j}} (1-\alpha)\cdot\frac{w(v_j,u_i)}{w_{v_j}} \cdot \mathbb{E}\left[r_r^{(\ell)}(u_i)\right] \\
&= \sum_{u_i\in N_{u_j}^{(2)}} (1-\alpha)\cdot \mathbf{P}(u_i,u_j)\cdot \mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{f}}^{(\ell)}(u_i)\right].
\end{aligned}$$

When $\ell = 0, \mathbb{E}\left[r_u^{(\ell)}(x)\right] = 0$ for all $\forall x \neq u$ and $E[\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(u)] = 1$. Thus, $\mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{r}}^{(0)}(\cdot)\right] = \boldsymbol{\pi}^{(0)}(u,\cdot)/\alpha$ holds. By mathematical induction, we assume it still holds at level ℓ. Based on recursive property that

$$\boldsymbol{\pi}^{(\ell+1)}(u,u_i) = \sum\nolimits_{u_j\in N^{(2)}(u_i)} (1-\alpha)\cdot\mathbf{P}(u_j,u_i)\cdot\boldsymbol{\pi}^{(\ell)}(u,u_j),$$

we then conclude that $\mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(u_i)\right] = \boldsymbol{\pi}^{(\ell+1)}(u,u_i)/\alpha$. Next for `phase-f`, each safe node $\forall u_i \in U_\gamma$ directly shares the conclusion as

$$\mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{f}}^{(\ell)}(u_i)\right] = \mathbb{E}\left[\frac{w_{u_i}}{w_u}\cdot\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(u_i)\right] = \frac{w_{u_i}}{w_u}\cdot\boldsymbol{\pi}^{(\ell)}(u_i,u)/\alpha = \frac{\boldsymbol{\pi}^{(\ell)}(u,u_i)}{\alpha}.$$

For each unsafe node $\forall u_j \in U \setminus U_\gamma$, it analogously holds the same prove process because it conducts the same `PDU` process while just changing its direction. Thus, $\mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{f}}^{(\ell)}(u_j)\right] = \boldsymbol{\pi}^{(\ell)}(u,u_j)/\alpha$ also holds.

Fact 3 (The Jensen's Inequality [50].). *For convex function $\varphi : (a,b) \rightarrow \mathcal{R}$, inequality $\varphi(\lambda_1 x_1 + \cdots + \lambda_n x_n) \le \lambda_1\varphi(x_1) + \cdots + \lambda_n\varphi(x_n)$ holds for any $\lambda_1, \ldots, \lambda_n$ satisfying $\lambda_1 +, \cdots + \lambda_n = 1$.*

Lemma 6. We first bound the variance of each discrete increment

$$\mathbb{V}\mathrm{ar}\left[X_d(v_i,u_j)|\boldsymbol{r}_{\boldsymbol{v}}(\cdot)\right] \le \mathbb{E}\left[X_d^2(u_i,v_j)|\boldsymbol{r}_{\boldsymbol{v}}(\cdot)\right] = \theta^2\cdot\frac{w(u_j,v_i)\cdot\boldsymbol{r}_{\boldsymbol{v}}(\cdot)}{w_{u_j}\cdot\theta}.$$

Based on Fact 2, the variance for $\forall u_j \in U$, $\mathbb{V}\mathrm{ar}\left[\boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(u_j)\right]$ equals

$$\underbrace{\mathbb{E}\left[\ \mathbb{V}\mathrm{ar}\left[\boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(u_j)|\boldsymbol{r}_{\boldsymbol{v}}(\cdot)\right]\right]}_{\text{denoted as } \mathcal{E}_{\mathcal{V}}} + \underbrace{\mathbb{V}\mathrm{ar}\left[\mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(u_j)|\boldsymbol{r}_{\boldsymbol{v}}(\cdot)\right]\right]}_{\text{denoted as } \mathcal{V}_{\mathcal{E}}}.$$

$$\begin{aligned}
\mathcal{E}_{\mathcal{V}} &= \theta\cdot\sum_{v_i\in u_j}\frac{w(u_j,v_i)}{w_{u_j}}\cdot\mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{v}}(v_i)\right] \\
&= \theta\cdot\sum_{v_i\in u_j}\frac{w(u_j,v_i)}{w_{u_j}}\cdot\sum_{u_i\in N_{v_i}}(1-\alpha)\cdot\frac{w(v_i,u_i)}{w_{v_i}}\cdot\mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{f}}^{(\ell)}(u_i)\right] \\
&= \theta\cdot\sum_{v_i\in u_j}\frac{w(u_j,v_i)}{w_{u_j}}\cdot\sum_{u_i\in N_{v_i}}(1-\alpha)\cdot\frac{w(v_i,u_i)}{w_{v_i}}\cdot\boldsymbol{\pi}^{(\ell)}(u_i,u) \\
&= \theta\cdot\boldsymbol{\pi}^{(\ell+1)}(u_j,u);
\end{aligned}$$

$$\mathcal{V}_{\mathcal{E}} = \mathbb{V}\mathrm{ar}\left[\sum_{v_i\in N_{u_j}}\frac{w(u_j,v_i)\cdot\boldsymbol{r}_{\boldsymbol{v}}(v_i)}{w_{u_j}}\right] \le \sum_{v_i\in N_{u_j}}\frac{w(u_j,v_i)}{w_{u_j}}\cdot\mathbb{V}\mathrm{ar}\left[\boldsymbol{r}_{\boldsymbol{v}}(v_i)\right]$$

holds based on the convex property of the variance function in Fact 3. Next, we aim to bound the variance of residue $\boldsymbol{r}_{\boldsymbol{v}}(v_i)$. By applying the total variance law again and repeat similar prove process above, we can further derive that $\mathbb{V}\mathrm{ar}\left[\boldsymbol{r}_{\boldsymbol{v}}(v_i)\right]$ is at most

$$\sum_{u_i\in N_{v_i}}(1-\alpha)\cdot\frac{w(v_i,u_i)}{w_{v_i}}\cdot\left[\theta\cdot\boldsymbol{\pi}^{(\ell)}(u_i,u) + (1-\alpha)\cdot\mathbb{V}\mathrm{ar}\left[r_u^{(\ell)}(u_i)\right]\right].$$

Combining with $\mathcal{E}_{\mathcal{V}}$ and $\mathcal{V}_{\mathcal{E}}$, we have $\mathbb{V}\mathrm{ar}\left[\boldsymbol{r}_{\boldsymbol{r}}^{(\ell+1)}(u_j)\right] \le$

$$2\theta\cdot\boldsymbol{\pi}^{(\ell+1)}(u_j,u) + \sum\nolimits_{u_i\in N_{u_j}^{(2)}}(1-\alpha)^2\cdot\mathbf{P}(u_i,u_j)\cdot\mathbb{V}\mathrm{ar}\left[\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(u_i)\right].$$

Based on this inequality, we have $\mathbb{V}\mathrm{ar}\left[\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(u_j)\right] \le 2\theta\cdot\boldsymbol{\pi}^{(\ell)}(u_j,u)$. We next consider the variance of $\widehat{\mathcal{B}}^{(\ell)}(u,u_j) = \boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(u_j) + \boldsymbol{r}_{\boldsymbol{f}}^{(\ell)}(u_j)$. Note that in `phase-f`, we re-use the estimation of the reverse residue for each safe node $\forall u_j \in U_\gamma$. It then follows that

$$\mathbb{V}\mathrm{ar}\left[\widehat{\mathcal{B}}^{(\ell)}(u,u_j)\right] = \mathbb{V}\mathrm{ar}\left[(1+\frac{w_{u_j}}{w_u})\cdot\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(u_j)\right] = (1+\frac{w_{u_j}}{w_u})^2\cdot\mathbb{V}\mathrm{ar}$$
$$\left[\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(u_j)\right] \le (1+\gamma)^2\cdot 2\theta\cdot\boldsymbol{\pi}^{(\ell)}(u_j,u) \le 2\theta\cdot(1+\gamma)^2\cdot\mathcal{B}^{(\ell)}(u,u_j).$$

Besides, for each node $u_j \in U \setminus U_\gamma$, since these nodes share no computation in the reverse residue, we thus need to bound the variance of $\boldsymbol{r}_{\boldsymbol{f}}^{(\ell)}$ from scratch. By performing similar proof steps in reverse residue, we have $\mathbb{V}\mathrm{ar}\left[\boldsymbol{r}_{\boldsymbol{f}}^{(\ell)}(u_j)\right] \le 2\theta\cdot\boldsymbol{\pi}^{(\ell)}(u,u_j)$ so that

$$\mathbb{V}\mathrm{ar}\left[\widehat{\mathcal{B}}^{(\ell)}(u,u_j)\right] \le 2\cdot\left(\mathbb{V}\mathrm{ar}\left[\boldsymbol{r}_{\boldsymbol{f}}^{(\ell)}(u_j)\right] + \mathbb{V}\mathrm{ar}\left[\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(u_j)\right]\right) \le 4\theta\cdot\mathcal{B}^{(\ell)}(u,u_j).$$

We then finish proof for all *safe* and *unsafe* nodes in U.

Equation (15). In `phase-r`, when conducting `PDU` from node u_i to v_j, $\mathbb{C}_r^{(\ell)}(u_i,v_j)$ equals 1 if $X(u_i,v_j) = w(u_i,v_j)/w_{v_j}\cdot(1-\alpha)\cdot\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(u_i) \ge \theta$. Otherwise $\mathbb{C}_r^{(\ell)}(u_i,v_j)$ equals 1 with probability $\frac{X(u_i,v_j)}{\theta}$ and 0 with $1-\frac{X(u_i,v_j)}{\theta}$. Thus, $\mathbb{E}\left[\mathbb{C}_r^{(\ell)}(v_i,u_j)|\boldsymbol{r}_{\boldsymbol{v}}(\cdot)\right] = \frac{w(u_j,v_i)}{\theta\cdot w_{v_i}}\cdot\boldsymbol{r}_{\boldsymbol{v}}(v_i)$ and $\mathbb{E}\left[\mathbb{C}_r^{(\ell)}(u_i,v_j)|\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(\cdot)\right] = \frac{(1-\alpha)\cdot w(v_j,u_i)}{\theta\cdot w_{v_j}}\cdot\boldsymbol{r}_{\boldsymbol{r}}^{(\ell)}(u_i)$. We further have:

$$\begin{aligned}
&\sum_{\ell=0}^{L-1}\sum_{(u,v)\in E}\mathbb{E}\left[\mathbb{C}_r^{(\ell)}(u,v)\right] = \sum_{\ell=0}^{L-1}\sum_{(v_j,u_i)\in E}\mathbb{E}\left[\mathbb{C}_r^{(\ell)}(u_i,v_j)\right] \\
&\le \frac{(1-\alpha)}{\alpha\theta}\cdot\sum_{\ell=0}^{L-1}\sum_{(v_j,u_i)\in E}\frac{w(v_j,u_i)}{w_{v_j}}\cdot\boldsymbol{\pi}^{(\ell)}(u_i,u) \\
&\le \frac{(1-\alpha)}{\alpha\theta}\cdot\sum_{v_j\in V}\sum_{\ell=1}^{\infty}\sum_{u_i\in N_{v_j}}\frac{w(v_j,u_i)}{w_{v_j}}\cdot\boldsymbol{\pi}^{(\ell)}(u_i,u) \\
&\le \frac{(1-\alpha)}{\alpha\theta}\cdot\sum_{v_j\in V}\mathbb{P}[v_j\rightarrow u]; \quad \text{And then } \sum_{\ell=0}^{L-1}\sum_{(u,v)\in E}\mathbb{E}\left[\mathbb{C}_r^{(\ell)}(v,u)\right] \\
&= \frac{1}{\theta}\cdot\sum_{\ell=0}^{L-1}\sum_{(u_j,v_i)\in E}\frac{w(u_j,v_i)}{w_{v_i}}\cdot\mathbb{E}\left[\boldsymbol{r}_{\boldsymbol{v}}(v_i)\right] \quad \textbf{(Expectation Property)} \\
&\le \frac{1-\alpha}{\theta}\cdot\sum_{\ell=0}^{L-1}\sum_{(u_j,v_i)\in E}\frac{w(u_j,v_i)}{w_{v_i}}\cdot\sum_{u_i\in N_{v_i}}\frac{w(v_i,u_i)}{w_{v_i}}\cdot\mathbb{E}\left[r_r^{(\ell)}(u_i)\right] \\
&= \frac{1}{\alpha\theta}\cdot\sum_{\ell=0}^{L-1}\sum_{u_j\in U}\sum_{u_i\in N_{u_j}^{(2)}}(1-\alpha)\cdot\frac{w(u_j,v_i)}{w_{v_i}}\cdot\frac{w(v_i,u_i)}{w_{v_i}}\cdot\boldsymbol{\pi}^{(\ell)}(u_i,u) \\
&= \frac{1}{\alpha\theta}\cdot\sum_{\ell=0}^{L-1}\sum_{u_j\in U}\boldsymbol{\pi}^{(\ell+1)}(u_j,u) \le \frac{1}{\alpha\theta}\cdot\sum_{u_j\in U}\boldsymbol{\pi}(u_j,u).
\end{aligned}$$

ACKNOWLEDGMENTS

This research is supported by Singapore MOE AcRF Tier-1 Seed Funding (RS05/21) and Tier-2 funding (MOE-T2EP20122-0003).

REFERENCES

[1] [n.d.]. https://drive.google.com/file/d/1s0D8yT9RSIKKzJynh34H9uEogUUB6_7A/view?usp=sharing.

[2] 2003. MovieLens 1M Dataset. https://grouplens.org/datasets/movielens.

[3] 2006. AOL Query Logs. http://www.cim.mcgill.ca/~dudek/206/Logs/AOL-user-ct-collection.

[4] 2010. Last.fm Dataset Version 1.2. http://ocelma.net/MusicRecommendationDataset/lastfm-360K.html.

[5] 2012. KDD Cup 2012, Track 2. https://www.kaggle.com/c/kddcup2012-track2.

[6] 2014. Amazon product data. https://jmcauley.ucsd.edu/data/amazon.

[7] 2015. Avito Context Ad Clicks. https://www.kaggle.com/c/avito-context-ad-clicks/data.

[8] Tasos Anastasakos, Dustin Hillard, Sanjay Kshetramade, and Hema Raghavan. 2009. A collaborative filtering approach to ad recommendation using the query-ad click graph. In *Proceedings of the 18th ACM conference on Information and knowledge management*. https://doi.org/10.1145/1645953.1646267

[9] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S Mirrokni, and Shang-Hua Teng. 2007. Local computation of pagerank contributions. In *Algorithms and Models for the Web-Graph: 5th International Workshop, WAW 2007, San Diego, CA, USA, December 11-12, 2007. Proceedings 5*. Springer, 150–165.

[10] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE, 475–486.

[11] Ioannis Antonellis, Hector Garcia-Molina, and Chi-Chao Chang. 2008. Simrank++ query rewriting through link analysis of the clickgraph (poster). In *Proceedings of the 17th international conference on World Wide Web*. 1177–1178.

[12] 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.

[13] Alejandro Bellogin, Pablo Castells, and Ivan Cantador. 2011. Precision-oriented evaluation of recommender systems: an algorithmic comparison. In *Proceedings of the fifth ACM conference on Recommender systems*. 333–336.

[14] 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.

[15] Chih-Ming Chen, Chuan-Ju Wang, Ming-Feng Tsai, and Yi-Hsuan Yang. 2019. Collaborative similarity embedding for recommender systems. In *The World Wide Web Conference*. 2637–2643.

[16] Kai Lai Chung. 2001. *A course in probability theory*. Academic press.

[17] Giulio Cimini, Alessandro Carra, Luca Didomenicantonio, and Andrea Zaccaria. 2022. Meta-validation of bipartite network projections. *Communications Physics* 5, 1 (2022), 76.

[18] Hongbo Deng, Michael R Lyu, and Irwin King. 2009. A generalized co-hits algorithm and its application to bipartite graphs. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 239–248.

[19] Prasenjit Dey, Kunal Goel, and Rahul Agrawal. 2020. P-Simrank: Extending Simrank to Scale-free bipartite networks. In *Proceedings of The Web Conference 2020*. 3084–3090.

[20] Alessandro Epasto, Jon Feldman, Silvio Lattanzi, Stefano Leonardi, and Vahab Mirrokni. 2014. Reduce and aggregate: similarity ranking in multi-categorical bipartite graphs. In *Proceedings of the 23rd international conference on World wide web*. 349–360.

[21] 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.

[22] Yasuhiro Fujiwara, Makoto Nakatsuji, Makoto Onizuka, and Masaru Kitsuregawa. 2012. Fast and exact top-k search for random walk with restart. *arXiv preprint arXiv:1201.6566* (2012).

[23] Qian Ge, Yu Liu, Yinghao Zhao, Yuetian Sun, Lei Zou, Yuxing Chen, and Anqun Pan. [n.d.]. Efficient and Accurate SimRank-based Similarity Joins: Experiments, Analysis, and Improvement. ([n. d.]).

[24] Taher H Haveliwala. 2002. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*. 517–526.

[25] Taher H Haveliwala and Sepandar D Kamvar. 2003. *The second eigenvalue of the Google matrix*. Technical Report. Citeseer.

[26] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. 2004. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems* (Jan 2004), 5–53. https://doi.org/10.1145/963770.963772

[27] Guanhao Hou, Qintian Guo, Fangyuan Zhang, Sibo Wang, and Zhewei Wei. 2023. Personalized PageRank on Evolving Graphs with an Incremental Index-Update Scheme. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[28] Paul Jaccard. 1912. The distribution of the flora in the alpine zone. 1. *New phytologist* 11, 2 (1912), 37–50.

[29] Kalervo Jaervelin and Jaana Kekaelaeinen. 2017. IR evaluation methods for retrieving highly relevant documents. *ACM SIGIR forum* (2017).

[30] Glen Jeh and Jennifer Widom. 2002. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 538–543.

[31] Yehuda Koren. 2008. Factorization meets the neighborhood. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. https://doi.org/10.1145/1401890.1401944

[32] Lina Li, Cuiping Li, Hong Chen, and Xiaoyong Du. 2013. MapReduce-Based SimRank Computation and Its Application in Social Recommender System. In *2013 IEEE International Congress on Big Data*. https://doi.org/10.1109/bigdata.congress.2013.26

[33] Yiming Li, Yanyan Shen, Lei Chen, and Mingxuan Yuan. 2023. Zebra: When Temporal Graph Neural Networks Meet Temporal Personalized PageRank. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1332–1345.

[34] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, Hongyang Chen, Hongchao Qin, and Guoren Wang. 2023. Efficient Personalized PageRank Computation: The Power of Variance-Reduced Monte Carlo Approaches. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–26.

[35] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, and Guoren Wang. 2022. Efficient personalized PageRank computation: A spanning forests sampling based approach. In *Proceedings of the 2022 International Conference on Management of Data*. 2048–2061.

[36] Ningyi Liao, Dingheng Mo, Siqiang Luo, Xiang Li, and Pengcheng Yin. 2022. SCARA: scalable graph neural networks with feature-oriented optimization. *arXiv preprint arXiv:2207.09179* (2022).

[37] Ningyi Liao, Dingheng Mo, Siqiang Luo, Xiang Li, and Pengcheng Yin. 2024. Scalable decoupling graph neural network with feature-oriented optimization. *The VLDB Journal* 33, 3 (2024), 667–683.

[38] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. 2019. Efficient (α, β)-core computation: An index-based approach. In *The World Wide Web Conference*. 1130–1141.

[39] Qin Liu, Zhenguo Li, John CS Lui, and Jiefeng Cheng. 2016. Powerwalk: Scalable personalized pagerank via random walks with vertex-centric decomposition. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 195–204.

[40] 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.

[41] 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.

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

[43] Peter A Lofgren, Siddhartha Banerjee, Ashish Goel, and Comandur 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.

[44] Siqiang Luo, Xiaokui Xiao, Wenqing Lin, and Ben Kao. 2019. BATON: Batch one-hop personalized PageRanks with efficiency and accuracy. *IEEE Transactions on Knowledge and Data Engineering* 32, 10 (2019), 1897–1908.

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

[46] Dingheng Mo and Siqiang Luo. 2021. Agenda: Robust personalized PageRanks in evolving graphs. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 1315–1324.

[47] Netflix. 2009. Netflix Prize Data Set. (2009). http://archive.ics.uci.edu/ml/datasets/Netflix+Prize

[48] Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. 2015. Efficient pagerank tracking in evolving networks. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 875–884.

[49] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1998. *The pagerank citation ranking: Bring order to the web*. Technical Report. Technical report, stanford University.

[50] Endre Pap and Mirjana Štrboja. 2010. Generalization of the Jensen inequality for pseudo-integral. *Information Sciences* 180, 4 (2010), 543–548.

[51] Georgios A Pavlopoulos, Panagiota I Kontou, Athanasia Pavlopoulou, Costas Bouyioukos, Evripides Markou, and Pantelis G Bagos. 2018. Bipartite graphs in systems biology and medicine: a survey of methods and applications. *GigaScience* 7, 4 (Apr 2018). https://doi.org/10.1093/gigascience/giy014

[52] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. https://networkrepository.com

[53] Ming-Sheng Shang, Yan Fu, and Duan-Bin Chen. 2008. Personal recommendation using weighted bipartite graph projection. In *2008 International Conference on Apperceiving Computing and Intelligence Analysis*. IEEE, 198–202.

[54] Michael Stauffer, Thomas Tschachtli, Andreas Fischer, and Kaspar Riesen. 2017. A survey on applications of bipartite graph edit distance. In *Graph-Based Representations in Pattern Recognition: 11th IAPR-TC-15 International Workshop, GbRPR 2017, Anacapri, Italy, May 16–18, 2017, Proceedings 11*. Springer, 242–252.

[55] Hanzhi Wang and Zhewei Wei. 2023. Estimating Single-Node PageRank in $\tilde{O}(min\{d_t, \sqrt{m}\})$ Time. *Proceedings of the VLDB Endowment* 16, 11 (2023), 2949–2961.

[56] 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.

[57] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2019. Vertex Priority Based Butterfly Counting for Large-scale Bipartite Networks. *PVLDB* (2019).

[58] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. Hubppr: effective indexing for approximate personalized pagerank. *Proceedings of the VLDB Endowment* 10, 3 (2016), 205–216.

[59] 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.

[60] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.

[61] Xiuli Wang, Zhuoming Xu, Xiutao Xia, and Chengwang Mao. 2017. Computing user similarity by combining simrank++ and cosine similarities to improve collaborative filtering. In *2017 14th Web Information Systems and Applications Conference (WISA)*. IEEE, 205–210.

[62] 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.

[63] Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. 2021. Unifying the global and local approaches: an efficient power iteration with forward push. In *Proceedings of the 2021 International Conference on Management of Data*. 1996–2008.

[64] Renchi Yang. 2022. Efficient and Effective Similarity Search over Bipartite Graphs. In *Proceedings of the ACM Web Conference 2022*. 308–318.

[65] Renchi Yang, Jieming Shi, Keke Huang, and Xiaokui Xiao. 2022. Scalable and Effective Bipartite Network Embedding. In *Proceedings of the 2022 International Conference on Management of Data*. 1977–1991.

[66] Minji Yoon, Jinhong Jung, and U Kang. 2018. Tpa: Fast, scalable, and accurate method for approximate random walk with restart on billion scale graphs. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1132–1143.

[67] Zulun Zhu, Sibo Wang, Siqiang Luo, Dingheng Mo, Wenqing Lin, and Chunbo Li. 2024. Personalized PageRanks over Dynamic Graphs–The Case for Optimizing Quality of Service. In *Proceedings of the 2024 IEEE 40th International Conference on Data Engineering*.