Efficient Algorithms for Group Hitting Probability Queries on Large Graphs

Qintian Guo*, Dandan Lin*, Sibo Wang, Raymond Chi-Wing Wong, and Wenqing Lin

Abstract—Given a source node s and a target node t, the hitting probability tells us how likely an α-terminating random walk (which stops with probability α at each step) starting from s can hit t before it stops. This concept originates from the hitting time, a classic concept in random walks. In this paper, we focus on the group hitting probability (GHP) where the target is a set of nodes, measuring the node-to-group structural proximity. For this group version of the hitting probability, we present efficient algorithms for two types of GHP queries: the pairwise query which returns the GHP value of a target set T with respect to (w.r.t.) a source node s, and the top-k query which returns the top-k target sets with the largest GHP value w.r.t. a source node s.
We first develop an efficient algorithm named SAMBA for the pairwise query, which is built on a group local push algorithm tailored for GHP, with rigorous analysis for correctness. Next, we show how to speed up SAMBA by combining the group local push algorithm with the Monte Carlo approach, where GHP brings new challenges as it might need to consider every hop of the random walk. We tackle this issue with a new formulation of the GHP and show how to provide approximation guarantees with a detailed theoretical analysis. With SAMBA as the backbone, we develop an iterative algorithm for top-k queries, which adaptively refines the bounds for the candidate target sets, and terminates as soon as it meets the stopping condition, thus saving unnecessary computational costs. We further present an optimization technique to accelerate the top-k query, improving its practical performance. Extensive experiments show that our solutions are orders of magnitude faster than their competitors.

Index Terms—Graph algorithms, graphs and networks, data mining.

1 Introduction

Measuring node-to-node similarity or proximity in graphs is a fundamental tool for various graph-based mining applications [1], [2], [3], [4], [5], and has been recognized as an important research problem in the data mining community. One common approach to capture "similarity" in the literature is *random-walk-based approaches* due to its effectiveness and efficiency, resulting in a lot of research studies [6], [7], [8], [9], [10], [11] in the communities of database and data mining. Among them, *hitting time* [12], a classic concept in random walk, finds many real-world applications, such as link prediction [3], product recommendation [13], query suggestions [14] and collaborative filtering [15].

Given a graph G, hitting time tells us the expected number of steps a random walk takes to hit a target node t from a source node s *for the first time*. The *fewer* steps a random walk from s takes to hit t, the *more relevant* (similar) node t to s is from the perspective of s. In [16], [17], it has been proved that for an α-terminating random walk, the hitting time $h(s,t)$ from s to t can be obtained by exploiting the probability that an α-terminating random walk starting from node s hits t before terminating, denoted as $f(s,t)$. That is, if $f(s,t)$ is higher, a random walk from s takes fewer steps to hit t for the first time. Hence, solving hitting probability queries is equivalent to solving the hitting time problem.

In this paper, we consider a generalized case of the hitting probability problem where the target is a node-set T, called the *Group Hitting probability* (GHP). It is easy to realize that the original hitting probability $f(s,t)$ of a target node t is a special case of our GHP problem, where the target set T contains only one node t. The motivation for adopting a group target setting to measure node-to-group proximity finds validation within the existing literature. In [18], Grady presents a group-based algorithm to address the image segmentation problem. In Grady's solution, it treats images as graphs, with pixels as nodes and edges connecting adjacent pixels. Each edge is assigned a real-valued weight, indicating the likelihood of a random walker moving across that edge. Given a small subset of pixels with pre-defined labels, it computes the probability that a random walker, starting from each unlabeled pixel, will reach one of the labeled pixel groups. This information is then utilized to assign the pixel to the group label that corresponds to the highest calculated probability. This process facilitates the attainment of a superior image segmentation outcome. In [13], given a graph G with m nodes that share the same event Q, Guan et al. propose an approach that employs the average proximity between a node in these m nodes and the remaining $m-1$ nodes, to assess whether any structural correlation exists between the event Q and the graph G. The essence of this approach lies in accurately measuring the proximity between a node and a set or group of nodes. Guan et al. demonstrate that group hitting time serves as a suitable choice, effectively identifying events highly correlated with the graph structure.

Other applications of the node-to-group proximity measure include: (1) *Group Suggestions*. In practical scenarios, both Facebook and LinkedIn offer services to provide personalized recommendations, assisting users in discovering groups, as stated

- *Q. Guo and D. Lin are joint first authors.
- D. Lin is the corresponding author.
- Q. Guo and S. Wang are with The Chinese University of Hong Kong. E-mails: {qtguo, swang}@se.cuhk.edu.hk
- D. Lin is with Shenzhen Institute of Computing Sciences. E-mails: lindandan@sics.ac.cn
- R. Wong is with The Hong Kong University of Science and Technology. E-mail: raywong@cse.ust.hk
- W. Lin are with Tencent Inc. E-mails: {danieslin, edwlin}@tencent.com

Manuscript received XXXX XX, XXXX; revised XXXX XX, XXXX.

on their websites [19], [20]. By recommending pertinent groups to join, social network platforms can enhance user engagement, encourage community interactions, and subsequently increase the duration users spend on the platform. Using our GHP measure, a platform can pinpoint groups that are in close proximity to a user, based on the graph's connectivity information, and then make a recommendation. (2) *Collaborative Filtering*. Recommendation systems can estimate a user's structural proximity to groups of users with similar tastes. If a user is proximate to a group that has expressed a liking for a certain item, that item might be recommended to the user. (3) *Web Network Analysis*. By gauging the proximity of a web page (node) to groups of web pages with established categories, search engines can refine categorization and heighten the relevance of search results.

The first hitting constraint also makes the GHP more reasonable when considering the proximity between a node and a group, as it fundamentally views the group as a singular entity. This contrasts with other metrics like personalized PageRank (PPR) [6], [21], which only consider the probability that an α-terminating random walk stops at a specific node. To obtain the proximity of a group with respect to the source node, one might aggregate the PPR scores of nodes within that group. However, as the case study in Section 5.3 shows, the PPR-aggregation-based metric is still inferior to the GHP in proximity-based group recommendation. Thus, it motivates us to probe into the GHP-based query.

In this paper, we first study the *pairwise query*. This query takes as input a source node s, a target set T, and outputs the value $f(s, T)$. Such a pairwise query can be employed to measure the similarity between a node and a specified community or group. However, in certain scenarios, there may be a desire to pinpoint a limited number of group sets that are of utmost relevance to a particular node. Consider the application of group suggestions in social networks: instead of suggesting an extensive list of groups, social networking platforms might prefer to highlight a limited number of groups that possess the highest appeal to the user, thereby enhancing the recommendation success rate. Motivated by this, we delve into the second type of GHP-based query: the *top-k query*. This query takes a source node s as input and retrieves the k target sets with the top GHP values with respect to s.

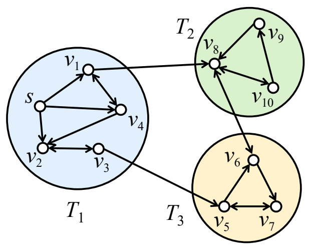
Deriving an exact solution for these two types of queries causes prohibitive computational costs. Specifically, the pairwise query requires $O(n^{2.37})$ time since it needs to compute the inverse of an $n \times n$ matrix. For top-k queries, it is even more expensive, which requires computing the matrix inversion much more times, depending on the number of groups. Thus, we aim to present efficient approximate algorithms with performance guarantees.

Contributions. Our contributions are presented as follows:

(1) We first develop an efficient algorithm called *SAMBA* to answer pairwise queries. Though SAMBA[1] is inspired by BiPPR [22], which is designed for the approximate pairwise PPR problem, as we will see in Section 3, it is far from a facile adaption.

(2) With SAMBA as the backbone, we further develop a framework called *PING* for the top-k GHP queries. PING runs in an iterative manner to gradually refine the lower and upper bounds for the estimated GHP values. The algorithm is designed to terminate as soon as the stopping condition is met, thus avoiding unnecessary computational costs. In addition, we present an optimization technique further to accelerate the computation of PING.

1. Random <u>Samp</u>ling with <u>Ba</u>ckward Local Push



(a) A toy graph

(1) $s \to v_1 \to v_4 \to v_1 \to v_8$
(2) $s \to v_1 \to v_8 \to v_{10} \to v_9$
(3) $s \to v_1 \to v_4 \to v_2$
(4) $s \to v_2 \to v_3 \to v_2$
(5) $s \to v_4 \to v_1$
(6) $s \to v_2 \to v_3 \to v_5 \to v_7 \to v_5$
(7) $s \to v_4 \to v_2 \to v_3 \to v_5 \to v_6 \to v_8$

(b) 7 random walks from node s

Fig. 1: (a) A toy graph G whose nodes are partitioned into 3 target sets T_1, T_2, and T_3. (b) 7 random walks starting from s.

(3) We run extensive experiments on real datasets to demonstrate the efficiency of our algorithms. Our experiments show that our algorithms SAMBA and PING take much less running time than their competitors on all datasets.

2 Preliminary

2.1 Problem Definition

Let $G = (V, E)$ be a directed graph with n nodes and m edges. For an undirected input graph, we convert it to a directed graph by treating each undirected edge as two directed edges in opposing directions [6], [8]. Given a source node $s \in V$ and a terminating probability α, an *α-terminating random walk* starts from s, and then, at each step, it chooses one of the following two options: either (i) terminates at the current node with α probability; or (ii), with $1 - \alpha$ probability, moves to an out-neighbor of the current node uniformly at random. From the definition, it immediately follows that the probability of such a random walk exceeding L steps is bounded by $(1-\alpha)^L$. Consequently, the occurrence of a random walk spanning a large number of steps becomes exceedingly unlikely, making it negligible.

Definition 1 (Hitting probability). *Given a graph G, let $(X_i)_{i=0}^{\tau}$ be an α-terminating random walk on G. Then for any two nodes s and t, we define the hitting probability as*

$$f(s,t) = \Pr\left[\{t\} \cap \{X_i\}_{i=0}^{\tau} \neq \emptyset | X_0 = s\right]$$

where $\{X_i\}_{i=0}^{\tau}$ is the node set visited by random walk $(X_i)_{i=0}^{\tau}$.

To explain, the hitting probability $f(s,t)$ is the probability that an α-terminating random walk starting from s could hit t before it terminates. Obviously when $s = t$, $f(s,t) = 1$ since a node s always hits itself at the very beginning. In the remainder of this paper, we omit the term *α-terminating* if the context is clear.

Example 1. *Consider the toy graph in Fig. 1(a). We simulate 7 random walks starting from s, shown in Fig. 1(b). Note that among them there are 4 (resp. 2) random walks that hit target node v_1 (resp. v_5). Thus, from the definition, the hitting probability $f(s, v_1)$) and $f(s, v_5)$ is estimated as $4/7$ and $2/7$, respectively.*

Definition 2 (Group hitting probability). *Given a graph G, let $(X_i)_{i=0}^{\tau}$ be an α-terminating random walk on G. Then for node s and set of nodes $T \neq \emptyset$, the group hitting probability is*

$$f(s,T) = \Pr\left[T \cap \{X_i\}_{i=0}^{\tau} \neq \emptyset | X_0 = s\right].$$

In this paper, we focus on the *group hitting probability* (GHP) $f(s,T)$. Formally, the GHP $f(s,T)$ is the probability that a random walk from the source node s hits any one of the nodes in T during the traversal. It is easy to see that the hitting probability $f(s,t)$ of a single target node t is a special case of the GHP $f(s,T)$ with $T=\{t\}$.

Example 2. *Consider the toy graph shown in Fig. 1(a) and the random walks in Fig. 1(b), where the nodes are partitioned into 3 target sets, namely $T_1=\{s,v_1,v_2,v_3,v_4\}$, $T_2=\{v_8,v_9,v_{10}\}$ and $T_3=\{v_5,v_6,v_7\}$. Firstly, we could know that $f(s,T_1)$ is always 1 since all the walks hit the target set T_1 at the very beginning. Note that the target set T_2 is hit by the walks with index 1, 2, and 7, thus yielding an estimated GHP $\hat{f}(s,T_2)=3/7$.*

To the best of our knowledge, we are the first to consider the approximate query for the group hitting probability with performance guarantee and aim to provide inspirational insights for future research. In this paper, we study the approximate pairwise GHP query and the approximate top-k GHP query, which are defined as follows.

Definition 3 (Approximate Pairwise GHP Query)**.** *Given a source node s, a target set T, a relative error ϵ, a threshold δ, and a failure probability p_f, the approximate pairwise GHP query returns the estimated GHP $\hat{f}(s,T)$ such that*

$$|f(s,T)-\hat{f}(s,T)|\le\epsilon\cdot f(s,T),\ \textit{if}\ f(s,T)\ge\delta, \qquad (1)$$

holds with at least $1-p_f$ probability.

This definition indicates that the estimated GHP $\hat{f}(s,T)$ should provide a relative error guarantee when $f(s,T)\ge\delta$. We do not consider a relative error guarantee when $f(s,T)<\delta$. Providing high accuracy for a target set T with $f(s,T)<\delta$ is unnecessary since such a small GHP indicates that T tends to be irrelevant to source s. Following previous work [6], [22], both δ and the fail probability p_f are set as $\frac{1}{n}$.

Definition 4 (Approximate Top-k GHP query)**.** *Given a graph $G(V,E)$, a set $\mathcal{T}$ of target sets such that $\mathcal{T}=\{T_1,T_2,...,T_{|\mathcal{T}|}\}$ where $|\mathcal{T}|$ is the number of target sets, a source node s, a positive integer k, a relative error ϵ, and a threshold δ, the approximate top-k GHP query returns a set $\mathcal{R}=\langle T'_1,...,T'_k\rangle$ such that for any $i\in\{1,...,k\}$ with $f(s,T^*_i)\ge\delta$:*

$$\hat{f}(s,T'_i)\ge(1-\epsilon)\cdot f(s,T'_i) \qquad (2)$$

$$f(s,T'_i)\ge(1-\epsilon)\cdot f(s,T^*_i) \qquad (3)$$

hold with at least $1-p_f$ probability, where T^*_i is the set whose exact GHP w.r.t. node s is the i-th largest.

Note that Equation (2) guarantees the accuracy of the estimated GHP values of returned top-k sets, while Equation (3) ensures that the returned i-th set T'_i should have an exact GHP *close* to the exact i-th largest GHP $f(s,T^*_i)$. It has been shown in previous work of PPR [6], [23], [24] that with these two conditions, the output of the approximate top-k query is nearly close to the true top-k result.

2.2 Existing Solutions

We first introduce the Monte Carlo (MC) approach for computing the GHP, and next, we present the existing local-update techniques (i.e., Forward Push and Backward Push) that are widely used for other random walk based queries [6], [22], [23].

MC method. To estimate the GHP value $f(s,T)$, we may simulate a sufficient number of random walks starting from s, and then estimate $f(s,T)$ as the fraction of random walks that hit at least one node in T, denoted by $\hat{f}(s,T)$. In order to obtain an estimation with a guarantee of at most ϵ relative error and probability of at least $1-1/n$, the number of random walks required, denoted as ω, is $O(\frac{\log n}{\delta\cdot\epsilon^2})$. This conclusion can be derived by concentration bounds (e.g., Chernoff bounds). When $\delta=O(1/n)$, the running time is $O(\frac{n\log n}{\epsilon^2})$. When it comes to the top-k GHP queries, for each target set $T\in\mathcal{T}$, the MC method computes an approximate GHP $\hat{f}(s,T)$ and then returns the top-k sets with the highest estimated GHP as the results. The time complexity is still $O(\frac{n\log n}{\epsilon^2})$. However, on large graphs, the MC method is not efficient enough. In particular, answering the top-k GHP query on the Orkut dataset, which comprises 3 million nodes, takes several hours to complete.

Forward Push [25]. Forward Push is usually used for accelerating the single source PPR query (which approximates the PPR values of each node $t\in V$ w.r.t. a source s) and top-k PPR query [6], [7], [24]. In particular, for each $t\in V$, it maintains a reserve $\pi^f(s,t)$ (i.e., the amount of PPR values that have been propagated to node t from s) and a residue $r^f(s,v)$ (i.e., the amount of PPR values that are "temporarily" held by node v and waiting for further propagation from node v to other nodes). Initially, Forward Push assigns unit PPR value to the residue of source node s, and then, continually updates the reserves and residues by performing forward push operations on each node v with non-zero residue (which distributes the residue of v evenly to its out-neighbours) [26]. When the algorithm finishes, the final reserve of each $t\in V$ could be considered as the approximate PPR value.

However, this technique does not work for the GHP. For ease of explanation, we assume the target set T includes only one node t. The Forward Push algorithm might perform forward push operations on node t multiple times during the propagation process, which puts computations of GHP in trouble. In particular, after performing a forward push operation on node t, the residue of node t is distributed to out-neighbours of node t, and then a few push iterations later, it might happen that a portion of the residue mass returns to t, resulting in double counting of the returning probability mass. Therefore, to represent this feature, we say the GHP is ***target-orient***. We still take Fig. 1(a) as an example. Consider node v_1 is the target node. The residue of v_1 comes from source s and v_4, and then, this residue will be distributed to its out-neighbours v_4 and v_8. Since the residue of v_4 is non-zero, it will be pushed to v_1 again and consequently, this probability mass is counted more than once. One might consider a simple solution to correct the above error: for a target node t, Forward Push does not distribute any residue from node t. From the definition of the GHP, it is easy to understand that such an adaption is correct. Therefore, we will take it as a competitor in our experimental evaluation and make a comparison with our proposed solution.

Backward Push [8]. Unlike Forward Push, Backward Push propagates the PPR value of a target node t via the reverse direction of each edge in the graph. The underlying rationale of Backward Push lies in a recursive definition where the PPR $\pi(s,t)$ equals a weighted sum of the PPR values $\pi(s,v)$, where v's are the in-neighbors of node t. The invariant is presented as follows: $\pi(s,t)=\alpha\cdot\mathbb{I}_s(t)+\sum_{v\in\mathcal{N}^-_T(t)}\frac{(1-\alpha)}{d^+(v)}\cdot\pi(s,v)$, where $\mathbb{I}_s(t)$ is an indicator function that is 1 if $t=s$ and otherwise 0. Backward push technique, however, cannot be directly applied to the GHP problem, as there exists no such a recursion for GHP. To explain, consider the simple case where the target set T contains only one target node $T = \{t\}$. Intuitively, the hitting probability $f(s,t)$ of target node t is related to the number of arrivals at t's in-neighbors (say v) in the traversal of a random walk, since for each arrival of in-neighbor node v, the walk could hit t in the next step with a certain probability. However, by the GHP definition, $f(s,v)$ is the probability that a random walk could hit node v before terminating, rather than how many times it hits v.

In the case where target set T contains more than one node, it is not correct to sum up $f(s,v_i)$ for all $v_i \in T$ to get the GHP $f(s,T)$. To explain, consider the second random walk in Fig. 1(b). This random walk first hits node $v_8 \in T_2$ and then node $v_{10} \in T_2$ and finally node $v_9 \in T_2$. If we simply sum up $f(s,v_8)$, $f(s,v_9)$ and $f(s,v_{10})$ to compute the GHP $f(s,T_2)$, the second random walk will be counted triple times, yielding a wrong answer. This is in contrast to the group setting in the PPR problem, which could be computed by summing up all the PPRs $\pi(s,v_i)$ for each node $v_i \in T$. It is because the PPR just takes into account the terminating node of a random walk, and thus when simulating a random walk, the events of stopping at node v_8 and stopping at node v_9 are disjoint.

2.3 Related Work

In graph analysis, measuring structural proximity between two graph objects based on graph topology is a fundamental problem that has been studied for decades. A significant category in this realm is the random-walk-based proximity measures. Personalized PageRank and SimRank are two notable representatives of this category. However, both focus on node-to-node proximity, which is distinct from the node-to-group case we explored in this paper.

Personalized PageRank (PPR) is first introduced by [27] to measure personalized views of importance between node pairs. The PPR score $\pi(s,t)$ between a source node s and a target node t represents the probability that an α-terminating random walk, starting from node s, concludes at node t. Given its efficacy in various graph mining applications, numerous studies aim to devise efficient algorithms for PPR computation. Andersen et al. [8] introduce the Forward Push algorithm for single-source PPR queries, though it lacks an accuracy guarantee. The Backward Push algorithm is presented in [25] to cater to single-target PPR queries. Lofgren et al. [22] combine the Backward Push approach with the Monte Carlo method, resulting in BiPPR, which addresses pairwise PPR queries. In [6], Wang et al. formulate an index-free algorithm named FORA, offering solutions for both single-source and top-k PPR queries with assured accuracy. Wu et al. [28] blend Power Iteration with Forward Push to create the SpeedPPR algorithm, tailored for single-source PPR queries. Nonetheless, all these methods do not apply to our GHP query.

The SimRank score $s(u,v)$ for nodes u and v, as proposed by Jeh et al. in [29], is recursively defined as:

$$s(u,v) = \begin{cases} 1, & \text{if } u = v \\ \frac{c}{|\mathcal{I}(u)|\cdot|\mathcal{I}(v)|} \sum\limits_{u' \in \mathcal{I}(u)} \sum\limits_{v' \in \mathcal{I}(v)} s(u',v'), & \text{otherwise} \end{cases}$$

where $\mathcal{I}(u)$ denotes the in-neighbors of node u, and c is a constant between 0 and 1. Tian et al. [30] interpret the SimRank concept $s(u,v)$ as the probability that two $(1-\sqrt{c})$-terminating random walks, starting from nodes u and v respectively, intersect with each other. They introduce an index-based algorithm, SLING, to address the single-source and top-k queries. In [31], Zhe et al. harness the reverse PageRank distribution of the input graph to formulate the index-based algorithm, PRSim, and demonstrate that it attains sub-linear time complexity for power-law graphs. ExactSim [32] offers high-precision single-source and top-k SimRank results for large graphs. Analogous to PPR, these methods cannot be applied to our GHP query.

Besides random-walk-based measures, other categories of proximity measures in the literature include common neighbors, shortest paths, and effective conductance. Let $N(u)$ represent the set of neighbors of node u. The Jaccard similarity, defined as $\frac{|N(u)\cap N(v)|}{|N(u)\cup N(v)|}$, proves useful in graph structural clustering [33]. The Katz measure [34] postulates that the more paths exist from node s to node t, and the shorter these paths, the greater the similarity of t to s. Koren et al. [35] employ cycle-free effective conductance to evaluate node proximity within a graph. These measures are distinct from our GHP problem.

One potential application of our GHP proximity measure is community or group recommendations. Most existing works on group recommendations utilize the Collaborative Filtering (CF) technique, which typically requires user history records or profiles. In [36], Chen et al. introduce the CCF model, considering both community-user co-occurrences and community-description co-occurrences, and derive the joint probability distribution over community, user, and description. With extensive profile data, Sharma [37] adopts a content-based approach and proposes a probabilistic latent preference model (Pairwise PLSI) for group recommendations on LinkedIn. Wang et al. [38] leverage production and consumption engagement records to determine the user-group affinity and introduce a time-dependent matrix factorization model for recommending groups to users. These methods stand in contrast to our application of group recommendation based on structural proximity. In fact, as highlighted in our experimental evaluation, our GHP solution exhibits high scalability. Resultantly, the research community can immediately reap benefits: utilizing our solution to pre-prune groups with minimal structural relevance to the user. This reduces the pool of candidate groups for other existing algorithms, thereby enhancing their scalability.

3 Pairwise GHP Query

As mentioned earlier, the target-oriented nature of GHP makes it challenging to formulate a correct local push algorithm. To tackle these issues, we first bring in a new concept *T-absorbed reachability* and then make a connection between GHP and the T-absorbed reachability. With the new tool, we derive a recursive definition for T-absorbed reachability from scratch and propose a group local push algorithm for T-absorbed reachability in Section 3.2, which ensures the correctness for computing $f(s,T)$. Finally, we show how to combine MC with group local push to answer pairwise queries in Section 3.3.

3.1 The T-Absorbed Reachability

Let $\Pr[s \rightsquigarrow v|L]$ be the probability that a random walk from s reaches v at the L-th hop. Given a target set T, we define the L hop *T-absorbed visiting probability* $\Pr[s \rightsquigarrow v|L,T]$ of node v w.r.t. node s as the probability that a random walk from s reaches v at the L-th hop (not necessary to stop at the L-th hop) but with a constraint that no node in T is visited unless a node in T is visited at the L-th hop (in this case $v \in T$). Now, the T-absorbed reachability $r_T(s,v)$ of node v w.r.t. s is defined as the sum of

Algorithm 1: Reach-MC $(G, s, T, v, \alpha, \omega)$

```
1 $c(s,v) \leftarrow 0$;
2 for $i = 1$ to $\omega$ do
3     Simulate a T-ARW starting from $s$;
4     $j \leftarrow$ the number of times that this walk visits $v$;
5     $c(s,v) \leftarrow c(s,v) + j$;
6 return $c(s,v)/\omega$;
```

the T-absorbed visiting probability of node v from node s at all possible number of hops, namely $L \in [0, +\infty)$, such that

$$r_T(s,v) = \sum\nolimits_{L=0}^{+\infty} \Pr[s \rightsquigarrow v|L,T].$$

For each node $v \notin T$, we can know that the value $r_T(s,v)$ is the expected number of times that a random walk visits v before reaching any node in the target set T. A special case is $v \in T$. In this case, v is the first node in T that is reached by this random walk (and then it terminates). Thus, if $v \in T$, $r_T(s,v) \leq 1$, which can be interpreted as the probability that node v is the first one among nodes in T visited by the random walk. Recall that the hitting time $h(s,t)$ is the expected number of steps for a random walk from source node s to hit node t for the first time, while the hitting probability $f(s,t)$ is the probability that the random walk from s can hit node t before it terminates.

Subsequently, we define a T-absorbed reachability $r_T(s,T)$ of a target set T w.r.t. s as the sum of $r_T(s,v)$ for each $v \in T$, i.e., $r_T(s,T) = \sum_{v \in T} r_T(s,v)$. From the definition of GHP, We know that the value $r_T(s,T)$ is exactly equal to the value $f(s,T)$ since $r_T(s,T)$ is the probability that a random walk from s hits any node in set T.

Theorem 1. *For a target set T, $f(s,T) = r_T(s,T)$.*

The theorem allows us to build a bridge between GHP and T-absorbed reachability. We can estimate $f(s,T)$ by computing $r_T(s,T)$. To estimate the T-absorbed reachability, we define the T-Absorbed Random Walk (T-ARW) that runs as follows: it starts from node s, and at each step: *(i)* it terminates *if the current node is a node t in T*; *(ii)* or terminates *with α probability*; *(iii)* otherwise, it moves to an out-neighbour of the current node *with $1-\alpha$ probability*. Algorithm 1, called *Reach-MC*, computes the T-absorbed reachability of a node v by exploiting the MC approach.

3.2 Group Local Push Algorithm

Next, we demonstrate how to carefully devise the group local push algorithm for T-absorbed reachability. Firstly, we show that there exists a recursive relationship between $r_T(s,u)$ and $r_T(s,v)$ for $v \in \mathcal{N}_T^-(u)$ where $\mathcal{N}_T^-(u)$ is the set of in-neighbors of u except the node in T. Specifically, for a T-ARW that reaches u at the L-th hop, it must first reach one in-neighbour v of u at the $(L-1)$-th step; otherwise, it cannot reach u at the following step. Therefore, for the L-hop T-absorbed visiting probability $\Pr[s \rightsquigarrow u|L,T]$, we can derive the following equation:

$$\Pr[s \rightsquigarrow u|L,T] = \sum\nolimits_{v \in \mathcal{N}_T^-(u)} \Pr[s \rightsquigarrow v|L-1,T] \cdot \frac{1-\alpha}{d^+(v)},$$

where $d^+(v)$ is the out-degree of node v. To explain, for a random walk that visits v at the $(L-1)$-th hop, with the remaining $1-\alpha$ probability, it randomly jumps to an out-neighbour of node v. Thus, it moves to node u from node v with $\frac{(1-\alpha)}{d^+(v)}$ probability.

Algorithm 2: GroupLocalPush$(G, s, T, \alpha, R_{max})$

```
1  $R(v,T) \leftarrow 0$ for $v \in V \backslash T$;
2  $R(t,T) \leftarrow 1$ for each $t \in T$;
3  $z(s,T) \leftarrow 0$;
4  for each $t \in T$ do
5      for each $v$ such that $v \in \mathcal{N}_T^-(t)$ do
6          $R(v,T) \leftarrow R(v,T) + \frac{1-\alpha}{d^+(v)} \cdot R(t,T)$;
7      $R(t,T) = 0$;
8  while $\exists v \in V \backslash T$ such that $R(v,T) > R_{max}$ do
9      if $v$ is $s$ then
10         $z(v,T) \leftarrow z(v,T) + R(v,T)$;
11     for each $u$ such that $u \in \mathcal{N}_T^-(v)$ do
12         $R(u,T) \leftarrow R(u,T) + \frac{1-\alpha}{d^+(u)} \cdot R(v,T)$;
13     $R(v,T) = 0$;
14 return $z(s,T)$, and $R(v,T)$ of each node $v \in V \backslash T$;
```

Adding all such in-neighbours v of node u together, the above equation is obtained. Now, by summing up over all possible hops L where $L \in [1, +\infty)$ since the value of L starts from 1 since this case must visit an in-neighbour of node u, we have that:

$$\sum_{L=1}^{+\infty} \Pr[s \rightsquigarrow u|L,T] = \sum_{v \in \mathcal{N}_T^-(u)} (\sum_{L=0}^{+\infty} \Pr[s \rightsquigarrow v|L,T]) \cdot \frac{1-\alpha}{d^+(v)},$$

which is equal to $\sum_{v \in \mathcal{N}_T^-(u)} r_T(s,v) \cdot \frac{1-\alpha}{d^+(v)}$ from the definition of $r_T(s,v)$. Since $r_T(s,u) = \sum_{L=0}^{+\infty} \Pr[s \rightsquigarrow u|L,T]$, we can derive that:

$$r_T(s,u) = \Pr[s \rightsquigarrow u|L=0,T] + \sum\nolimits_{v \in \mathcal{N}_T^-(u)} r_T(s,v) \cdot \frac{1-\alpha}{d^+(v)}.$$

Obviously, the value of $\Pr[s \rightsquigarrow u|L=0,T]$ is 1 if $s = u$ since node s always visits itself at the first step, and otherwise is 0. Thus, the final recursive equation is obtained as follows:

$$r_T(s,u) = \mathbb{I}_s(u) + \sum\nolimits_{v \in \mathcal{N}_T^-(u)} r_T(s,v) \cdot \frac{(1-\alpha)}{d^+(v)}, \quad (4)$$

where $\mathbb{I}_s(u)$ is an indicator function that is 1 if $u = s$ and otherwise 0. This recursive definition plays an important role in our local push algorithm. It is because it indicates that the T-absorbed reachability $r_T(s,u)$ could be obtained by summing up the T-absorbed reachability $r_T(s,v)$ of all in-neighbours v of u.

Rationale of group local push. Now, we present the rationale behind our group local push algorithm to compute the T-absorbed reachability $r_T(s,T)$, by exploiting Equation (4). Since the reachability $r_T(s,u)$ of any node $u \in V$ is obtained from its in-neighbours, the intuition of our group local push is to perform a reverse local push which starts from the target set T and continually traverses the incoming edges of each node that is not in T in a reverse manner. For each node u encountered and each in-neighbour v of u, the group local push algorithm calculates the probability that a T-absorbed random walk hitting v at a certain step would arrive at u at the next step. First of all, we consider a node $t \in T$. In the R.H.S of the above equation, the first term is a constant whose value is 1 or 0 depending on whether $t = s$. We call this constant a *reserve* of node t w.r.t. the source node s, denoted as $z(s,t)$, which is the amount of reachability value $r_T(s,t)$ that has been distributed to node s from t via the incoming edges in the graph. Besides, The second term reveals a probability that a T-ARW reaches node t from an in-neighbour v of node t, namely $\frac{(1-\alpha)}{d^+(v)}$. We call this probability a *residue* of node v w.r.t. node t, denoted as $R(v,t)$, which is waiting for further backward propagation via v's in-neighbours. We thus rewrite the reachability $r_T(s,t)$ as follows:

$$r_T(s,t) = z(s,t) + \sum\nolimits_{v\in\mathcal{N}_T^-(t)} r_T(s,v)\cdot R(v,t).$$

Similarly, for each in-neighbour $v \in \mathcal{N}_T^-(t)$, the reachability $r_T(s,v)$ could be represented as: $r_T(s,v) = z(s,v) + \sum_{w\in\mathcal{N}_T^-(v)} r_T(s,w)\cdot R(w,v)$. Thus, we could find a relationship between the reachability of t and the reachability of t's 2-hop in-neighbours such that:

$$r_T(s,t) = z(s,t) + \sum\nolimits_{v\in\mathcal{N}_T^-(t)} z(s,v)\cdot R(v,t)$$
$$+ \sum\nolimits_{v\in\mathcal{N}_T^-(t)} \sum\nolimits_{w\in\mathcal{N}_T^-(v)} r_T(s,w)\cdot R(w,v)\cdot R(v,t),$$

where the constant is updated as $z(s,t) + \sum_{v\in\mathcal{N}_T^-(t)} z(s,v)\cdot R(v,t)$ and the probability that a T-ARW from node w reaches node t via t's in-neighbour v is updated as $R(w,v)\cdot R(v,t)$. Thus, we can represent the reachability $r_T(s,t)$ as follows:

$$r_T(s,t) = z(s,t) + \sum\nolimits_{v\in\mathcal{N}_T^-(t)} \sum\nolimits_{w\in\mathcal{N}_T^-(v)} r_T(s,w)\cdot R(w,t),$$

by executing the following 3 steps for each in-neighbour v:

- increase the reserve $z(s,t)$ by $R(v,t)$ if $v = s$;
- update the residue $R(w,t)$ of each w as $\frac{(1-\alpha)}{d^+(w)}\cdot R(v,t)$;
- set the residue $R(v,t)$ as 0.

These 3 steps caused by each in-neighbour v of t are called *a push operation* on node v. Similarly, we could perform the same push operation at the 2-hop in-neighbour w of node t, the 3-hop in-neighbours of node t, and so on. Finally, the reachability $r_T(s,t)$ could be represented by the reachability value $r_T(s,v)$ for each node $v \in V$:

$$r_T(s,t) = z(s,t) + \sum\nolimits_{v\in V\setminus T} r_T(s,v)\cdot R(v,t) \qquad (5)$$

Secondly, we consider how to compute $r_T(s,T)$ by exploiting $r_T(s,t)$ of each node $t \in T$. As analyzed in Section 3.1, the reachability value $r_T(s,T)$ is the sum of $r_T(s,t)$ of each node $t \in T$. By integrating Equation 5, the reachability $r_T(s,T)$ of a target set T becomes:

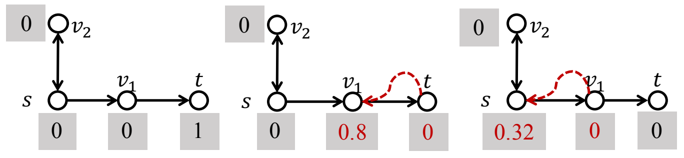
$$r_T(s,T) = \sum\nolimits_{t\in T} z(s,t) + \sum\nolimits_{v\in V\setminus T} r_T(s,v)\cdot \sum\nolimits_{t\in T} R(v,t)$$

Let $z(s,T)$ be the sum of $z(s,t)$ and let $R(s,T)$ be the sum of $R(v,t)$ for each node $t \in T$. We could derive the following invariant:

$$r_T(s,T) = z(s,T) + \sum\nolimits_{v\in V\setminus T} r_T(s,v)\cdot R(v,T). \qquad (6)$$

Hence, we could approximate $r_T(s,T)$ by $z(s,T)$ by performing a push operation at each node v with non-zero $R(v,T)$.

We develop our group local push algorithm, as shown in Alg. 2, where R_{max} is the user-specific threshold. Initially, it sets the residue $R(v,T)$ as 0 for each $v \in V\setminus T$ and the residue $R(t,T)$ as 1 for each $t \in T$ (Line 1). The reserve $z(s,T)$ is set as 0 (Line 2). Next, it performs the push operations on each $t \in T$ once (Lines 3-6) and never does the push operation again on the nodes in T since the nodes in T could be visited by a T-ARW only at the last hop. Then, it continues to perform the push operation on each node $v \in V\setminus T$ whose residue is larger than a threshold R_{max} (Lines 7-12). The algorithm finishes when no such node v can be found. Here, we set a threshold R_{max} to save the time cost following existing works [8]. If R_{max} is 0, then we derive the exact $r_T(s,T)$ value. To better understand our GroupLocalPush algorithm, we give a simple example where the target set T has only one node t.



(a) Initialization. (b) Backward push operation at t. (c) Backward push operation at v_1.

Fig. 2: Running example of the GroupLocalPush algorithm with target set $T = \{t\}$, where the residue value $R(v,T)$ for each v is shown inside grey box and the newly-updated ones are highlighted in red.

Example 3. *Consider the example in Fig. 2. Assume* $\alpha = 0.2$ *and the target set* $T = \{t\}$. *We set* $R_{max} = 0.4$. *Initially, the residue value of the target node t is set to* $R(t,T) = 1$, *while other nodes have zero residue values, as depicted in Figure 2(a). Next, a backward push operation is performed at node t as shown in Figure 2(b). This operation proceeds as follows: (i) for each in-neighbour v of node t, the residue of node v increases by* $\frac{1-\alpha}{d^+(v)}\cdot R(t,T)$ *(see Alg. 2 Line 12). Therefore, for the sole in-neighbour* v_1 *of t,* $R(v_1,T)$ *becomes* $\frac{1-0.2}{1} = 0.8$; *and (ii) it sets* $R(t,T) = 0$ *(see Alg. 2 Line 13). Subsequently, since* $R(v_1,T) = 0.8 > R_{max}$, *a backward push operation is performed at node* v_1, *as depicted in Figure 2(c). Here: (i) for the in-neighbour s of* v_1, $R(s,T)$ *increases by* $\frac{1-\alpha}{d^+(s)}\cdot R(v_1,T)$, *becoming* 0.32; *and (ii) it sets* $R(v_1,T)$ *to* 0. *Currently, as no node has a residue value exceeding* R_{max}, *the algorithm terminates. By Equation 6, we have* $r_T(s,T) = 0 + R(s,T)\times r_T(s,s) = 0.32\times r_T(s,s)$, *given that* $z(s,T) = 0$.

Finally, We show that the time complexity for the group local push algorithm is bounded by $O\left(\frac{|T|\cdot m}{n\cdot\alpha\cdot R_{max}}\right)$.

Theorem 2. *Given a target set T, the amortized cost for Algorithm 2 is bounded by* $O\left(\frac{|T|\cdot m}{n\cdot\alpha\cdot R_{max}}\right)$.

Proof. Our proof sketch consists of two steps. We firstly give the amortized time cost for the special case where the target set contains only a node t, which turns out to be $O\left(\frac{m}{n\cdot\alpha\cdot R_{max}}\right)$. The derivation is similar to that of the *Backward Push* [22]. Then we show that the amortized time cost for the general case with multiple nodes in T is $O(\frac{|T|\cdot m}{n\cdot\alpha\cdot R_{max}})$.

Consider the special case where $|T| = 1$. We claim that given a target node t, the time cost of all push operations conducted is bounded by $\sum_{v\in V\setminus\{t\}} \frac{r_t(u,t)}{R_{max}}\cdot O(d^-(u))$, where $d^-(u)$ is the in-degree of node u. This conclusion comes from the following two perspectives:

(i) The total number of push operations which is performed on any node $u \in V$ is at most $\lfloor\frac{r_t(u,t)}{R_{max}}\rfloor$ since each push operation on node u reduces its residue by at least R_{max} and the total residue of node u is at most $r_t(u,t)$.

(ii) Each push operation conducted on node u takes $O(d^-(u))$ time cost since the residue of each in-neighbour of node u is updated.

Now, we invoke Algorithm 2 n times with each node $u \in V$ as the target node. The total time is as follows:

$$\sum_{t\in V}\sum_{u\in V}\frac{r_t(u,t)}{R_{max}}O\big(d^-(u)\big)=\sum_{u\in V}\sum_{t\in V}\frac{r_t(u,t)}{R_{max}}O\big(d^-(u)\big).$$

On the other hand, due to the fact that $\sum_{t\in V} r_t(u,t) \le \frac{1}{\alpha}$, the total time cost is bounded by $\sum_{u\in V}\frac{O(d^-(u))}{\alpha\cdot R_{max}}$, which is bounded by $O(\frac{m}{\alpha\cdot R_{max}})$ since $\sum_{u\in V} O\big(d^-(u)\big) = O(m)$. Finally, by dividing the total time cost by n, the amortized running time for the single target node case is obtained.

Consider the case where T has multiple nodes. We directly obtain that the time cost of our group local push algorithm is bounded by $O(\frac{|T|\cdot m}{n\cdot\alpha\cdot R_{max}})$ since in the worst case, it has to individually perform push operations for each node $t \in T$. □

3.3 Combining Group Local Push with MC

The main idea of our SAMBA can be understood as a combination of the MC method and group local push: it first performs group local push with an elaborately designed threshold R_{max}, and then simulates a collection of T-ARWs. Since SAMBA exploits information obtained from the group local push, it can significantly cut down the number of required T-ARWs, saving computational overhead while still satisfying the desired quality guarantee.

Challenge. According to Section 3.2, after performing the phase of group local push, the invariant (i.e., Equation (6)) still holds. Thus, in order to compute $r_T(s,T)$, what we need is to estimate the GHP $r_T(s,v)$ for each v. Intuitively, we can sample a sufficiently large number of T-ARWs and get an unbiased estimation $\hat{r}_T(s,T)$, and then apply a concentration bound like Chernoff bound to bound the error of $\hat{r}_T(s,T)$. Unfortunately, all existing concentration bounds require that the range of the random variable (or its variance) is bounded. In our case, both the range and the variance of $r_T(s,v)$ are unbounded since the length of a T-ARW from s might be infinity. This makes it infeasible to guarantee the accuracy of the estimated $\hat{r}_T(s,T)$.

Hop-wise method. Recall that $r_T(s,v) = \sum_{L=0}^{+\infty}\Pr[s \rightsquigarrow v|L,T]$, where $\Pr[s \rightsquigarrow v|L,T]$ is bounded for each L. Thus we may approximate an $r_T(s,v)$ by separately estimating $\Pr[s \rightsquigarrow v|L,T]$ $(L = 1,2,\ldots)$, and then combining them together. From this idea, we develop our hop-wise method with approximation guarantee for $\sum_{v\in V\backslash T} r_T(s,v)\cdot R(v,T)$.

For ease of clarification, we use $H_L(s,v)$ to denote the L hop T-absorbed visiting probability $\Pr[s \rightsquigarrow v|L,T]$. In order to estimate $\sum_{v\in V\backslash T} r_T(s,v)\cdot R(v,T)$ by $H_L(s,v)$, we need to answer these two questions: (i) how to handle the infinite series of $H_L(s,v)$ $(L = 0,1,\ldots,\infty)$; (ii) how many L-hop random walks are required to estimate $H_L(s,v)$ respectively such that the subsequent aggregation of all $H_L(s,v)$ $(v \in V\backslash T)$ has accuracy guarantee to approximate $\sum_{v\in V\backslash T} r_T(s,v)\cdot R(v,T)$. We answer the first question with the truncation technique. Recall that an α-terminating random walk stops at each step with α probability. It can be directly derived that the probability that a random walk has L hops (or more) is $(1-\alpha)^L$. Therefore, for any $H_L(s,v)$, it is bounded by $(1-\alpha)^L$. That is, when L is sufficiently large, $H_L(s,v)$ is insignificant, and we can omit it. We give the following lemma about the truncation.

Lemma 1. *Let $L_{max} = \log_{1-\alpha}(\frac{\alpha}{1-\alpha}\cdot\frac{\epsilon\delta}{2})$. If $r_T(s,v) > \delta$, then the relative error of $\sum_{L=0}^{L_{max}} H_L(s,v)$ and $r_T(s,v)$ is at most $\frac{\epsilon}{2}$.*

Algorithm 3: SAMBA$(G,s,T,\alpha,\epsilon,p_f,\delta,R_{max})$

```
1  if $s \in T$ then
2      $\hat{f}(s,T) = 1$;
3  else
4      $[z(s,T), \mathbf{R}(T)] \leftarrow$ GroupLocalPush$(s,T,\alpha,R_{max})$;
5      $Y \leftarrow R(s,T)$, $L_{max} \leftarrow \log_{1-\alpha}(\frac{\alpha}{1-\alpha}\cdot\frac{\epsilon}{2}\cdot\delta)$ ;
6      $\omega \leftarrow \frac{3\cdot R_{max}\cdot\log(2/p_f)}{(1-R_{max}/2)\cdot\delta\cdot\epsilon^2}$;
7      for $L = 1$ to $L_{max}$ do
8          $Y_L \leftarrow 0$;
9          $\omega_L \leftarrow \lceil\omega\cdot(1-\alpha)^L\rceil$;
10         $a_L = \frac{(1-\alpha)^L\cdot\omega}{\omega_L}$;
11         for $i = 1$ to $\omega_L$ do
12             $W_i \leftarrow$ RandomWalk$(s,L)$;
13             if $W_i \cap T = \emptyset$ then
14                 $v \leftarrow$ the visited node at the $L$-th hop of $W_i$;
15                 $Y_L \leftarrow Y_L + R(v,T)\cdot a_L$;
16         $Y \leftarrow Y + Y_L$;
17     $\hat{f}(s,T) \leftarrow z(s,T) + R(s,T) + Y/\omega$;
18 return $\hat{f}(s,T)$;
```

Proof. Since $H_L(s,v)$ is bounded by $(1-\alpha)^L$, then we have

$$r_T(s,v)-\sum_{L=0}^{L_{max}}H_L(s,v)=\sum_{L=L_{max}+1}^{+\infty}H_L(s,v)$$
$$=\frac{(1-\alpha)^{L_{max}+1}}{1-(1-\alpha)}=\frac{1-\alpha}{\alpha}(\frac{\alpha}{1-\alpha}\cdot\frac{\epsilon}{2}\delta)\le\frac{\epsilon}{2}\cdot\delta$$
$$\le\frac{\epsilon}{2}\cdot r_T(s,v).$$

Thus, it completes the proof. □

Before answering the second question about the random walk number issues, we first present the pseudo-code of SAMBA, as shown in Algorithm 3. Specifically, for the first case (where $s \in T$), SAMBA sets $\hat{f}(s,T)$ as 1 (Lines 1-2). Otherwise, it starts the estimation by integrating the group local push with our random walk sampling (Lines 3-14): it firstly invokes our group local push algorithm with R_{max} which returns the reserve $z(s,T)$ and residue $R(v,T)$ for each node $v \in V\backslash T$ (Line 4). Next, it generates sufficient random walks (Lines 7-16), where the procedure **RandomWalk**(s,L) returns a random walk with L hops starting from s. Given a specific L, the number of L-hop random walks is $\omega_L = \lceil\omega\cdot(1-\alpha)^L\rceil$. Let $W_i[L]$ denote the node visited by the i-th random walk W_i at its L-th hop (Line 12). Let X_i be the random variable that takes value $R(v,T)\cdot a_L$ if $W_i[L] = v$ and $W_i \cap T = \emptyset$ (i.e., W_i does not visit any node in T), otherwise take value 0. Then the expectation of X_i is:

$$E[X_i]=\sum_{v\in V\backslash T}\frac{H_L(s,v)}{(1-\alpha)^L}\cdot R(v,T)\cdot a_L.$$

Then the expectation of Y_L is as follows:

$$E[Y_L]=\omega_L\cdot E[X_i]=\omega\cdot\sum_{v\in V\backslash T}H_L(s,v)\cdot R(v,T).$$

Therefore, we have:

$$E[Y/\omega]=E\left[\frac{1}{\omega}\cdot\sum_{L=1}^{L_{max}}Y_L\right]=\frac{1}{\omega}\cdot\sum_{L=1}^{L_{max}}E[Y_L]$$

$$= \sum_{v \in V \setminus T} \sum_{L=1}^{L_{max}} H_L(s,v) \cdot R(v,T). \quad (7)$$

As $H_0(s,s) = 1$ and $H_0(s,v) = 0$ for $v \neq s$, we have:

$$R(s,T) + E[Y/\omega] = \sum_{v \in V \setminus T} \sum_{L=0}^{L_{max}} H_L(s,v) \cdot R(v,T).$$

We claim that by setting $\omega = \frac{3 \cdot R_{max} \cdot \log(2/p_f)}{(1-R_{max}/2) \cdot \delta \cdot \epsilon^2}$, SAMBA returns a estimated GHP $\hat{f}(s,T)$ which is close to $f(s,T)$. Besides, to minimize the total time complexity, we can choose an R_{max} such that the cost of group local push and the MC are balanced.

Theorem 3. *SAMBA (Algorithm 3) guarantees that Equation (1) holds with at least* $1 - p_f$ *probability. By setting* $R_{max} = \epsilon \cdot \sqrt{\frac{\alpha \cdot |T| \cdot m \cdot \delta}{3n \cdot \log(2/p_f)}}$*, the time complexity of SAMBA is* $O(\frac{1}{\epsilon \cdot \alpha}\sqrt{\frac{|T| \cdot m \log(1/p_f)}{n \cdot \delta \cdot \alpha}})$.

Proof. To prove this theorem, we first analyze the correctness of SAMBA, and then its time complexity.

Correctness. Recall that the probability that a random walk has L hops (or more) is $(1-\alpha)^L$. Then we can have $\sum_{v \in V} H_L(s,v) = (1-\alpha)^L$. Therefore, due to the truncation on the random walks, the error is as follows:

$$\sum_{v \in V \setminus T} r_T(s,v) R(v,T) - \sum_{v \in V \setminus T} \left(\sum_{L=0}^{L_{max}} H_L(s,v) \right) R(v,T)$$
$$= \sum_{v \in V \setminus T} \left(\sum_{L=L_{max}+1}^{\infty} H_L(s,v) \right) R(v,T)$$
$$\leq \sum_{v \in V \setminus T} \sum_{L=L_{max}+1}^{\infty} H_L(s,v) R_{max}$$
$$\leq R_{max} \sum_{L=L_{max}+1}^{\infty} \sum_{v \in V \setminus T} H_L(s,v)$$
$$= R_{max} \sum_{L=L_{max}+1}^{\infty} (1-\alpha)^L$$
$$= R_{max} \cdot (1-\alpha)^{L_{max}+1} \cdot \frac{1}{\alpha} \leq R_{max} \cdot \frac{\epsilon}{2} \cdot \delta$$

Obviously, $R_{max} \cdot \frac{\epsilon}{2} \cdot \delta \leq \frac{\delta}{2}$, due to $R_{max} \leq 1$ and $\epsilon \leq 1$. Let $\delta' = \delta - \frac{R_{max}}{2} \cdot \delta = (1 - R_{max}/2)\delta$. If the number $\omega = \frac{3 \cdot R_{max} \cdot \log(2/p_f)}{\delta' \cdot \epsilon^2 \cdot \alpha}$ can give an estimation with (ϵ, δ')-approximation guarantee (with $1 - p_f$ probability), then we can confirm the correctness of SAMBA:

$$f(s,T) - \hat{f}(s,T)$$
$$= \sum_{v \in V \setminus T} r_T(s,v) R(v,T) - \sum_{v \in V \setminus T} \left(\sum_{L=0}^{L_{max}} \hat{H}_L(s,v) \right) R(v,T)$$
$$= \sum_{v \in V \setminus T} r_T(s,v) R(v,T) - \sum_{v \in V \setminus T} \left(\sum_{L=0}^{L_{max}} H_L(s,v) \right) R(v,T)$$
$$+ \sum_{v \in V \setminus T} \left(\sum_{L=0}^{L_{max}} H_L(s,v) \right) R(v,T)$$
$$- \sum_{v \in V \setminus T} \left(\sum_{L=0}^{L_{max}} \hat{H}_L(s,v) \right) R(v,T)$$
$$\leq R_{max} \cdot \frac{\epsilon}{2} \cdot \delta + \delta' \epsilon = \delta \cdot \epsilon.$$

Recall that there are $\omega' = \sum_{L=1}^{L_{max}} \omega_L$ random walks generated by SAMBA. We can re-index them as $W_j, j = 1, \ldots, \omega'$. Let Z_j be the random variable which takes value $R(v,T) \cdot a_L$ if the j-th random walk (assume it has L hops and v is its last node) visits no node in T, otherwise take value 0. Let $Z = \sum_{j=1}^{\omega'} Z_j$. For ease of explanation, let $\mu = \sum_{v \in V \setminus T} \sum_{L=1}^{L_{max}} H_L(s,v) R(v,T)$. According to Equation 7, we know

$$E[Z] = \omega \cdot \sum_{v \in V \setminus T} \sum_{L=1}^{L_{max}} H_L(s,v) R(v,T) = \omega \cdot \mu.$$

Let $X_j = Z_j / R_{max}$. Then we know $X_j \in [0,1]$ due to $a_L \leq 1$. Let $X = \sum_{j=1}^{\omega'} X_j$, and it is easy to get $E[X] = \frac{\omega \cdot \mu}{R_{max}}$ Then, we have:

$$\begin{aligned} &\Pr[|Z/\omega - \mu| \geq \epsilon\mu] \\ =&\Pr[|Z - E[Z]| \geq \epsilon E[Z]] \\ =&\Pr[|X - E[X]| \geq \epsilon E[X]] \\ \leq&2 \cdot \exp\left(-\frac{\epsilon^2 \cdot E[X]}{3}\right) \qquad //\text{Chernoff bound} \\ =&2 \cdot \exp\left(-\frac{\epsilon^2 \cdot \omega \cdot \mu}{3R_{max}}\right). \end{aligned} \quad (8)$$

Plugging $\omega = \frac{3 \cdot R_{max} \cdot \log(2/p_f)}{\delta' \cdot \epsilon^2}$ and $\mu \geq \delta'$ into the above expression,

$$\Pr[|Z/\omega - \mu| \geq \epsilon\mu] \leq 2\exp\left(-\frac{\epsilon^2 \cdot \omega \cdot \mu}{3R_{max}}\right) \leq p_f.$$

It completes the proof for the correctness.

Time complexity. The number of random walks generated by SAMBA is as follows:

$$\begin{aligned} \omega' =& \lceil (1-\alpha) \cdot \omega \rceil + \lceil (1-\alpha)^2 \cdot \omega \rceil + \ldots + \lceil (1-\alpha)^{L_{max}} \cdot \omega \rceil \\ \leq& (1-\alpha) \cdot \omega + (1-\alpha)^2 \cdot \omega + \ldots + (1-\alpha)^{L_{max}} \cdot \omega + L_{max} \\ \leq& \frac{1-\alpha}{\alpha}\omega + L_{max} \end{aligned}$$

As $L_{max} \ll \omega$ and α is typically small (say 0.1 and 0.2), we have $\omega' \approx \omega \cdot \frac{1-\alpha}{\alpha} \approx \omega/\alpha$.

Besides, since the number of L-hop random walks follows the geometric distribution (namely $\omega_L = (1-\alpha)^L \omega = \alpha(1-\alpha)^L \omega'$), for $L \leq L_{max}$. Then we can derive the expected length of the random walks generated by SAMBA as $1/\alpha$ by using the expectation of geometric distribution. Therefore, the total cost to generate random walks is $\omega'/\alpha = \omega/\alpha^2$. Following the strategy of BiPPR, we can balance the cost of group local push and generating random walks by setting up the following equality:

$$\frac{3 \cdot R_{max} \cdot \log(2/p_f)}{\delta \cdot \epsilon^2} \cdot \frac{1}{\alpha^2} = \frac{|T| \cdot m}{n \cdot \alpha \cdot R_{max}}$$

where we omit the term $(1 - R_{max}/2)$ for simplification (usually $R_{max} \ll 1$). Solving the above equation, we get

$$R_{max} = \epsilon \cdot \sqrt{\frac{\alpha \cdot |T| \cdot m \cdot \delta}{3n \cdot \log(2/p_f)}}.$$

Therefore, the time complexity of SAMBA is $O(\frac{1}{\epsilon \cdot \alpha}\sqrt{\frac{|T| \cdot m \log(1/p_f)}{n \cdot \delta \cdot \alpha}})$. □

4 Top-k GHP Query

In this section, we study the approximate top-k GHP problem (see Definition 4) and present an efficient algorithm PING[2] to answer the top-k queries with desirable approximation guarantee. Before that, we first derive upper and lower bounds, namely $f^u(s,T)$ and $f^l(s,T)$ for the GHP $f(s,T)$ if we invoke SAMBA. Note that these two bounds hold for any fixed δ and $R_{\max}$. The derivation is based on Chernoff bounds.

Lemma 2. *Given a source node s, a target set T and a failure probability p_f, by running SAMBA, we have that with $(1-p_f)$ probability, $f(s,T)$ is bounded:*

$$f(s,T) \leq f^u(s,T) \triangleq z(s,T) + R(s,T) + \Delta^u,$$
$$f(s,T) \geq f^l(s,T) \triangleq z(s,T) + R(s,T) + \Delta^l.$$

Let $\beta = \log(2/p_f)$. The notations Δ^u and Δ^l are defined as:

$$\Delta^u = \left(\sqrt{\frac{Y}{R_{max}} + \frac{\beta}{2}} + \sqrt{\frac{\beta}{2}}\right)^2 \cdot \frac{R_{max}}{\omega} + \frac{\epsilon}{n}$$
$$\Delta^l = \left(\left(\sqrt{\frac{Y}{R_{max}} + \frac{2}{9}\beta} - \sqrt{\frac{\beta}{2}}\right)^2 - \frac{\beta}{18}\right) \cdot \frac{R_{max}}{\omega} - \frac{\epsilon}{n}.$$

Proof. Recap that there are $\omega' = \sum_{L=1}^{L_{max}} \omega_L$ random walks generated by SAMBA. We can re-index them as $W_j, j = 1, \ldots, \omega'$. Let Y_j be the random variable which takes value $R(v,T) \cdot a_L$ if the j-th random walk (assume it has L hops and v is its last node) visits no node in T, otherwise take value 0. Let $Y = \sum_{j=1}^{\omega'} Y_j$. For ease of explanation, let $\mu = \sum_{v \in V \setminus T} \sum_{L=1}^{L_{max}} H_L(s,v) R(v,T)$. We know

$$E[Y] = \omega \cdot \sum_{v \in V \setminus T} \sum_{L=1}^{L_{max}} H_L(s,v) R(v,T) = \omega \cdot \mu.$$

Let $X_j = Y_j / R_{max}$. Then we know $X_j \in [0,1]$ due to $a_L \leq 1$. Let $X = \sum_{j=1}^{\omega'} X_j$, and it is easy to get $E[X] = \frac{\omega \cdot \mu}{R_{max}}$. In the following, we first derive an upper bound for $E[X]$ when we have the sample results of X. Let $\beta = \log(2/p_f)$.

$$\begin{aligned}
&\Pr\left[E[X] > \left(\sqrt{X + \frac{\beta}{2}} + \sqrt{\frac{\beta}{2}}\right)^2\right] \\
\leq\ &\Pr\left[\left(\sqrt{E[X]} - \sqrt{\frac{\beta}{2}}\right)^2 \geq X + \frac{\beta}{2}\right] \\
\leq\ &\Pr\left[X - E[X] \leq -\sqrt{2\beta \cdot E[X]}\right] \quad //\text{Chernoff Bound} \\
\leq\ &\exp\left(-\frac{2\beta \cdot E[X]}{2 \cdot E[X]}\right) = \frac{p_f}{2}.
\end{aligned}$$

On the other hand, as $X = Y \cdot R_{max}$ and $E[X] = \frac{\omega \cdot \mu}{R_{max}}$, by plugging it into the above inequality, we have the upper bound of u,

$$\mu \leq \left(\sqrt{\frac{Y}{R_{max}} + \frac{\beta}{2}} + \sqrt{\frac{\beta}{2}}\right)^2 \cdot \frac{R_{max}}{\omega}.$$

Further, by setting $\delta = \frac{2}{n}$ and thus $L_{max} = \log_{1-\alpha}\left(\frac{\alpha}{1-\alpha} \cdot \frac{\epsilon}{n}\right)$, according to Lemma 1, we derive the truncation error is at most $\frac{\epsilon}{n}$. Therefore, the upper bound can be obtained

$$\Delta^u = \left(\sqrt{\frac{Y}{R_{max}} + \frac{\beta}{2}} + \sqrt{\frac{\beta}{2}}\right)^2 \cdot \frac{R_{max}}{\omega} + \frac{\epsilon}{n}.$$

Similarly, we give a derivation for the lower bound:

$$\begin{aligned}
&\Pr\left[E[X] < \left(\sqrt{X + \frac{2}{9}\beta} - \sqrt{\frac{\beta}{2}}\right)^2 - \frac{\beta}{18}\right] \\
\leq\ &\Pr\left[\sqrt{E[X] + \frac{\beta}{18}} \leq \sqrt{X + \frac{2}{9}\beta} - \sqrt{\frac{\beta}{2}}\right] \\
\leq\ &\Pr\left[X - E[X] \geq \sqrt{2\beta \cdot E[X] + \frac{\beta^2}{9}} + \frac{\beta}{3}\right] \quad //\text{Chernoff Bound} \\
\leq\ &\exp(-\beta) = \frac{p_f}{2}.
\end{aligned}$$

Therefore, taking into account the truncation error, we derive the lower bound as follows:

$$\Delta^l = \left(\left(\sqrt{\frac{Y}{R_{max}} + \frac{2}{9}\beta} - \sqrt{\frac{\beta}{2}}\right)^2 - \frac{\beta}{18}\right) \cdot \frac{R_{max}}{\omega} - \frac{\epsilon}{n}.$$

□

4.1 PING Algorithm

In order to answer a top-k GHP query, one may consider invoking SAMBA for each target set $T \in \mathcal{T}$, and return the k target sets with top-k largest estimated GHPs. However, if we want to satisfy the accuracy requirement described in Definition 4, we have to set up the parameters according to the exact k-th largest GHP $f(s, T_k^*)$, which is unknown in advance. To tackle this problem, a naive solution is to conservatively set $\delta = \frac{1}{n}$. It yields huge unnecessary computational costs, as the value of $f(s, T_k^*)$ is usually much larger than $\frac{1}{n}$.

Motivated by the aforementioned deficiency, we propose an iterative algorithm, whose main idea is that we adaptively diminish the value of δ, obtain tighter and tighter bounds for each GHP in the candidate set, and then examine whether the terminating condition is met. If it is met, it indicates that the approximate top-k GHPs satisfying Definition 4 have been found, and we stop the algorithm immediately without wasting computational resources. If not, we halve δ to refine the lower and upper bounds and repeat the process until the approximate top-k results are found. Furthermore, in each iteration, we prune those target sets that are impossible to be the top-k results, such that in subsequent iterations we do not need to compute their GHP, further saving computational overhead.

Algorithm 4 shows the pseudo-code of PING. Initially, we initialize the candidate set $\mathcal{C}$ to include all the target sets. Then the algorithm goes into the iteration loops. In each iteration, it calculates the parameters R_{max} and ω according to the current value of δ. Then it invokes SAMBA to estimate the GHP for each $T \in \mathcal{C}$, and meanwhile derive its bounds $f^l(s,T)$ and $f^u(s,T)$ according to Lemma 2. Next, it constructs a set $\mathcal{R}$ of target sets whose upper bound $f^u(s,T)$ are top k largest among $\mathcal{C}$ (Line 8). If each target set T in $\mathcal{R}$ satisfies that the ratio $\frac{f^l(s,T)}{f^u(s,T)}$ is greater than $1-\epsilon$, the algorithm terminates. If this stopping condition is

2. Top-k GHP with Refining Bounds

Algorithm 4: PING$(G, \mathcal{T}, s, k, \epsilon)$

```
1  $\hat{f}(s,T) \leftarrow 0$ for each $T \in \mathcal{T}$;
2  $\mathcal{C} \leftarrow \mathcal{T}$, $p_f \leftarrow \frac{1}{n}$;
3  $p'_f \leftarrow \frac{p_f}{|\mathcal{T}| \cdot \log_2(n/k)}$;
4  for $\delta = \frac{1}{k}, \frac{1}{2k}, \frac{1}{4k}, \ldots, \frac{1}{n}$ do
5      $R_{max} \leftarrow \frac{\epsilon \cdot \sqrt{|\mathcal{C}| \cdot |T| \cdot \alpha \cdot m \cdot \delta}}{\sqrt{3n \cdot \log(2/p'_f)}}$;
6      $\omega \leftarrow \frac{3 \cdot R_{max} \cdot \log(2/p'_f)}{(1-R_{max}/2) \cdot \delta \cdot \epsilon^2}$;
7      Invoke SAMBA, and compute $f^l(s,T)$ and $f^u(s,T)$ for
         each $T \in \mathcal{C}$ according to Lemma 2;
8      $\mathcal{R} \leftarrow$ top-$k$ target sets from $C$ according to $f^u(s,T)$;
9      if $\frac{f^l(s,T)}{f^u(s,T)} \geq (1-\epsilon)$ for each $T \in \mathcal{R}$ then
10         return $\mathcal{R}$;
11     $f^l_k \leftarrow$ the $k$-largest $f^l(s,T)$ for $T \in \mathcal{C}$;
12     for each $T \in \mathcal{C}$ do
13         if $f^u(s,T) < f^l_k$ then
14             Prune $T$ from the candidate set $\mathcal{C}$;
15 Return $\mathcal{R}$;
```

not met, we have to halve the value of δ such that in the next iteration it can provide tighter bounds for each target set in $\mathcal{C}$. Before beginning with the next iteration, we further prune those target sets with upper bound $f^u(s,T)$ being smaller than the line f^l_k, which must not be the top-k answers, as there exist at least k target sets with larger GHP.

Next, we analyze the correctness of Algorithm 4, that is, it can provide approximate top-k results satisfying Definition 4.

Theorem 4. *Algorithm 4 returns the top-k result $\mathcal{R}$ which satisfies Definition 4 with at least $1 - 1/n$ probability.*

Proof. Let $\mathcal{R} = \{T'_1, T'_2, \ldots, T'_k\}$ where $f^u(s,T'_1) \geq f^u(s,T'_2) \geq \ldots \geq f^u(s,T'_k)$, namely, sorted in deceasing order of upper bounds. In the following, we show the correctness of Algorithm 4. When the algorithm terminates, each target set $T'_i \in \mathcal{R}$ (where $i = 1, 2, \ldots, k$) meets

$$f^l(s,T'_i) \geq (1-\epsilon) \cdot f^u(s,T'_i).$$

Then, as $\hat{f}(s,T'_i) \geq f^l(s,T'_i)$ (due to the definition of $f^l(s,T'_i)$) and $f(s,T'_i) \leq f^u(s,T'_i)$, we have for each $T'_i \in \mathcal{R}$

$$\hat{f}(s,T'_i) \geq f^l(s,T'_i) \geq (1-\epsilon) \cdot f^u(s,T'_i) \geq (1-\epsilon) \cdot f(s,T'_i).$$

Therefore, $\mathcal{R}$ conforms to the first condition of Definition 4.

Next, we show $\mathcal{R}$ satisfies the second condition as well. Since $f^u(s,T'_i)$ is the i-th largest upper bound, we conclude that $f(s,T^*_i) \leq f^u(s,T'_i)$; otherwise, there exists less than $i-1$ target sets with exact GHPs greater than $f(s,T^*_i)$. It contradicts with the fact that T^*_i is the i-th largest target set. On the other hand, Lemma 2 has guaranteed that $f(s,T'_i) \geq f^l(s,T'_i)$. Therefore, it yields that for $i = 1, 2, \ldots, k$,

$$f(s,T'_i) \geq f^l(s,T'_i) \geq (1-\epsilon) \cdot f^u(s,T'_i) \geq (1-\epsilon) \cdot f(s,T^*_i) \quad (9)$$

satisfying the second condition.

On the other hand, if the stopping condition (Lines 9-10) cannot hold, we have to keep diminishing the value of δ until $\delta = \frac{1}{n}$ in the worst case. According to Definition 4, we still obtain a qualified solution for the top-k GHP query.

In the above analysis, the upper and lower bounds hold for each target set in each iteration with $1 - p'_f$. Since there are at most $\log_2(n/k))$ iterations, and there are $|\mathcal{T}|$ target sets, by union bound, the total failure probability is $\log_2(n/k) \cdot |\mathcal{T}| \cdot p'_f = \frac{1}{n}$. □

Theorem 5. *The expected time complexity of PING (Algorithm 4) is $O(\frac{1}{\epsilon \cdot \alpha}\sqrt{\frac{m \cdot \log(1/p_f)}{\alpha \cdot \delta^*}})$, where δ^* is the value of δ when the algorithm terminates.*

Proof. Let δ_i denote the value of the variable of δ in the i-th iteration, namely $\delta_i = \frac{1}{2^{i-1} \cdot k}$. From the analysis procedure for the time complexity of SAMBA in Section 3, it is easy to realize that in the i-th iteration, the running cost is $O(C_0 \cdot \frac{1}{\sqrt{\delta_i}})$ where $C_0 = \frac{1}{\epsilon \cdot \alpha}\sqrt{\frac{|\mathcal{T}| \cdot |T| \cdot m \cdot \log(1/p_f)}{n \cdot \alpha}}$. Here we assume each node belongs to only one group set, namely $|\mathcal{T}| \cdot |T| \leq n$, then C_0 becomes $\frac{1}{\epsilon \cdot \alpha}\sqrt{\frac{m \cdot \log(1/p_f)}{\alpha}}$. Assume when $\delta = \delta^*$, Algorithm 4 meets the stopping condition. As $\delta_{i+1} = \frac{1}{2} \cdot \delta_i$, we know

$$\frac{1}{\sqrt{\delta_1}} + \frac{1}{\sqrt{\delta_2}} + \ldots + \frac{1}{\sqrt{\delta^*}} = O(\frac{1}{\sqrt{\delta^*}}).$$

It completes the proof. □

Lemma 3. *When $\delta = O(f(s,T^*_k))$, Algorithm 4 meets the stopping condition with high probability, where $f(s,T^*_k)$ is the exact k-th largest GHP.*

Proof. We first analyze the case of the pure Monte-Carlo method, namely, $R_{max} = 1$. When $\delta < \frac{1}{4} \cdot f(s,T^*_k)$, the stopping condition (Lines 9-10) in Algorithm 4 can terminate with high probability. Let $\mathcal{R}^*$ denote the set of group sets being the exact top-k GHP values.

Note that for any $T \in \mathcal{R}^*$, the number ω has guaranteed

$$(1-\epsilon/2) \cdot f(s,T) \leq \hat{f}(s,T) \leq (1+\epsilon/2) \cdot f(s,T).$$

Therefore, we obtain that with at least $1 - k \cdot p'_f$,

$$\frac{(1-\epsilon/2)f(s,T)}{(1+\epsilon/2)f(s,T)} \geq \frac{1-\epsilon/2}{1+\epsilon/2} \geq (1-\epsilon).$$

On the other hand, note that the function Δ^u is a non-decreasing function with respect to the variable Y when $Y \geq 0$, according to the expression of Δ^u. According to the construction of $\mathcal{R}$, for any $T' \in \mathcal{R}$ we have

$$f^u(s,T') \geq \min_{T \in \mathcal{R}^*} f^u(s,T).$$

In addition, the function Δ^l/Δ^u is also a non-decreasing function with respect to Y (by calculating its derivation). Therefore, for any $T' \in \mathcal{R}$, we have that

$$\frac{f^l(s,T')}{f^u(s,T')} \geq (1-\epsilon).$$

Now we turn to the case of SAMBA. As SAMBA provides the same approximate ratio as the MC method, it is considered to provide similar upper/lower bounds. Therefore, we know when $\delta = O(f(s,T^*_k))$, Algorithm 4 meets the stopping condition with high probability. □

Algorithm 5: Optimization $(G, s, \alpha, \omega, R_{max}, \mathcal{C})$

```
1  for each node $T \in \mathcal{C}$ do
2      $[z(s,T), \mathbf{R}(T)] \leftarrow$ Invoke Algorithm 2 with $G, s, T, \alpha, R_{max}$;
3  $\mathcal{H}(T) \leftarrow \emptyset$ for each $T \in \mathcal{C}$;
4  $i \leftarrow 0$;
5  $c(v) \leftarrow 0$ for each $v \in V$;
6  for $L = 1$ to $L_{max}$ do
7      $i \leftarrow i + 1$;
8      $\omega_L \leftarrow \lceil (1-\alpha)^L \omega \rceil$;
9      $a_L = (1-\alpha)^L \omega / \omega_L$;
10     for $j = 1$ to $\omega_L$ do
11         $W_i \leftarrow$ RandomWalk$(s, L)$;
12         $v \leftarrow$ the last node of $W_i$ ;
13         $c(v) \leftarrow c(v) + a_L$;
14         for each node $v$ in $W_i$ do
15             if $v$ is a member node of any candidate set $T$ then
16                 Append walk id $i$ to $\mathcal{H}(T)$;
17 for each node $T \in \mathcal{C}$ do
18     $\hat{f}(s,T) \leftarrow z(s,T) + R(s,T)$;
19     for each node $v$ whose residue $R(v,T) \neq 0$ do
20         $\hat{f}(s,T) \leftarrow \hat{f}(s,T) + c(v) \cdot R(v,T)/\omega$;
21     for each walk $i$ in $\mathcal{H}(T)$ do
22         $v \leftarrow$ the last node of $W_i$;
23         $f(s,T) \leftarrow \hat{f}(s,T) - a_L \cdot R(v,T)/\omega$;
24 Return $\hat{f}(s,T)$ for each node $T \in \mathcal{C}$;
```

4.2 Optimization

Recall that in the algorithm PING, it has to compute $\hat{f}(s,T)$ for each candidate set $T \in \mathcal{C}$ by invoking SAMBA, where $|\mathcal{C}|$ is the size of the candidate sets. For the sake of saving computational cost, we can generate only $\sum_{L=1}^{L_{max}} \omega_L$ random walks (Line 12), and reuse them to estimate the GHP values for each $T \in \mathcal{C}$. However, due to the target-orient property, we have to estimate the term $\sum_{v \in V \setminus T} r_T(s,v) R(v,T)$ (Lines 13-15) for each $T \in \mathcal{C}$ one by one. As a result, we need to scan these random walks $|\mathcal{C}|$ times in total. Such a solution is expensive and significantly hampers its practical performance.

Optimization. Our optimization is based on an observation that most candidate target sets are not frequently hit by random walks. Thus we can first assume that all random walks do not visit any node in any target set $T \in \mathcal{C}$. Then, we can scan random walks only once and estimate $r_T(s,v)$ for all $v \in V \setminus T$. Surely, the estimated $\hat{f}(s,T)$ under such an assumption is incorrect. Subsequently, a rectifying step is needed to correct the outcome.

Motivated by this idea, we propose our optimization technique whose pseudocode is given in Algorithm 5. Algorithm 5 consists of two phases: the *sampling* phase (Lines 6-16) and the *correction* phase (Lines 17 - 23). In particular, in the sampling phase, we sample $\sum_{L=1}^{L_{max}} \omega_L$ random walks. For each random walk W_i whose last node is v (Line 11), we directly increase the counter $c(v)$ by a_L since we have assumed W_i does not visit any target set. Then if any node of a target set T has been visited by W_i, we record the walk id in $\mathcal{H}(T)$ (Lines 14-16). Finding out all the target sets visited by W_i can be solved efficiently, if we maintain a data structure for all v indicating which target sets include v. In the second phase, for each $T \in \mathcal{C}$ it firstly estimates $\hat{f}(s,T)$ (Lines 18-20) by summing up $c(v) \cdot R(v,T)$ for each node v whose $R(v,T) \neq 0$. However, as the counter $c(v)$ has counted in those random walks that visit certain nodes in T, $\hat{f}(s,T)$ is over-estimated. Thus we go through the records in $\mathcal{H}(T)$, and remove the over-estimating amount from $\hat{f}(s,T)$ (Lines 21-23).

Example 4. *Consider the graph presented in Fig. 1. We set $L_{max} = 3$ and the candidate set $\mathcal{C}$ to consist solely of T_2. For simplicity, we omit the random walks of lengths $L = 1$ and $L = 2$. Suppose we sample $\omega_3 = 4$ random walks starting from the source node s. These walks are as follows: $\{W_1$: $s \to v_1 \to v_8 \to v_6$, W_2: $s \to v_4 \to v_2 \to v_3$, W_3: $s \to v_2 \to v_3 \to v_5$, W_4: $s \to v_4 \to v_2 \to v_3\}$. The terminal nodes of W_1, W_2, W_3, and W_4 are v_6, v_3, v_5, and v_3, respectively. This results in the counts $c(v_6) = a_3$, $c(v_3) = 2 \cdot a_3$, and $c(v_5) = a_3$ (as described in Algorithm 5, Line 9 and Line 13). As random walk W_1 visits node v_8 from T_2, we deduce $\mathcal{H}(T_2) = 1$, where the value 1 represents W_1's id. Next, we describe the calculation of $\hat{f}(s,T_2)$ using the sampled random walks, W_1 through W_4. Let $z(s,T_2) + R(s,T_2) = z_0$. We initially estimate $f(s,T_2)$ as $\hat{f}(s,T_2) = z_0 + (a_3 \cdot R(v_6,T_2) + 2 \cdot a_3 \cdot R(v_3,T_2) + a_3 \cdot R(v_5,T_2))/\omega$ (Line 20). However, this estimate is flawed as it incorrectly takes into account W_1, which has traversed the target set T_2 before terminating. To correct this, we adjust for the impact of W_1 (Line 23), resulting in $\hat{f}(s,T_2) = z_0 + (2 \cdot a_3 \cdot R(v_3,T_2) + a_3 \cdot R(v_5,T_2))/\omega$.*

5 Experiment

In this section, we experimentally evaluate our solutions SAMBA and PING against alternatives on real datasets. All experiments are conducted on a Linux server clocked at 2.3GHz, and all codes are implemented in C++ and compiled with -O3.

Dataset. We use eight real datasets that can be obtained from public sources SNAP [39] and Konect [40]. These datasets are frequently used in previous work of graph-based algorithms [6], [23]. Their statistics are reported in Table 1. Among them, the largest dataset Friendster has 1.8 billion edges and is used to examine the scalability of our algorithms.

Algorithms. For the pairwise GHP query, we compare our solution SAMBA with the MC method and Ad-FORA [6]. For the MC method, we simulate $O\left(\frac{\log n}{\delta \cdot \epsilon^2}\right)$ random walks such that it could provide ϵ-approximation guarantee. Ad-FORA is an adapted version of the state-of-the-art PPR method FORA with Forward Push. It is also designed to provide ϵ-approximation guarantee.

For the top-k GHP query, we evaluate our solution PING against the MC method, which generates sufficient random walks to provide an approximation guarantee presented in Definition 4. We also conduct an adapted version of PING which runs MC only without group local push (namely, R_{max} is always set as 1), denoted as *MC-PING*. In this way, we examine the effectiveness of the termination condition. We omit Ad-FORA in the top-k query as from the experimental results of pairwise queries, it is significantly worse than SAMBA.

We set the relative error parameter ϵ as 0.1 for both the pairwise query and for the top-k query. The values of both δ and p_f are set as $\frac{1}{n}$, following existing work [6], [7], [23]. The value of k ranges from 20 to 100.

Target sets. For the pairwise query, we vary the size of a target set T with $\{1, 10, 20, 50, 100\}$. We regard a target set as a

TABLE 1: Datasets. ($K = 10^3, M = 10^6, B = 10^9$)

Name	Abbr.	n	m	Type
DBLP	*DB*	317K	1.0M	undirected
YouTube	*YT*	1.1M	3.0M	undirected
Pokec	*PK*	1.6M	30.6M	directed
Wiki	*WK*	1.8M	28.5M	directed
Flickr	*FL*	2.3M	33.1M	directed
LiveJournal	*LJ*	4.0M	34.7M	undirected
Orkut	*OK*	3.1M	117.2M	undirected
Friendster	*FS*	65.6M	1.8B	undirected

certain community where nodes are usually intimate with each other. Under this assumption, we use a neighbour-based method to sample a target set. Specifically, we randomly sample a node as the seed of a target set and then add its neighbours to the target set. For the top-k GHP query, we set the size of a target set T as $|T| = 20$. The total number of all target sets is $|\mathcal{T}| = 10000$.

Accuracy Metrics. For the pairwise GHP query, obtaining the exact GHP value $f(s,T)$ requires prohibitive computational cost. By setting $\epsilon = 0.005$, we employ the SAMBA algorithm to obtain an estimated result denoted as $\tilde{f}(s,T)$. Our SAMBA guarantees that if $f(s,T) \geq \delta$, then

$$(1 - 0.005) \cdot f(s,T) \leq \tilde{f}(s,T) \leq (1 + 0.005) \cdot f(s,T).$$

Given this guarantee, we treat $\tilde{f}(s,T)$ as the exact GHP value of node s with respect to target set T. Suppose there's another estimated result $\hat{f}(s,T)$ for the GHP $f(s,T)$. We define the relative error of $\hat{f}(s,T)$ as

$$\text{Relative error} = \frac{|\hat{f}(s,T) - \tilde{f}(s,T)|}{\max\left(\tilde{f}(s,T), \delta\right)}.$$

For the top-k GHP query, we consider the results obtained by MC as the exact top-k results. To gauge the accuracy performance of each method, we use the precision metrics, defined as

$$\text{Precision} = \frac{|\mathcal{R} \cap \mathcal{R}'|}{k},$$

where $\mathcal{R}$ and $\mathcal{R}'$ are the exact top-k result and the approximate top-k result returned by other algorithms, respectively.

5.1 Pairwise GHP Query

In the first set of experiments, we evaluate the performance of our SAMBA as the target set size $|T|$ varies. As the target set size $|T|$ ranges from 1 to 100, we generate 50 random pairwise queries. We then employ SAMBA, along with baseline methods Ad-FORA and MC, to answer these queries. Subsequently, we report the average running time and the practical maximum relative error for each algorithm, to measure efficiency and accuracy performance.

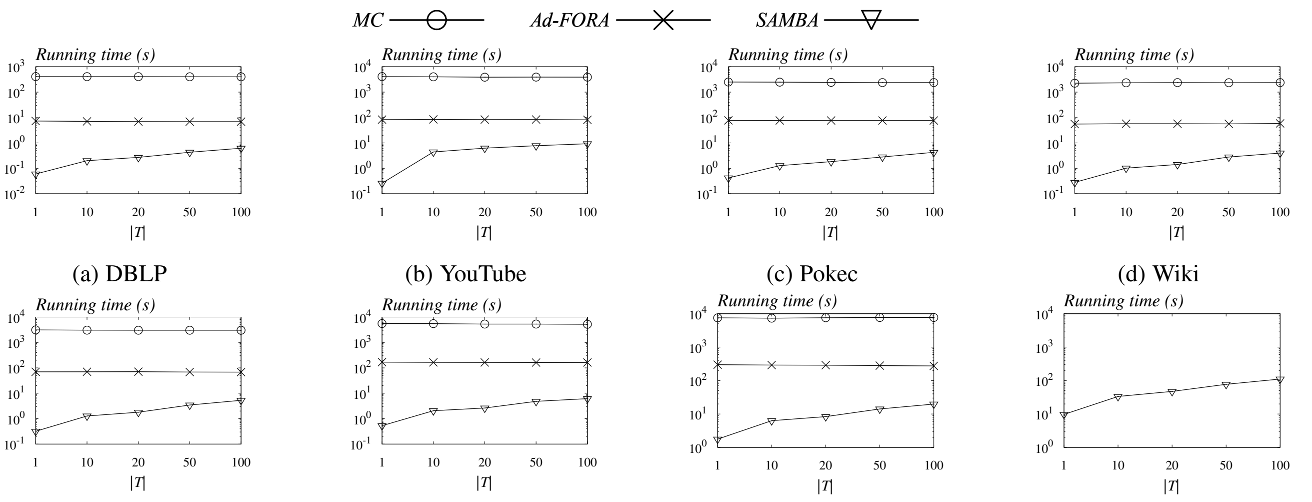
Figure 3 reports the average query time of each method for the pairwise query with varying target set size $|T|$. The results demonstrate that our algorithm SAMBA always runs the fastest, followed by Ad-FORA, with the MC method being the slowest. We omit the curves of Ad-FORA and MC on the largest dataset Friendster, since they take more than 4 hours and 72 hours, respectively, to complete a single pairwise query. Specifically, compared to the second fastest method Ad-FORA, our SAMBA is two orders of magnitude faster when $|T| = 1$ and one order of magnitude faster when $|T| = 100$, on all eight datasets. For example, on dataset Orkut, SAMBA is 170x (resp. 14x) faster than Ad-FORA when $T = 1$ (resp. $T = 100$). Furthermore, in comparison to MC, our SAMBA consistently takes advantage by at least two orders of magnitude across all eight datasets, even when $T = 100$. To illustrate, SAMBA is 400x (resp. 560x) faster than MC on dataset Orkut (resp. Pokec) with $T = 100$. The high efficiency of our SAMBA can be explained in that it integrates group local push algorithm to significantly reduce the number of random walks required to guarantee accuracy. We also observe from Figure 3 that the relationship between the running time of SAMBA (with fixed parameters δ, ϵ, and α) and $\sqrt{|T|}$ is roughly linear. That is, for all eight datasets, SAMBA takes a tenfold increase in running time when $|T|$ varies from 1 to 100. These experimental results conform with our analysis about the time complexity of SAMBA $O(\frac{\sqrt{|T|}}{\epsilon \cdot \alpha} \cdot \sqrt{\frac{m \log(1/p_f)}{n \cdot \delta \cdot \alpha}})$, which is presented in Theorem 3.

Then, as shown in Figure 4, we report the practical maximum relative errors of SAMBA, Ad-FORA, and MC to present their accuracy performance. From these figures, it's observed that the maximum relative errors of all algorithms across all datasets are less than $\epsilon = 0.1$. This confirms the algorithms' correctness, given their design to offer ϵ-approximation guarantees. Notably, our SAMBA algorithm exhibits smaller maximum relative errors than both Ad-FORA and MC on every dataset tested. This indicates that SAMBA outperforms Ad-FORA and MC in accuracy. This improved performance can be attributed to our utilization of the group local push technique, which significantly cuts down the number of required random walks. This reduction also aids in minimizing randomness during the estimation of the GHP value. We also note that in most cases MC suffers from higher maximum relative errors than Ad-FORA.

In the second set of experiments, we evaluate the impact of the relative error parameter ϵ on SAMBA's efficiency and accuracy, setting $|T| = 20$. We execute SAMBA to answer 50 random pairwise queries and subsequently report the average running time and the maximum relative error. The value of ϵ is set to $\{0.5, 0.1, 0.05, 0.01\}$. As depicted in Figure 5, the results indicate that as ϵ diminishes, the maximum relative error correspondingly decreases, but this improvement comes with a longer running time. A smaller maximum relative error signifies better accuracy performance. Notably, the maximum relative error consistently remains below the specified ϵ value across all eight datasets. For instance, in the LiveJournal dataset with ϵ values of 0.5, 0.1, 0.05, and 0.01, the maximum relative errors are 6.33×10^{-2}, 1.55×10^{-2}, 6.51×10^{-3}, and 1.45×10^{-3}, respectively. This observation aligns with SAMBA's design, which guarantees approximation performance. Moreover, the average running time appears to increase roughly in proportion to $\frac{1}{\epsilon}$. For example, SAMBA's running time on the LiveJournal dataset with ϵ values of 0.5, 0.1, 0.05, and 0.01 is 0.52, 2.63, 5.08, and 25.34 seconds, respectively. This is consistent with the derived time complexity of SAMBA, given as $O\left(\frac{1}{\epsilon \cdot \alpha}\sqrt{\frac{|T| \cdot m \log(1/p_f)}{n \cdot \delta \cdot \alpha}}\right)$.

5.2 Top-k GHP Query

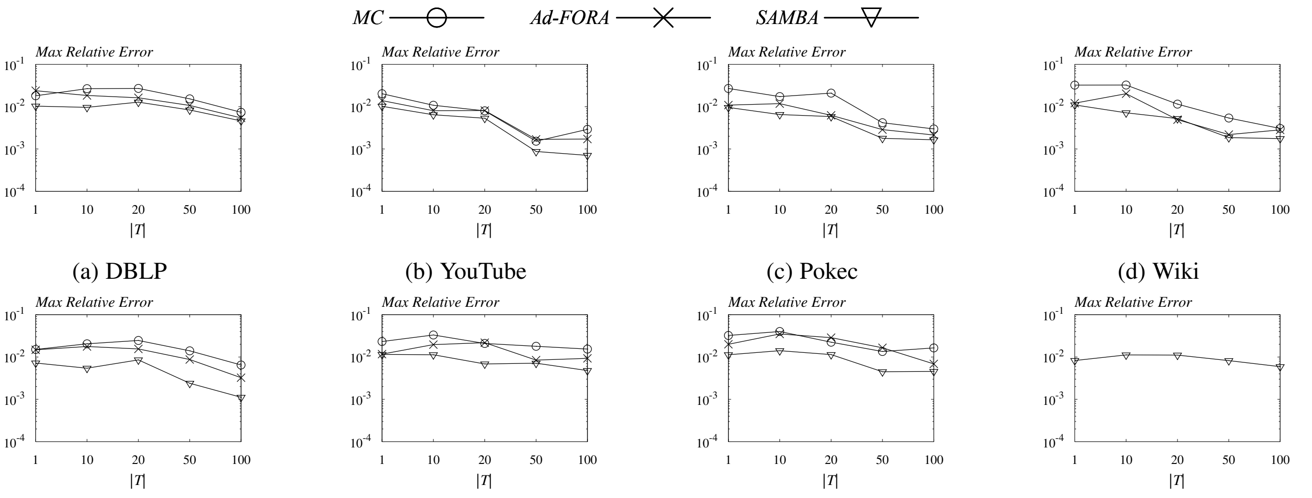
In the third set of experiments, we evaluate the effectiveness and accuracy of our PING/MC-PING in comparison to MC for answering top-k GHP queries. We set $|T| = 20$. By sampling 20 random sources, we execute PING, MC-PING, and MC to identify the target sets with the largest k GHP scores from the entire



(a) DBLP (b) YouTube (c) Pokec (d) Wiki

(e) Flickr (f) LiveJournal (g) Orkut (h) Friendster

Fig. 3: Varying $|T|$: Running time of each method for the pairwise GHP query.



(a) DBLP (b) YouTube (c) Pokec (d) Wiki

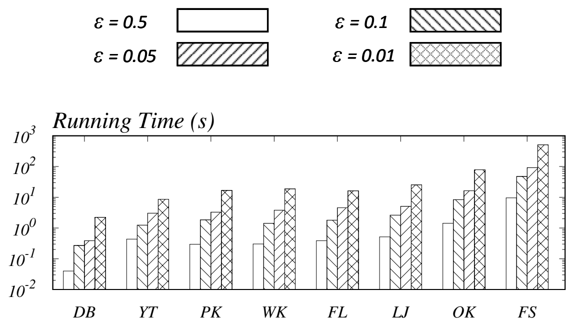
(e) Flickr (f) LiveJournal (g) Orkut (h) Friendster

Fig. 4: Varying $|T|$: Maximum relative error of each method for the pairwise GHP query.

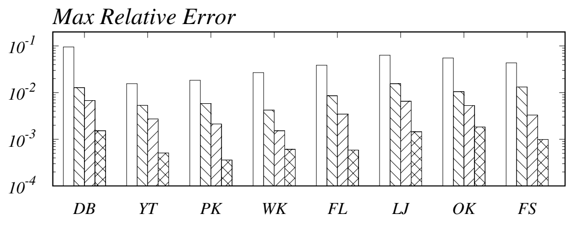
target set collection $\mathcal{T}$. Each algorithm answers each query five times. We then report the average running time and P@k value, assessing both efficiency and accuracy. We opt not to incorporate the terminating condition with the algorithm Ad-FORA, as the experimental results in Figure 3 already demonstrate its inferior performance compared to our SAMBA.

Figure 6 presents the average query time of MC, MC-PING, and PING for each dataset. The figures show that our PING and MC-PING algorithms consistently outpace the MC method significantly. Without prior knowledge of the exact top-k GHP scores, the MC method must conservatively provide approximation guarantees for all target sets with GHP scores greater than $\frac{1}{n}$. In contrast, our algorithms, PING and MC-PING, operate iteratively with a diminishing threshold δ, allowing them to terminate once they identify the top-k results that meet the user-specified approximation requirement. Consequently, our algorithms demonstrate greater efficiency than the MC method. Additionally, PING consistently outperforms MC-PING regarding query time across all tested datasets. For instance, with $k = 100$, PING processes a top-k query on the Orkut dataset in an average of 30 seconds, while MC-PING requires 250 seconds. This efficiency arises because PING integrates with our SAMBA and employs an optimization technique, enabling efficient estimation of the GHP value for each target set in $\mathcal{C}$. We omit the curve of running time of the MC method on the Friendster dataset, as it incurs prohibitive computational overhead.

Figure 7 presents the precision results of PING and MC-PING. Notably, MC-PING utilizes the same terminating condition as described in Lines 9-10 of Algorithm 4. The precision scores for the MC method consistently equal 1 since we regard its output as the ground truth. It's noteworthy that the precision scores of PING and MC-PING closely align, and in most instances, they exceed 99%. This underscores the effectiveness of our terminating condition. Additionally, our PING achieves an optimal balance between efficiency and accuracy. In comparison to the MC method, we realize a significantly reduced running time (by up to three orders of magnitude) while maintaining nearly identical query quality, enhancing its appeal for real-world applications. For the Friendster dataset, the single-threaded version of the MC method takes an excessive amount of time to compute the top-k target sets. To



(a) Running Time



(b) Maximum Relative Error

Fig. 5: Performance of SAMBA with varying parameter ϵ

address this, we implemented a multi-threaded version of MC. This allowed us to obtain the ground truth, facilitating accuracy evaluation for PING and MC-PING.

We next explore the performance of MC-PING/PING with respect to different values of relative error ϵ, set at $\{0.5, 0.1, 0.05, 0.01\}$. The value of k is set to 100. The experimental results for the DBLP dataset are depicted in Figure 8(a). They indicate that PING consistently achieves better runtime than MC-PING. As the relative error ϵ decreases, the performance difference between PING and MC-PING grows, with PING being more than 100x faster when $\epsilon = 0.01$. Although similar trends appear in other datasets, we omit these results due to space limitations. To further showcase the effectiveness of our optimization strategy, as detailed in Section 4.2, we present the runtime of PING for top-k queries, where $k = 100$, in Figure 8(b). This figure highlights that our optimization technique significantly reduces PING's execution time compared to the non-optimized version. Specifically, using our optimization leads to up to an 11x improvement in efficiency. This boost is attributed to our optimization design, which combines a single scan of the random walks with a subsequent refinement step for adjusting the GHP score estimates.

5.3 Case Study

To examine the performance of our GHP in measuring node-to-group proximity, we conducted a case study on proximity-based group recommendation. We compared our GHP method against three PPR-aggregation-based methods. Given that PPR is a widely-used random-walk-based tool for measuring node-to-node similarity [41], [42], [43], we select it as the baseline. We aggregate the PPR scores of the nodes within a group using max, sum, and average. These are denoted as PPR-max, PPR-sum, and PPR-avg, respectively. The real datasets DBLP and LiveJournal (abbreviated as LJ) available on the SNAP website [39] are associated with ground-truth communities, facilitating the evaluation of prediction performance.

TABLE 2: Community Prediction Performance (%).

Datasets		GHP	PPR-max	PPR-sum	PPR-avg
DBLP	NDCG@1	**78.1**	71.2	69.9	56.9
	NDCG@5	**86.7**	83.4	82.5	64.7
LJ	NDCG@1	**92.5**	83.4	90.8	81.4
	NDCG@5	**95.2**	90.5	94.2	88.2

We define $\mathcal{S}$ as the set of nodes associated with a specific community in the dataset. The experimental setup is as follows:

1) The set $\mathcal{S}$ is randomly split into two subsets: the training set $\mathcal{S}_1$ and the test set $\mathcal{S}_2$. The sizes of $\mathcal{S}_1$ and $\mathcal{S}_2$ are $70\% \times |\mathcal{S}|$ and $30\% \times |\mathcal{S}|$, respectively.
2) Nodes in $\mathcal{S}_2$ are then removed from their respective communities, aiming at hiding their community information.
3) From $\mathcal{S}_2$, we sample 1000 nodes. For each sampled node, its GHP scores and PPR aggregated scores with respect to communities are computed based on the modified community structure. The top-k results are then used to predict the community to which the node belongs.

To illustrate, consider a community $T = \{v_1, v_2, v_3\}$ in the dataset. After Step 1, we may have $v_1 \in \mathcal{S}_1, v_2 \in \mathcal{S}_1$, and $v_3 \in \mathcal{S}_2$. In Step 2, we remove node v_3 from T, thus hiding the fact that v_3 is a part of T. Consequently, T only contains the nodes v_1 and v_2. In Step 3, if node v_3 is among the sampled nodes, we compute its GHP and PPR aggregated scores regarding community sets. The top-k results are chosen as the predicted communities, with the ground truth for v_3 being community T.

Prediction performance is evaluated using the Normalized Discounted Cumulative Gain (NDCG), a prevalent metric in the literature for ranking quality [44], [45]. A higher NDCG value indicates better prediction performance. Table 2 presents the experimental results. The highest score is highlighted in bold, and the second highest is underlined. GHP significantly surpasses the PPR-aggregation-based methods. Notably, the NDCG@1 score of GHP leads by 6.9% (and 1.7%) over the second-best score in DBLP (and LJ). Furthermore, we observe that PPR-max performs best among the PPR-aggregation methods in DBLP, whereas PPR-sum leads in LJ, suggesting the sensitivity of PPR-aggregation methods to the input graph structure and its potential limitations in real-world scenarios.

6 Conclusion

In this paper, we introduce the concept of group hitting probability (GHP) to measure the node-to-group structural proximity on graphs. We investigate two basic types of GHP queries: the pairwise query and the top-k query. To answer the pairwise query, we present the SAMBA algorithm which combines the MC and group local push algorithm to achieve high efficiency while still providing performance guarantee. Further, to answer the top-k GHP query with approximation guarantee, we present the PING algorithm which could stop immediately as soon as the desired top-k results are found. We also propose an optimization technique to improve efficiency. Experimental results demonstrate that our solutions are more efficient than competitors.

References

[1] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," in *PVLDB*, 2009.

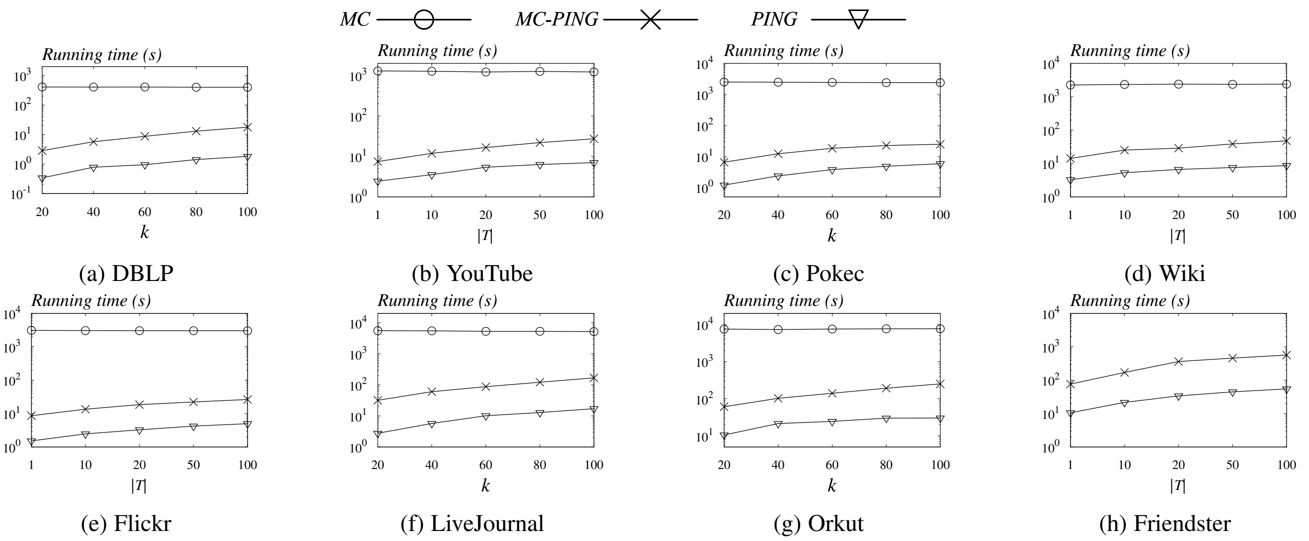


Fig. 6: Varying k: Running time of each method for the top-k GHP query where $|T| = 20$.

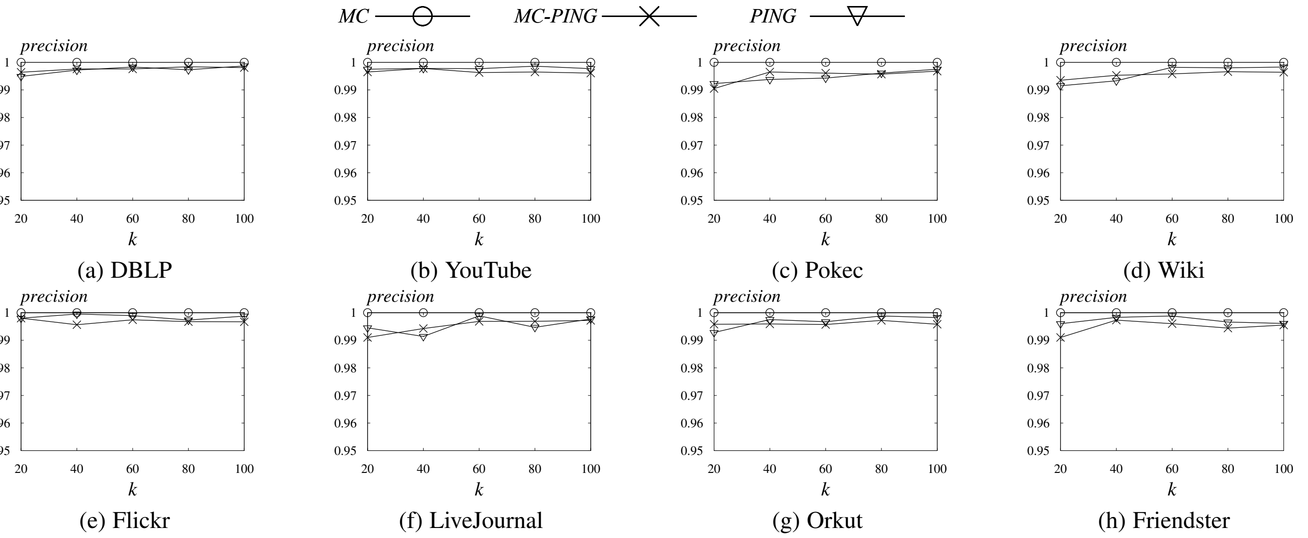


Fig. 7: Varying k: Accuracy performance of each method for the top-k GHP query where $|T| = 20$.

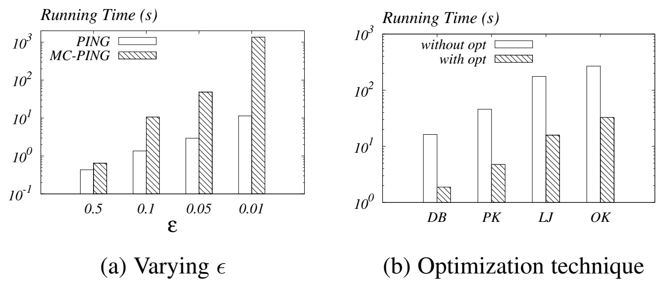


Fig. 8: (a) Time consumption of PING/MC-PING when varying relative error ϵ on dataset DBLP; (b) Time consumption of PING with/without optimization technique.

[2] R. Mihalcea and D. Radev, "Graph-based natural language processing and information retrieval," 2011.

[3] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *JAS-IST*, 2007.

[4] A. A. Benczúr, K. Csalogány, and T. Sarlós, "Link-based similarity search to fight web spam," in *AIRWEB*, 2006.

[5] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," in *Communications of the ACM*, 1992.

[6] S. Wang, R. Yang, X. Xiao, Z. Wei, and Y. Yang, "Fora: simple and effective approximate single-source personalized pagerank," in *SIGKDD*, 2017.

[7] Z. Wei, X. He, X. Xiao, S. Wang, S. Shang, and J.-R. Wen, "Topppr: top-k personalized pagerank queries with precision guarantees on large graphs," in *SIGMOD*, 2018.

[8] R. Andersen, F. Chung, and K. Lang, "Local graph partitioning using pagerank vectors," in *FOCS*, 2006.

[9] B. Bahmani, A. Chowdhury, and A. Goel, "Fast incremental and personalized pagerank," in *PVLDB*, 2010.

[10] P. Berkhin, "Bookmark-coloring algorithm for personalized pagerank computing," in *Internet Mathematics*, 2006.

[11] Y. Fujiwara, M. Nakatsuji, T. Yamamuro, H. Shiokawa, and M. Onizuka, "Efficient personalized pagerank with accuracy assurance," in *SIGKDD*, 2012.

[12] L. Lovász *et al.*, "Random walks on graphs: A survey," in *Combinatorics, Paul erdos is eighty*, 1993.

[13] Z. Guan, J. Wu, Q. Zhang, A. Singh, and X. Yan, "Assessing and ranking structural correlations in graphs," in *SIGMOD*, 2011.

[14] Q. Mei, D. Zhou, and K. Church, "Query suggestion using hitting time," in *CIKM*, 2008.

[15] M. Brand, "A random walks perspective on maximizing satisfaction and profit," in *SDM*, 2005.

[16] B. Liu, "Better than pagerank: Hitting time as a reputation mechanism," *Doctoral Dissertation in University of Harvard*, 2014.

[17] P. Sarkar and A. W. Moore, "Fast nearest-neighbor search in disk-resident graphs," in *SIGKDD*, 2010.

[18] L. Grady, "Random walks for image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 11, pp. 1768–1783, 2006.

[19] Meta, "Find facebook groups to join," https://www.facebook.com/help/197647530805955, 2023.

[20] Linkedin, "Find and join a linkedin group," https://www.linkedin.com/help/linkedin/answer/a544795, 2021.

[21] P. Lofgren, S. Banerjee, A. Goel, and C. Seshadhri, "Fast-ppr: scaling personalized pagerank estimation for large graphs," in *SIGKDD*, 2014.

[22] P. Lofgren, S. Banerjee, and A. Goel, "Personalized pagerank estimation and search: A bidirectional approach," in *SDM*, 2016.

[23] S. Wang, Y. Tang, X. Xiao, Y. Yang, and Z. Li, "Hubppr: effective indexing for approximate personalized pagerank," in *PVLDB*, 2016.

[24] S. Wang, R. Yang, R. Wang, X. Xiao, Z. Wei, W. Lin, Y. Yang, and N. Tang, "Efficient algorithms for approximate single-source personalized pagerank queries," in *TODS*, 2019.

[25] R. Andersen, C. Borgs, J. Chayes, J. Hopcraft, V. S. Mirrokni, and S.-H. Teng, "Local computation of pagerank contributions," in *WAW*, 2007.

[26] D. Lin, R. C.-W. Wong, M. Xie, and V. J. Wei, "Index-free approach with theoretical guarantee for efficient random walk with restart query," in *ICDE*, 2020.

[27] G. Jeh and J. Widom, "Scaling personalized web search," in *WWW*, 2003, pp. 271–279.

[28] H. Wu, J. Gan, Z. Wei, and R. Zhang, "Unifying the global and local approaches: An efficient power iteration with forward push," in *SIGMOD*, 2021, pp. 1996–2008.

[29] G. Jeh and J. Widom, "Simrank: a measure of structural-context similarity," in *SIGKDD*, 2002.

[30] B. Tian and X. Xiao, "Sling: A near-optimal index structure for simrank," in *SIGMOD*, 2016.

[31] Z. Wei, X. He, X. Xiao, S. Wang, Y. Liu, X. Du, and J.-R. Wen, "Prsim: Sublinear time simrank computation on large power-law graphs," in *SIGMOD*, 2019.

[32] H. Wang, Z. Wei, Y. Yuan, X. Du, and J. Wen, "Exact single-source simrank computation on large graphs," in *SIGMOD*. ACM, 2020, pp. 653–663.

[33] F. Zhang and S. Wang, "Effective indexing for dynamic structural graph clustering," *Proc. VLDB Endow.*, vol. 15, no. 11, pp. 2908–2920, 2022.

[34] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, 1953.

[35] Y. Koren, S. C. North, and C. Volinsky, "Measuring and extracting proximity in networks," in *SIGKDD*, 2006, pp. 245–255.

[36] W. Chen, D. Zhang, and E. Y. Chang, "Combinational collaborative filtering for personalized community recommendation," in *SIGKDD*, 2008, pp. 115–123.

[37] A. Sharma and B. Yan, "Pairwise learning in recommendation: experiments with community recommendation on linkedin," in *RecSys*, 2013, pp. 193–200.

[38] X. Wang, R. Donaldson, C. Nell, P. Gorniak, M. Ester, and J. Bu, "Recommending groups to users using user-group engagement and time-dependent matrix factorization," in *AAAI*, 2016, pp. 1331–1337.

[39] "SNAP Datasets," http://snap.stanford.edu/data, 2014.

[40] "KONECT," http://konect.cc/networks/, 2013.

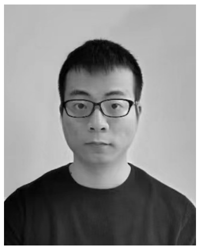
[41] X. Zhang, K. Xie, S. Wang, and Z. Huang, "Learning based proximity matrix factorization for node embedding," in *KDD*, 2021, pp. 2243–2253.

[42] R. Yang, J. Shi, X. Xiao, Y. Yang, and S. S. Bhowmick, "Homogeneous network embedding for massive graphs via reweighted personalized pagerank," *Proc. VLDB Endow.*, vol. 13, no. 5, pp. 670–683, 2020.

[43] L. Backstrom and J. Leskovec, "Supervised random walks: predicting and recommending links in social networks," in *WSDM*, 2011, pp. 635–644.

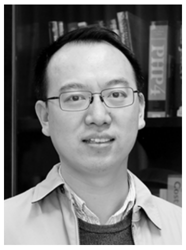
[44] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, "Neural collaborative filtering," in *WWW*, 2017, pp. 173–182.

[45] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li, "DRN: A deep reinforcement learning framework for news recommendation," in *WWW*, 2018, pp. 167–176.



Qintian Guo is currently a Ph.D. candidate in the Department of Systems Engineering and Engineering Management, the Chinese University of Hong Kong. He received his B.E. degree from the University of Science and Technology of China and his M.Sc. degree from the University of Alberta. His research interests include graph data management and graph mining.

Dandan Lin received the Ph.D. degree from The Hong Kong University of Science and Technology in 2021. Currently, she is a research fellow at Shenzhen Institute of Computing Sciences. Her research interests include large-scale graph analysis, graph neural networks, and graph mining.



Sibo Wang is an Assistant Professor in the Department of Systems Engineering and Engineering Management, the Chinese University of Hong Kong. He received his B.E. in Software Engineering in 2011 from Fudan University and his Ph.D. in Computer Science in 2016 from Nanyang Technological University. He is currently interested in graph data management, big data analysis.

Raymond Chi-Wing Wong received the BSc, MPhil, and PhD degrees in computer science and engineering from the Chinese University of Hong Kong (CUHK), Hong Kong, in 2002, 2004, and 2008, respectively. He is a professor with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include database and data mining.

Wenqing Lin received the PhD degree in computer science from Nanyang Technological University, in 2015. Currently, he is a principal researcher with Tencent in Shenzhen, China. His research interests include graph databases and data mining.