SimTab: Accuracy-Guaranteed SimRank Queries through Tighter Confidence Bounds and Multi-Armed Bandits

Yu Liu[1], Lei Zou[12], Qian Ge[1], Zhewei Wei[3]
[1]Wangxuan Institute of Computer Technology, Peking University, China
[2]National Engineering Laboratory for Big Data Analysis Technology and Application (PKU), China
[3]Gaoling School of Artificial Intelligence, Renmin University of China
[1]{dokiliu, zoulei, geqian}@pku.edu.cn [2]{zhewei}@ruc.edu.cn

ABSTRACT

SimRank is a classic measure of vertex-pair similarity according to the structure of graphs. Top-k and thresholding SimRank queries are two important types of similarity search with numerous applications in web mining, social network analysis, spam detection, etc. However, extensive studies for SimRank most focus on single-pair and single-source queries and fail to provide any feasible solution for the top-k and thresholding queries, e.g., with theoretical accuracy guarantee or acceptable empirical performance. In this paper, we propose *SimTab* (SimRank queries with Tighter confidence bounds and multi-armed bandits) to answer top-k and thresholding queries in a unified manner. First, we integrate several techniques with random walk sampling to tighten the confidence bound of SimRank estimation, which enhances the query efficiency. Second, we answer top-k and thresholding queries from the perspective of the Multi-Armed Bandits (MAB) problems. The proposed algorithms significantly improve the theoretical efficiency over state of the art, whereas the algorithmic complexity closely matches the hardness of the problem. We further propose a novel sampling strategy specially tailored for node similarity queries, which improves both the theoretical and practical query efficiency of the MAB-based algorithms. Our method is the first with query accuracy guarantee for these two queries, and the sole algorithm to achieve high-quality query results on large graphs. Moreover, all proposed algorithms are index-free, and thus can be naturally applied to dynamic graphs.

Extensive experiments on several large graph datasets demonstrate that our algorithms achieve much superior effectiveness with comparable or less query time cost than all index-free and index-based state of the art. Besides, our work proposes the first thorough empirical evaluation of the existing SimRank algorithms over top-k and thresholding queries.

PVLDB Reference Format:
Yu Liu, Lei Zou, Qian Ge, and Zhewei Wei. SimTab: Accuracy-Guaranteed SimRank Queries through Tighter Confidence Bounds and Multi-Armed Bandits. *PVLDB*, 13(11): 2202-2214, 2020.
DOI: https://doi.org/10.14778/3407790.3407819

1. INTRODUCTION

Proceedings of the VLDB Endowment, Vol. 13, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3407790.3407819

SimRank [16] is a widely adopted measure of the similarities of graph nodes, with numerous applications such as web mining [18], social network analysis [23], and spam detection [29]. The formulation of SimRank is based on a recursive concept, i.e., two objects are similar if they are linked to similar objects, while an object is most similar to itself. Given a graph $G = (V, E)$, the SimRank similarity between two nodes u and v is defined as:

$$s(u,v) = \begin{cases} 1, & \text{if } u = v \\ \dfrac{c}{|I(u)| \cdot |I(v)|} \displaystyle\sum_{x \in I(u)} \sum_{y \in I(v)} s(x,y), & \text{otherwise} \end{cases} \tag{1}$$

where $I(u)$ denotes the set of in-neighbors of u, and $c \in (0,1)$ is a decay factor typically set to 0.6 or 0.8 [16,25].

A plethora of techniques has been proposed for the efficient computation of SimRank. Since computing all-pair SimRank incurs excessive time and space cost for large-sized graphs, most existing work focuses on the single-pair and single-source queries[1]. The single-pair query answers the SimRank similarity of a given node pair (u, v), whereas the single-source query takes a query node u as input and returns the similarity of each node w.r.t. u. Due to the recursive definition of SimRank, its exact values can not be computed in limited time. Given an error parameter ε and a failure probability δ, we say the query result achieves *absolute error guarantee* [24,30,32], if with at least $1-\delta$ probability, each returned estimated SimRank value $\hat{s}(u,v)$ satisfies that $|\hat{s}(u,v) - s(u,v)| \leq \varepsilon$.

Motivated by the real-world application scenarios [18,23,29], in this paper, we study the following top-k and thresholding queries.

DEFINITION 1 (TOP-k QUERY). *We are given a node u in G, a positive integer $k < n$, and a failure probability δ. Let v_i be the node in G whose SimRank similarity to u (denoted by $s(u, v_i)$) is the i-th largest, $i \in [1, k]$. A top-k SimRank query returns a set of k nodes $V'_k = \{v'_1, \ldots, v'_k\}$, such that with at least $1-\delta$ probability, for any $k \in [1, n)$, it satisfies that $s(u, v'_i) \geq s(u, v_k) - \varepsilon_{min}$ for all $i \in [1, k]$, where ε_{min} is a very small error tolerance parameter.*

DEFINITION 2 (THRESHOLDING QUERY). *Given a node u in G, a real number $\tau \in [0, 1]$, and a failure probability δ, denote by V_τ the set of nodes with SimRank similarity no smaller than τ w.r.t. u, i.e., $V_\tau = \{v | s(u, v) \geq \tau, v \in V\}$. A thresholding SimRank query returns a set of nodes V'_τ, such that with $1-\delta$ probability, for any v with $s(u, v) \geq \tau + \varepsilon_{min}$, v is included in V'_τ; and for any v with $s(u, v) < \tau - \varepsilon_{min}$, v is excluded from V'_τ. Here, ε_{min} is a very small error tolerance parameter.*

[1]Throughout this paper, we use the term single-pair and single-source query to denote the computation of SimRank with up to an additive error, due to the recursive nature of SimRank definition. They are also referred to as the *approximate* single-pair and single-source queries in [24,30,32].

In this paper, we set $\varepsilon_{min} = 10^{-6}$ for the following reasons. First, even the *exact* algorithm (e.g., the *Power Method*) needs ε_{min} to guarantee its convergence, and computes an approximation $\hat{s}(u, v)$ such that $|\hat{s}(u, v) - s(u, v)| \leq \varepsilon_{min}$. Second, our setting of ε_{min} is orders of magnitude smaller than the state-of-the-art single-source algorithms [24, 30, 32], which are the baselines that achieve the best query performance for top-k and thresholding SimRank queries. Finally, as shown later, the MAB algorithms [12, 19, 26] also need such an error parameter, to the best of our knowledge.

Table 1: A toy example with query node u_1 and u_2, where $s(*, *)$ (resp. $\hat{s}(*, *)$) denotes the exact (resp. estimated) SimRank values ($\varepsilon = 0.05$)

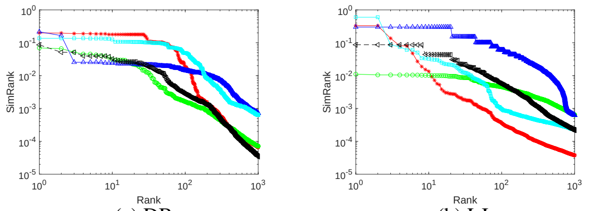
Node	$s(u_1, v)$	$\hat{s}(u_1, v)$
v_1	0.28	0.3
v_2	0.26	0.25
v_3	0.08	0.1
v_4	0.06	0.05

Node	$s(u_2, v)$	$\hat{s}(u_2, v)$
v_1	0.09	0.085
v_2	0.08	0.1
v_3	0.07	0.09
v_4	0.065	0.06

Motivations. Although single-pair and single-source queries have been extensively studied, very few works directly consider the top-k or thresholding queries. The only known method is *TopSim* [21], which is specially designed for top-k queries. *TopSim* estimates similarity values by forward (i.e., following in-edges) and then backward (i.e., following out-edges) traversals level by level from the query node. A stopping rule is used to speculate if the k-th largest estimated SimRank value is larger than the *heuristic* upper bound of the $k + 1$-th to the $|V|$-th largest ones. However, the heuristics sacrifices the query accuracy for speed, and always leads to inferior answer quality. On the other hand, algorithms for single-source queries [17, 24, 28, 30, 32] can naturally be extended to answer top-k and thresholding queries following the *return all and postprocessing* paradigm. For example, to answer top-k queries it first computes an approximate similarity for each node, followed by sorting all estimations and returns the nodes with top-k largest SimRank values. Among them, the algorithms [24, 30, 32] with absolute error guarantee achieve state-of-the-art performance.

Nonetheless, all these algorithms suffer from a few deficiencies in answering the top-k or thresholding queries. First, most of them [17, 28, 30, 32] are index-based, which pre-compute a fraction of the intermediate results to accelerate query-time performance. As a consequence, they have quite limited flexibility to answer SimRank queries on dynamic graphs or with the user-defined error parameter. Second, the absolute error guarantee of the single-source query turns out to have little correlation with the answer quality of top-k and thresholding queries. As an example, Table 1 demonstrates the similarity of four nodes $\{v_1, v_2, v_3, v_4\}$ w.r.t. query node u_1 and u_2. Although the estimation achieves an absolute error of 0.05, the precision of the top-2 query varies significantly (1 vs. $\frac{1}{2}$). To the best of our knowledge, no single-source algorithm can achieve acceptable answer quality with reasonable query speed, even on a million-node graph.

Contributions. In this paper, we improve both the query efficiency and effectiveness of top-k and thresholding SimRank queries in a unified manner. We refer to the algorithm as *SimTab* (SimRank queries with Tighter confidence bounds and multi-armed bandits), which is sampling-based and depends on the random walk interpretation of SimRank. In particular, we model the top-k and thresholding queries from the perspective of the Multi-Armed Bandit (MAB) problems. The MAB problem takes a set of n arms as input, while each sample of an arm yields a reward in $[0, 1]$, which is generated randomly following a fixed distribution associated with the arm. Based on this, a bunch of problems [7] have been proposed and



(a) DB (b) LJ

Figure 1: The distribution of top-1000 largest SimRank values for five random query nodes. For each SimRank value, the estimation error is below 10^{-6}.

well studied, such as the top-k arm identification problem, which aims to find the arms with top-k largest rewards by as few samples as possible. Since existing SimRank algorithms [10, 17, 28] answer SimRank query by generating random walks from the corresponding nodes repeatedly and independently, the related SimRank queries can be naturally modeled as the corresponding MAB problems. However, these algorithms treat each node equally, i.e., each node is sampled the same times. By integrating the arm sampling strategies, our algorithm treats each node differently by their SimRank values in the sampling procedure, which significantly improves the sample complexity. For our proposed method, the query complexity is dependent on the hardness of the query instance and closely matches the theoretical sampling complexity for the corresponding MAB problem. In particular, given a query node u, the hardness of the top-k query is determined by $H_k = \sum_{v \in V} \frac{1}{\Delta_v{}^2}$, where intuitively Δ_v denotes the gap between $s(u, v)$ and the k-th largest SimRank w.r.t. u. As for the thresholding query with parameter τ, the hardness is described by $H_\tau = \sum_{v \in V} \frac{1}{\Delta_{\tau,v}{}^2}$, whereas $\Delta_{\tau,v} = |s(u, v) - \tau|$ for each node v.

Notably, we observe that the SimRank similarities have skewed distribution in most cases, partially due to the power-law distribution of real-world graphs[2]. Meanwhile, the absolute SimRank values for different query nodes also vary dramatically (shown in Figure 1). This phenomenon produces a great influence on the problem complexity as the query parameter varies. For example, nodes with small SimRank values are hard to be distinguished from each other since the gap between them is small. Hence, a large number of samples are needed for top-k queries with large k, or thresholding queries with small τ. To improve the practical efficiency, we propose a novel sampling strategy specially tailored for SimRank queries, for that the MAB algorithms have severe scalability problems on large-sized graphs. We employ several techniques to tighten the confidence bound in SimRank estimation, including the empirical Bernstein inequality [27], a few variance reduction tricks, and careful algorithm design. It turns out that the tightness of confidence bound not only improves the accuracy of SimRank estimation, but also has a major effect on the practical efficiency of the algorithms. Our algorithms are the first to answer *exact* top-k and thresholding queries, i.e., the query result can be at least as good as the *Power Method* [16], which is taken as the ground truth [24, 30]. Last but not least, our algorithms are index-free, which can be naturally applied to dynamic graphs. Table 2 compares our proposed algorithms with state of the art.

Finally, we conduct extensive experiments to evaluate our algorithms against state-of-the-art methods on several large datasets. For both top-k and thresholding queries, we provide the first detailed analysis to demonstrate the performance of existing algorithms on different query parameters (i.e. k and threshold τ).

[2]We use this term to describe the skewed distribution observed in a variety of networks.

Table 2: Comparison of SimRank algorithms for top-k and thresholding queries. For the empirical performance, non-stable means the evaluation metric (e.g., Precision@k for top-k queries) fluctuates when varying the query parameters (k and τ). Note that the complexity of the MAB algorithms for top-k and thresholding queries are $O(H_{k,\varepsilon_{min}} \log \frac{n}{\delta})$ and $O(H_{\tau,\varepsilon_{min}} \log \frac{n}{\delta})$, respectively. (See Section 2.4 for definition of $H_{k,\varepsilon_{min}}$ and $H_{\tau,\varepsilon_{min}}$.) Therefore, our proposed algorithms also improve the theoretical efficiency.

Method	Query Complexity	Index Cost (Space, Time)	Theoretical Guarantee	Empirical Performance
TopSim [21]	$O(\bar{d}^{2l})$	N/A	N/A	Non-stable
TSF [28]	$O(R_g R_q T^2)$	$O(R_g\|V\|)$	N/A	Non-stable
SLING [30]	$O(\frac{n}{\varepsilon})$ or $O(m \log^2 \frac{1}{\varepsilon})$	$O(\frac{n}{\varepsilon}), O(\frac{m}{\varepsilon} + \frac{n}{\varepsilon^2} \log \frac{n}{\delta})$	Absolute error	Non-stable
ProbeSim [24]	$O(\frac{n}{\varepsilon^2} \log \frac{n}{\delta})$	N/A	Absolute error	Non-stable
READS [17]	$O(\frac{n}{\varepsilon^2} \log \frac{n}{\delta})$	$O(\frac{n}{\varepsilon^2} \log \frac{n}{\delta})$	Absolute error	Non-stable
PRSim [32]	$O(\frac{n}{\varepsilon^2} \log \frac{n}{\delta} \cdot \sum_{w \in V} \pi(w)^2)$	$O(\min(\frac{n}{\varepsilon}, m)), O(\frac{m}{\varepsilon})$	Absolute error	Non-stable
SimTab-Top-k	$\min(O(\frac{\theta}{\varepsilon_{min}^2} \log \frac{n}{\delta}), O(H_{k,\varepsilon_{min}} \log \frac{n}{\delta}))$	N/A	Exact answer	Good and stable
SimTab-Thres	$\min(O(\frac{\theta}{\varepsilon_{min}^2} \log \frac{n}{\delta}), O(H_{\tau,\varepsilon_{min}} \log \frac{n}{\delta}))$	N/A	Exact answer	Good and stable

We adopt Precision@k to evaluate the quality of top-k query results, while for thresholding queries, we measure precision, recall, and F1-score. We also empirically demonstrate that query accuracy does not have a direct correlation with the absolute estimation error. For all studied experiments, our algorithms significantly outperform existing methods in terms of both practical efficiency and effectiveness. In particular, all existing methods fail to answer both queries with acceptable query time and result quality even on a million-node graph, while our algorithms are able to return the exact query result with reasonable speed on billion-edge graphs.

2. PRELIMINARIES

Table 3 shows the notations that are frequently used in the remainder of the paper.

Table 3: Table of notations.

Notation	Description
$G(V, E)$	Graph G with vertex set V and edge set E
n, m	Numbers of nodes and edges in G
$I(v), O(v)$	In-neighbor/out-neighbour set of a node v in G
$s(u, v)$	SimRank similarity between two nodes u and v in G
$\hat{s}(u, v)$	Estimation of $s(u, v)$
$W(u)$	A $\sqrt{c}$-walk from a node u
c	Decay factor in the definition of SimRank
ε, δ	Additive error parameter and failure probability
$\beta(v)$	Confidence interval of the estimated similarity of node v, i.e., with high probability $\hat{s}(u, v) - \beta(v) \leq s(u, v) \leq \hat{s}(u, v) + \beta(v)$
t_a, t_o	The running time of one sample-all-arms (resp. sample-one-arm) operation
θ	t_a/t_o

2.1 SimRank with Random Walks

As indicated in [16], SimRank similarities can be interpreted with coupled random walks. In particular, let u and v be two nodes in G, and $I(u)$ (resp. $I(v)$) be a random walk from u (resp. v) that follows a randomly selected incoming edge at each step. Let t be the step that $I(u)$ and $I(v)$ *first meet*, we have

$$s(u, v) = \mathbb{E}[c^{t-1}], \tag{2}$$

where c is the decay factor in the definition of SimRank (see Equation 1). Subsequent work [17, 30] demonstrate that Equation 2 can be simplified by considering the probabilistic stop at each step. For example, the $\sqrt{c}$*-walk* [30] is defined as follows.

DEFINITION 3 ($\sqrt{c}$-WALK). *Given a node u in G, a $\sqrt{c}$-walk from u is a random walk that follows the incoming edges of each node and stops at each step with $1 - \sqrt{c}$ probability.* □

Consequently, two $\sqrt{c}$-walks $W(u) = (w_0 = u, \ldots, w_l, \ldots)$ and $W(v) = (w'_0 = v, \ldots, w'_l, \ldots)$ from u and v meet if there exists some l such that $w_l = w'_l$. According to [30],

$$s(u, v) = \Pr[W(u) \text{ and } W(v) \text{ meet}]. \tag{3}$$

Based on this random walk interpretation of SimRank, the Monte Carlo approach [10, 17, 30] estimates $s(u, v)$ as follows. The algorithm generates n_r coupled $\sqrt{c}$-walks from u and v, respectively. Let $n_{r,meet}$ be the number of $\sqrt{c}$-walk pairs that meet, then $\frac{n_{r,meet}}{n_r}$ is used as an estimation of $s(u, v)$. The estimation error is guaranteed by the Chernoff-Hoeffding inequality [17, 30].

LEMMA 1 (CHERNOFF-HOEFFDING INEQUALITY [14]). *Let $X_1, ..., X_{n_r}$ be independent random variables where X_i is strictly bounded by the interval $[a_i, b_i]$ for every $i \in [1, n_r]$. Let $\bar{X} = \frac{1}{n_r} \sum_{i=1}^{n_r} X_i$. Then*

$$\Pr[|\bar{X} - \mathrm{E}[\bar{X}]| \geq \varepsilon] \leq 2\exp\left(-\frac{2{n_r}^2\varepsilon^2}{\sum_{i=1}^{n_r}(b_i - a_i)^2}\right). \tag{4}$$

Since each pair of walks gives an unbiased estimation of the SimRank value, we have $\mathrm{E}[\bar{X}] = s(u, v)$. Given n_r and the constraint on failure probability, i.e., $\Pr[|\bar{X} - s(u, v)| \geq \varepsilon] \leq \delta$, we have $\beta_{n_r} = |\bar{X} - s(u, v)| \geq \sqrt{\frac{1}{2n_r} \log \frac{2}{\delta}}$. We refer to β_{n_r} as the *confidence interval*, which means that with $1-\delta$ probability $s(u, v)$ falls into $[\bar{X} - \beta_{n_r}, \bar{X} + \beta_{n_r}]$. In particular, $\bar{X} - \beta_{n_r}$ (resp. $\bar{X} + \beta_{n_r}$) is referred to as the lower (resp. upper) confidence bound. It can be shown that when $n_r \geq \frac{1}{2\varepsilon^2} \log \frac{2}{\delta}$, with at least $1 - \delta$ probability we have $\left|\frac{n_{r,meet}}{n_r} - s(u, v)\right| \leq \varepsilon$. In addition, the expected time required to generate n_r $\sqrt{c}$-walks is $O(n_r)$, since each $\sqrt{c}$-walk has $\frac{1}{1-\sqrt{c}}$ nodes in expectation. Therefore, the expected time complexity of answering a single-pair query is $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta})$.

Note that *MC* can be straightforwardly extended to answer single-source queries by conducting single-pair query for each node $v \in V \backslash \{u\}$ and the query node u. To guarantee the absolute estimation error ε for every node v with at least $1 - \delta$ probability, by the union bound the complexity is $O(\frac{n}{\varepsilon^2} \log \frac{n}{\delta})$, where n denotes the number of vertices in G. Since each node has to generate a large number of $\sqrt{c}$-walks, this approach incurs considerable query overheads for large graphs, and is practically infeasible.

2.2 The Forward and Backward Random Walk Scheme

To improve the practical efficiency of the Monte Carlo method for single-source queries, a few works [17, 21, 24, 28, 30, 32] have been recently proposed based on the random walk interpretation of SimRank. We unify them as *the forward and backward random walk scheme*, as listed in Table 4. Generally speaking, all these methods contain two stages in the SimRank computation, i.e., the *forward* stage and the *backward* stage, while both stages can be implemented in a deterministic or randomized way. Specifically, the deterministic computation relies on the following equation, which enumerates all coupled *similarity paths* [21] from u and v that meet:

$$s(u,v) = \sum_{t=1}^{\infty} \sum_{w \in V} p_{ft}(u,v,w) \cdot c^t. \tag{5}$$

Here, we denote by $p_{ft}(u,v,w)$ the probability of two random walks from u and v first meet at w. On the other hand, the randomized computation is based on Equation 3. In the forward stage, the algorithm deterministically enumerates all reachable nodes w from the query node u following *in-edges*, or randomly samples a subset from them. In the backward stage, from each w, a deterministic or randomized traversal following *out-edges* is conducted to reach a set of nodes v, and we can estimate $s(u,v)$ accordingly. By implementing the forward and backward stage with different strategies, the efficiency and effectiveness of SimRank computation vary significantly. We will discuss the key idea of existing solutions in Section 5.

2.3 The Relation between SimRank and Personalized PageRank

Inspired by [30], *PRSim* [32] proposes a new interpretation of SimRank, where $s(u,v)$ is closely related to the reverse Personalized PageRank of both u and v. Formally, we have the following Equation:

$$s(u,v) = \frac{1}{(1-\sqrt{c})^2} \sum_{l=0}^{\infty} \sum_{w \in V} \pi_l(u,w)\pi_l(v,w)d(w). \tag{6}$$

Here, $\pi_l(u,v)$ is the l-hop Reverse Personalized PageRank (RPPR) from u to v, i.e., the probability of an $\sqrt{c}$-walk from u stopping at v with exact l steps ("reverse" means that each step of the walk follows in-edges), while $d(w)$ represents the probability that two $\sqrt{c}$-walks starting from w never meet again [30]. As we will see in Section 3 and 4, this interpretation enables us to apply a few techniques to significantly tighten the confidence bound in SimRank estimation, such as the forward push [5].

Forward push [5]. The forward push is proposed to compute the Personalized PageRank deterministically. Specifically, let $\pi(s,t)$ denote the PPR values between the source node s and the target node t, which represents the probability of an $\sqrt{c}$-walk from s stopping at t. To estimate $\pi(s,t)$, we initialize the *reserve* $\hat{\pi}_f(s,v) = 0$ for each $v \in V$, which is an underestimation of $\pi(s,v)$. Meanwhile, we initialize the residue $r_f(s,v) = 0$ for $v \in V \backslash \{s\}$ and $r_f(s,s) = 1$. Intuitively, $r_f(s,v)$ denotes the probability staying at node v that has not been handled yet. The push operation on a node v first transmits $1-\sqrt{c}$ fraction of its residue to its reserve, then evenly distributes the remaining residue to its out neighbors. This process can be formulated by the following Equation: $\hat{\pi}_f(s,v) \leftarrow \hat{\pi}_f(s,v) + (1-\sqrt{c})r_f(s,v), r_f(s,u) \leftarrow r_f(s,u) + \frac{\sqrt{c}}{|O(v)|} r_f(s,v), \forall u \in O(v)$. As more push operations are conducted, the residues are transferred into the reserves, resulting in a more accurate estimation of $\pi(s,t)$. The following Equation holds in any step of forward push, for $s,t \in V$: $\pi(s,t) = \hat{\pi}_f(s,t) + \sum_{v \in V} r_f(s,v)\pi(v,t)$. We omit the subscripts when the context is clear.

2.4 The Multi-Armed Bandits Problem

In this paper, we answer top-k and thresholding SimRank queries by modeling them as the corresponding multi-armed bandits (MAB) problems. We briefly describe the problem setting as follows. The MAB problem considers an arbitrary instance of an n-armed bandit ($n \geq 2$). Each arm a is associated with a fixed but unknown distribution with expected reward $p_a \in [0,1]$, while each sample (or "pull") of the arm yields a reward generated randomly from it. The rewards for different trials of an arm or between different arms are mutually independent.

A variety of MAB problems have been extensively studied in the field of theoretical computer science, such as regret minimization [6] and pure exploration [7]. For the latter, it considers finding the best set of arms meeting some specific criteria via the minimum number of arm pulls. In particular, the top-k arm identification [12, 15, 19] problem finds the top-k arms with the largest rewards.

DEFINITION 4. *(The top-k arm identification problem [7]) Given n arms and a failure probability δ, find k arms that have the top-k largest rewards, by using as few samples (i.e. pull of arms) as possible, with at least $1-\delta$ success probability.*

We will demonstrate (later in Section 3.1) how to convert SimRank computation to the MAB problems through its random walk interpretation. In this paper, we also show that the thresholding SimRank queries can be modeled as the following MAB problem, which returns all arms with expected reward above some threshold.

DEFINITION 5. *(The thresholding bandits problem [7, 26]) We are given a set of n arms and a failure probability δ. Find all arms that have estimated rewards above a given threshold τ, by using as few samples as possible.*

Sampling complexity. It has been shown [7, 19, 26] that the minimum number of arm pulls (i.e., the sampling complexity) for MAB is determined by the expected reward of each arm. Specifically, for the top-k problem, the sampling complexity is defined as $H_k = \sum_{i=1}^{n} \frac{1}{\Delta_i{}^2}$ [19], where

$$\Delta_i = \begin{cases} p_i - p_{k+1}, & \text{if } i \leq k; \\ p_k - p_i, & \text{if } i > k. \end{cases} \tag{7}$$

Intuitively, Δ_i characterizes the hardness to differentiate the i-th arm a_i from the actual top-k results. Here, for simplicity of notation, we assume an indexing of the arms such that $p_1 \geq p_2 \geq \ldots \geq p_n$. Similarly, the sample complexity of the thresholding bandits problem can be defined as $H_\tau = \sum_{i=1}^{n} \frac{1}{\Delta_{\tau,i}{}^2}$ [7, 26], where $\Delta_{\tau,i} = |p_i - \tau|$ denotes the gap between the reward of the i-th arm and the given threshold.

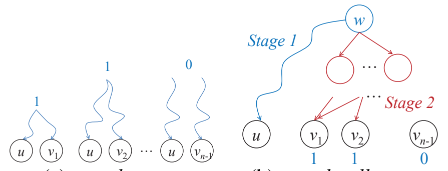
We also note that *all* existing MAB solutions [12, 19, 26] introduce a small error parameter ε_{min} to guarantee that the sampling-based procedure can terminate in extreme cases, e.g., several arms have (nearly) identical expected reward. In this way, arms with expected rewards no smaller than ε_{min} from the actual answer are also considered correct. Correspondingly, the sampling complexity for top-k (resp. thresholding) queries is defined as $O(H_{k,\varepsilon_{min}}) = O((\sum_{i=1}^{n} \frac{1}{\max(\Delta_i, \varepsilon_{min})^2}))$ [12, 19] (resp. $O(H_{\tau,\varepsilon_{min}}) = O((\sum_{i=1}^{n} \frac{1}{(\Delta_{\tau,i} + \varepsilon_{min})^2}))$ [26]).

Table 4: **Algorithms following the forward and backward random walk scheme.**

Method	The forward stage	The backward stage
TopSim [21]	similarity path enumeration	similarity path enumeration
TSF [28]	random walk sampling	similarity path enumeration (on the indexed one-way graph)
SLING [30]	similarity path enumeration (with pruning)	similarity path enumeration (with pruning)
ProbeSim [24]	$\sqrt{c}$-walk sampling	deterministic or randomized path enumeration, i.e., *DeterministicProbe* or *RandomizedProbe*
READS [17]	(SimRank-aware) random walk sampling	(SimRank-aware) random walk sampling and indexing
PRSim [32]	$\sqrt{c}$-walk sampling	deterministic or randomized path enumeration, e.g., *Variance Bounded Backward Walk*

3. TOP-K QUERIES

3.1 Sampling Strategy in SimRank Computation: from the Perspective of MAB



(a) *sample-one-arm* (b) *sample-all-arms*

Figure 2: The sampling strategies for SimRank: *sample-one-arm* vs. *sample-all-arms*.

3.1.1 Modeling SimRank computation via MAB

In this section, we demonstrate how to model the top-k SimRank query as the corresponding top-k arm identification problem from the perspective of multi-armed bandits. Given a query node u, for all nodes $V\backslash\{u\} = \{v_1, \ldots, v_{n-1}\}$, we construct $n-1$ arms with expected reward $s(u, v_1), \ldots, s(u, v_{n-1})$. Note that the expected reward of each arm is not known to the algorithm, but can be estimated via sampling according to the random walk interpretation of SimRank. To be precise, for each v_i, a "pull" of the arm is implemented by sampling a pair of $\sqrt{c}$-walks from the query node u and v_i, respectively (Figure 2(a)). The result of each sampling is either 0 or 1, and gives an unbiased estimation of $s(u, v_i)$ [24,30]. Therefore, designing an efficient top-k SimRank algorithm is equivalent to finding top-k arms with the largest rewards by using as few samples as possible, i.e., the *top-k arm identification* problem [12, 19]. We refer to this sampling strategy as *sample-one-arm*, since each arm (i.e., node pair) is sampled independently.

3.1.2 Sample-all-arms: another arm sampling strategy for SimRank

Unfortunately, for any known top-k MAB algorithm, each arm needs to be sampled at least once so that the confidence bound can be computed. The cost, however, is unacceptable for the top-k SimRank queries, which contains a large amount of arms (i.e., nodes). In fact, as our empirical analysis shows, even the state-of-the-art MAB algorithms cannot finish in a reasonable time on a graph with million-sized nodes. On the other hand, we notice that there exists another sampling strategy specially tailored for SimRank queries. Recall the forward and backward random walk scheme proposed in Section 2.2. For the algorithms following this paradigm, a few of them [24,32] adopt random walk sampling in the forward stage. To be precise, they first sample a set of random walks, while for each walk, the backward stage computes an estimation of SimRank similarities for *all* nodes. If the estimation is *unbiased* and *bounded*, we can model this forward and backward procedure as the *sample-all-arms* strategy (Figure 2(b)). For real-world graphs following power-law distribution, the distribution of SimRank scores is highly skewed in practice. Hence, the sample-all-arms strategy achieves much superior practical efficiency than applying sample-one-arm strategy for all nodes (e.g., the *MC* algorithm [9]), because only a fraction of nodes can be reached during the backward searching stage. We will demonstrate in Section 3.3 how to design such a procedure to meet this criterion.

```
Algorithm 1: SimTab-Top-k
  Input: Directed graph $G = (V, E)$; $u \in V$; $k \in [1, n)$; failure
         probability $\delta$
  Output: $V_k$, the estimated top-$k$ nodes with largest SimRank
          values
1 $C$ = Prefiltering($G, u, k, \frac{\delta}{2}$);
2 $V_k$ = Top-$k$-Identification($G, u, k, C, \frac{\delta}{2}$);
3 return $V_k$;
```

3.2 Algorithm Overview

We propose *SimTab-Top-k*, a two-phase algorithm that takes the advantages of both sample-one-arm and sample-all-arms strategies. Specifically, our algorithm contains a *prefiltering* phase and a *top-k identification* phase. In the prefiltering phase, we iteratively apply the sample-all-arms strategy to compute the upper and lower confidence bounds for each estimated SimRank value $\hat{s}(u, v)$. Meanwhile, nodes with low SimRank values that make them impossible to be top-k answers are safely pruned. We refer to the set of remaining nodes as the *candidates*. Once the size of the candidate set is small enough so that applying the sample-one-arm strategy for each candidate is more economical, we stop the prefiltering phase and proceed to the top-k identification phase. Then, a MAB-based algorithm is invoked to keep sampling the nodes independently and following a specified strategy, until we are confident to separate the top-k nodes from other candidates. In this way, the algorithm has at least an identical complexity to the top-k bandits algorithm but achieves much more efficiency in practice.

The pseudo-code is illustrated in Algorithm 1. We first invoke the *Prefiltering* algorithm, which guarantees to return a candidate set C which contains *all* actual top-k nodes with at most $\frac{\delta}{2}$ failure probability. Next, we invoke the *Top-k-Identification* algorithm which finds top-k nodes among the candidates, again with at most $\frac{\delta}{2}$ failure probability. Hence, with high probability, *SimTab-Top-k* returns the true top-k results.

3.3 The Prefiltering Phase

In this section, we propose a *prefiltering* algorithm that can efficiently prune nodes with low SimRank similarities, which is illustrated in Algorithm 2. We adopt an iterative process to continuously apply the sample-all-arm strategy (Lines 3-5), of which the implementation will be amplified later. Specifically, we invoke the sample-all-arms operation 2^{i-1} times in the i-th iteration. For each iteration, the estimated SimRank value $\hat{s}(u, v)$ and confidence interval $\beta(v)$ of every node v are computed from all previously conducted sample-all-arms operations. After that, the top-k nodes with the largest empirical means are added to the candidate set (Line

Algorithm 2: Prefiltering

```
Input: $G = (V, E)$; $u \in V$; $k$; failure probability $\delta$;
Output: Candidate node set $C$
1  Initialize $n_r = 0, numSample = 1$;
2  while true do
3      for $i = 1$ to $numSample$ do
4          $R$ = Sample-all-arms$(G, u)$;
5          Update $\hat{s}(u, v)$ and $\beta(v)$ for every $v \in V$ with $R$;
6      Let $C = \{v_1, \ldots, v_k\}$ be the nodes with top-$k$ empirical means;
7      Let $v' = \text{argmin}_v\{\hat{s}(u, v) - \beta(v)\}, v \in C$;
8      for each $v \in V\backslash(\{u\} \cup C)$ do
9          if $\hat{s}(u, v') - \beta(v') + \varepsilon_{min} \leq \hat{s}(u, v) + \beta(v)$ then
10             $C = C \cup \{v\}$;
11     if $|C|t_o \leq t_a$ then
12         return $C$;
13     else if $\max_{v \in C} \beta(v) \leq \varepsilon_{min}$ then
14         Sort $C$ according to the empirical means;
15         return the first $k$ nodes in $C$, and skip the top-$k$ identification phase;
16     Increase $n_r$ by $numSample$ and double $numSample$;
```

Algorithm 3: Sample-all-arms

```
Input: $G = (V, E)$; query node $u$;
Output: $R = \{(v, Score(u, v))\}$, a set of nodes with non-zero estimated SimRank values
1  Sample an $\sqrt{c}$-walk $W(u) = (w_0 = u, \ldots, w_l)$ from $u$;
2  Sample two independent $\sqrt{c}$-walks $W_1(w_l)$ and $W_2(w_l)$ from $w_l$;
3  if $W_1(w_l)$ and $W_2(w_l)$ do not meet then
4      Initialize hash set $H_j$ for $j = 0, \ldots, l-1$;
5      Insert $w_l$ to $H_0$;
6      for $j = 0$ to $l - 2$ do
7          if $\sum_{x \in H_j} |O(x)| < n$ then
8              $U = \cup_{x \in H_j} O(x)$;
9          else
10             $U = V\backslash\{u\}$;
11         for each $y \in U$ do
12             Uniformly sample an edge $(x, y)$ from $I(y)$;
13             if $x \in H_j$ then
14                 Insert $y$ to $H_{j+1}$ with probability $\sqrt{c}$;
15     return $R = \{(v, \frac{1}{(1-\sqrt{c})^2}) | v \in H_{l-1}\}$;
```

6). Then, we find the node v' with the smallest lower confidence bound, i.e., $\hat{s}(u, v) - \beta(v)$ (Line 7). Next, we check for each $v \in V\backslash(\{u\} \cup C)$ if it can be pruned from the top-k answers (Lines 8-10). Specifically, we have the following pruning rule, of which the correctness can be easily derived.

The pruning rule. If $\hat{s}(u, v') - \beta(v') + \varepsilon_{min} > \hat{s}(u, v) + \beta(v)$ for some node v, it means that with at least $1 - \delta'$ probability $\min_{v' \in C} s(u, v') + \varepsilon_{min} > s(u, v)$ holds, i.e., v can not be in top-k answer, and will be safely pruned.

For the computation of confidence interval, we use the empirical Bernstein inequality [8]. Intuitively, it states that if the empirical variance is small, then the confidence interval is reversely related to the number of samples n_r. This bound is significantly tighter than that of the Chernoff-Hoeffding inequality, which is in reverse proportional to $\sqrt{n_r}$.

LEMMA 2 (EMPIRICAL BERNSTEIN INEQUALITY [8]). *For any set $\{X_i\}$ ($i \in [1, t]$) of i.i.d. random variables with mean μ and $X_i \in [0, R]$, with $1 - \delta$ probability,*

$$\left|\bar{X}_t - \mu\right| \leq \hat{\sigma}_t \sqrt{\frac{2 \ln 3/\delta}{t}} + \frac{3R \ln 3/\delta}{t}, \tag{8}$$

where $\bar{X}_t = \sum_{i=1}^{t} X_i/t$ is the empirical mean of $\{X_i\}$, and $\hat{\sigma}_t = \sqrt{\frac{1}{t}\sum_{i=1}^{t}(X_i - \bar{X}_t)^2}$ is the empirical standard deviation of $\{X_i\}$.

Hence, the confidence bound is set to $\beta(v) = \hat{\sigma}(v)\sqrt{\frac{2 \ln 3n/\delta}{n_r}} + \frac{3 \ln 3n/\delta}{n_r}$, where n_r is the total number of sample-all-arms operation ever conducted.

The stopping condition. Now we return back to the prefiltering algorithm and explain its stopping condition. In particular, we set the stopping condition as $|C|t_o \leq t_a$ (Line 11), where t_a (resp. t_o) denotes the expected running time of one sample-all-arms (resp. sample-one-arm) operation. (Note that the cost of a single sample-one-arm operation is asymptotically identical to one $\sqrt{c}$-walk generation.) Therefore, $|C|t_o$ denotes the cost of applying sample-one-arm strategy to each candidate node, and the prefiltering algorithm stops when applying sample-one-arm strategy on *all* candidates is more efficient.

When the stopping condition is not satisfied but the confidence interval of *every* candidate is below ε_{min} (Line 13), the algorithm can still safely terminates. To this end, the prefiltering algorithm computes an approximate SimRank with absolute error ε_{min} for every candidate node. We sort all node in C by descending order of the estimated SimRank values, and return the top-k largest nodes as the final answer (i.e., the top-k-identification phase is skipped). Lastly, if the algorithm cannot finish in this round, we double the number of sample-all-arm operations, and proceed to the next iteration (Line 16).

The *Sample-all-arms* algorithm. The implementation of the sample-all-arms operation is demonstrated in Algorithm 3. It generates a $\sqrt{c}$-walk $W(u) = (w_0 = u, ..., w_l)$, and then samples two $\sqrt{c}$-walks $W_1(w_l)$ and $W_2(w_l)$ from w_l. If the walks do not meet, we invoke a backward traversal procedure from w_l (Lines 4-14). Intuitively, the traversal guarantees that for each $v \in V\backslash\{u\}$, the probability of v contained in H_{l-1} equals the probability that a $\sqrt{c}$-walk from v stops at w_l. Note that by sampling two walks from w_l, we can get an unbiased estimation of $d(w_l)$. Hence, by Equation 6, the Sample-all-arms algorithm computes an unbiased estimation of $s(u, v)$ for *each* v.

Algorithm correctness. We formally prove the correctness of the prefiltering algorithm, which guarantees that all actual top-k nodes are contained in the returned candidate set with high probability. For space constraints, all proofs in our paper are referred to its full version [1]. The following lemma guarantees the effectiveness of the *Sample-all-arms* algorithm.

LEMMA 3. *For each node $v \in V\backslash\{u\}$,* Sample-all-arms *returns an unbiased estimator of $s(u, v)$ that falls in* $[0, 1]$.

From the above lemma and the pruning rule of the prefiltering algorithm, we have the following theorem.

THEOREM 1. *With at least $1 - \delta$ probability, the prefiltering algorithm guarantees that the returned candidate set C contains all actual top-k nodes.*

3.4 The Top-k Identification Phase

3.4.1 Baseline algorithm

In the top-k identification phase, we try to select the top-k nodes by separating them from the candidate nodes that are not likely to be in the top-k answer. We demonstrate how to adapt the MAB algorithms for top-k arm identification to our scenario. We use the UGapEc-V algorithm [12] as an example. The pseudo-code is demonstrated in Algorithm 4. Given graph G, the query node u, an integer k, the candidate set C which contains the estimated SimRank $\hat{s}(u,v)$ and the confidence interval $\beta(v)$ for each candidate v, and a failure probability δ, *Top-k-Identification* returns the actual top-k nodes contained in C. Note that to efficiently update the confidence bound we do not explicitly store $\beta(v)$. For the Chernoff bound, it is sufficient to record $n_r(v)$, the number of samples for node v. If the empirical Bernstein inequality is applied, to compute the empirical variance, we also need to store $\sum_{i=1}^{n_r(v)} \hat{s}_i(u,v)^2$, where $\hat{s}_i(u,v)$ denotes the estimation of $s(u,v)$ in the i-th trial. The algorithm first computes the upper and lower confidence bound for each $v \in C$, denoted as $UB(v)$ and $LB(v)$, respectively. Then, for each C we compute $B(v) = \overset{k}{\max\limits_{w \in C\setminus\{v\}}} UB(w) - LB(v)$, where operator $\overset{k}{\max\limits_{v \in S}} f(v)$ returns the k-th largest value $f(v)$ among all $v \in S$. Intuitively, $B(v)$ represents how bad is a candidate compared to the one with the k-th largest similarity in the worst case. Next, we find the k nodes with smallest $B(v)$, denoted as V_k. Let v_k be the node with the k-th smallest $B(v)$. If $B(v_k) \le \varepsilon_{min}$, we are confident that V_k is the top-k nodes (with additive error ε_{min}). Otherwise, let v_h be the node in V_k having the smallest $LB(v)$, and v_l the node in $C\setminus V_k$ having the largest $UB(v)$. They represent the worst possible arm in V_k and the best possible arm in $C\setminus V_k$, respectively. Then, we sample the arm in $\{v_h, v_l\}$ with the larger confidence interval. Note that each arm sampling is implemented by a coupled random walk from the query node u and the candidate v. Following [12], the confidence bound is computed based on the empirical Bernstein inequality:

$$\beta(v, n_r(v)) = \hat{\sigma}(v, n_r(v))\sqrt{\frac{\log\frac{|C|r^3}{\delta}}{n_r(v)}} + \frac{\frac{7}{6}\log\frac{|C|r^3}{\delta}}{n_r(v)-1}, \quad (9)$$

where $\hat{\sigma}(v, n_r(v))$ denotes the empirical standard variance over $n_r(v)$ estimations, and r the round of iteration.

Running example for Algorithm 4. We use a toy example in Figure 3(a) to illustrate the arm sampling strategy and the stopping rule of the top-k algorithm. Suppose that for the query node u, there exists a candidate set C of five candidate nodes $\{v_1, \dots, v_5\}$ after the prefiltering phase, and we aim to find the top-2 nodes with the largest similarity. For each node v_i, let μ_i denote the estimated SimRank score, i.e., $\mu_i = \hat{s}(u, v_i)$, and β_i the corresponding confidence interval. Without loss of generality, we assume that the candidates are numbered in descending order of the estimated similarities. Note that even these candidates are sampled identical times via the sample-all-arm strategy, their confidence intervals can be different when the empirical Bernstein inequality is applied.

The algorithm first computes the upper and lower confidence bound for each candidate v_i, denoted as UB_i and LB_i, respectively. In particular, we have $\{(UB_1 = 0.11, LB_1 = 0.07), (UB_2 = 0.11, LB_2 = 0.05), (UB_3 = 0.1, LB_3 = 0.04), (UB_4 = 0.09, LB_4 = 0.05), (UB_5 = 0.08, LB_5 = 0.04)\}$. Second, we compute B_i for each node v_i. Note that for v_1 and v_2 with top-2 largest estimated score, $\overset{2}{\max\limits_{w \in C\setminus\{v\}}} UB(w) = UB_3 = 0.1$, whereas for v_3, v_4 and v_5, $\overset{2}{\max\limits_{w \in C\setminus\{v\}}} UB(w) = UB_2 = 0.11$. (Since v_1 and v_2 have identical upper bound, the tie is broken randomly.) Therefore, we have $\{B_1 = 0.03, B_2 = 0.05, B_3 = 0.07, B_4 = 0.06, B_5 = 0.07\}$. We set V_2 as the two nodes with the smallest B_i, i.e., $V_2 = \{v_1, v_2\}$. Since the second smallest B_i is B_2, and $B_2 = 0.05 > \varepsilon_{min}$, more sample-one-arm operations are needed. We select $v_h = v_2$, the node with the smallest LB from V_2, and $v_l = v_3$, the node with the largest UB from $C\setminus V_2$. Since they have the same confidence interval, we sample v_2 in the next iteration (again, the tie is broken randomly).

Algorithm 4: Top-k-Identification

```
Input: $G = (V, E)$; $u \in V$; $k$; candidate set
       $C = \{v, \hat{s}(u,v), \beta(v)\}$; failure probability $\delta$;
Output: $V_k$ as the top-$k$ results
1  while true do
2      for each $v \in C$ do
3          Compute $UB(v) = \hat{s}(u,v) + \beta(v)$,
           $LB(v) = \hat{s}(u,v) - \beta(v)$;
4      for each $v \in C$ do
5          Compute $B(v) = \overset{k}{\max\limits_{w \in C\setminus\{v\}}} UB(w) - LB(v)$;
6      Let $V_k$ be the set of $k$ nodes with smallest $B(v)$, and $v_k$ be
       the node with $k$-th smallest $B(v)$;
7      if $B(v_k) \le \varepsilon_{min}$ then
8          return $V_k$;
9      else
10         Let $v_h \in V_k$ be the node with smallest $LB$;
11         Let $v_l \in C\setminus V_k$ be the node with largest $UB$;
12         Sample node $v \in \{v_h, v_l\}$ with the larger $\beta(v)$;
13         Update $\hat{s}(u,v)$ and $\beta(v)$;
```

Assume that after several rounds of iteration, we have the configuration shown in Figure 3(b). Again, we compute UB_i, LB_i and then B_i for each candidate v_i. Now, the second smallest B_i (i.e. B_3) is below zero, and we are sure that $\{v_1, v_3\}$ is the top-2 nodes with the largest similarity, and the algorithm terminates.

Analysis. The efficiency of *Top-k-Identification* is bounded by the following lemma.

LEMMA 4. Top-k-Identification *returns the actual top-k nodes with at least* $1-\delta$ *probability and with expected time complexity* $\min\left(O(\frac{\theta}{\varepsilon_{min}{}^2}\log\frac{n}{\delta}), O(\sum\limits_{v \in V\setminus\{u\}} \frac{1}{\max\left(\frac{\Delta_v + \varepsilon_{min}}{2}, \varepsilon_{min}\right)^2}\log\frac{n}{\delta})\right)$, *where* $\theta = \frac{t_a}{t_o}$ *denotes the ratio between the time cost of one sample-all-arms operation (i.e.,* t_a*) and one sample-one-arm operation (i.e.,* t_o*).*

We now analyze the expected running time of *SimTab-Top-k*.

THEOREM 2. *The* SimTab-Top-k *algorithm answers the top-k SimRank queries with at least* $1-\delta$ *success probability, with asymptotically the same time complexity of* Top-k-Identification.

3.4.2 Optimizations

The *Top-k-Identification* algorithm uses $\sqrt{c}$-walk sampling to estimate the reward of an arm (i.e. the SimRank similarity of a pair of nodes), and always gives 0/1 estimation for each sample-one-arm operation. As a consequence, the empirical variance of the estimation is large, resulting in loose confidence bounds (see Equation 8). Thus, more samples are needed until the stopping condition is satisfied. We propose an optimization algorithm, denoted by *Adaptive-Top-k*. Firstly, we note that SimRank can be interpreted from the perspective of reverse Personalized PageRank,

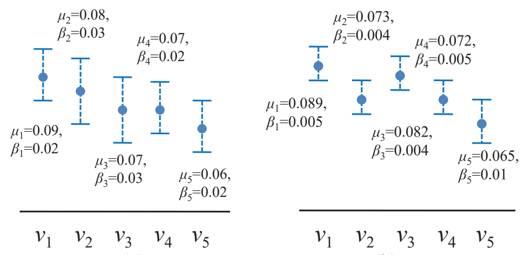


Figure 3: An example for the top-k query.

as demonstrated in Equation 6. Instead of using Monte Carlo sampling, the PPR values can also be computed in a deterministic way by forward push [5], which generates estimators with smaller variances but with higher computation cost. Secondly, we conduct the push operation in an adaptive way, i.e., the number of push operations for each candidate varies. Intuitively, for each candidate, we conduct as many forward pushes as possible until the asymptotical complexity of the push operation matches the cost of the walks that has been sampled. Therefore, more push operations are applied to a candidate node if it has been sampled extensively. Note that more push operations significantly tighten the confidence bound. This helps us to effectively distinguish the nodes with SimRank values close to the boundary (i.e., the k-th largest SimRank value or the given threshold), while saving the cost for most other candidates, which guarantees the efficiency of the whole algorithm.

The pseudo-code is demonstrated in Algorithm 5. For each level $l = [0, \infty)$, we first initialize $\hat{\pi}_l(v, w) = 0$ for each $v \in C \cup \{u\}$ and $w \in V$. We set $r_0(v, v) = 1$ for each $v \in C \cup \{u\}$, and $r_l(v, w) = 0$ for each $v \in C \cup \{u\}$ and $w \in V$ for $l \geq 1$. The level information must be recorded to guarantee the walks meet at the same step. Initially, no push operation is performed. For each candidate v, once the number of samples doubled, we conduct a few pushes based on a parameter r_{max}. Intuitively, r_{max} prevents those push operations from nodes with small residues to get a good tradeoff between algorithm efficiency and effectiveness. When we are to estimate the similarity of the query node u and a candidate v, we follow the idea of Equation 6. Combining with the property of forward push, we rewrite the Equation as follows.

$$s(u, v) = \frac{1}{(1-\sqrt{c})^2} \sum_{l=0}^{\infty} \sum_{w \in V} \tilde{\pi}_l(u, w) \tilde{\pi}_l(v, w) d(w), \qquad (10)$$

where $\tilde{\pi}_l(s, w) = \hat{\pi}_l(s, w) + \delta_l(s, w)$, $\hat{\pi}_l(s, w)$ is the reserve by forward push, and $\delta_l(s, w) = \sum_{v \in V} \sum_{i=0}^{l} r_i(s, v) \pi_{l-i}(v, w)$. First, we fetch all node w having non-zero estimated reserves from both u and v at each level l, and aggregate them following Equation 10. Next, for both u and v we sample a $\sqrt{c}$-walk to derive an unbiased estimation of $\delta_l(u, w)$ (resp. $\delta_l(v, w)$). Take v as the example, first a node y with $r_{l_2}(v, y)$ is sampled with probability $\frac{r_{l_2}(v,y)}{r_{sum}(v)}$, where $r_{sum}(v) = \sum_l \sum_{w \in V} r_l(v, w)$ denotes the sum of all probabilities not handled yet. Then we sample a $\sqrt{c}$-walk from y, which terminates at node y'. Let $l_2' = l_2 + |W(y)|$. If $\hat{\pi}_{l_2'}(u, y') \neq 0$, we add $\hat{\pi}_{l_2'}(u, y') r_{sum}(v) \hat{d}(y')$ to the estimation, and vice versa for node u. If the walk of u and v meet (i.e., $x' = y'$), we add $r_{sum}(u) r_{sum}(v) \hat{d}(x')$ to $\hat{s}(u, v)$. It can be proved that the estimator is unbiased for $s(u, v)$. For the estimation of $d(w)$, we sample two $\sqrt{c}$-walks from w, which gives a 0/1 estimation.

LEMMA 5. *The time and space complexity of the Adaptive-Top-k algorithm is asymptotically identical to those of the Top-k-Identification algorithm.*

Algorithm 5: Adaptive-Top-k Algorithm

```
Input: $G = (V, E)$; $u \in V$; $k$; candidate set
       $C = \{v, \hat{s}(u, v), \beta(v)\}$; failure probability $\delta$;
Output: $V_k$, the estimated top-$k$ nodes with largest SimRank
        values
$n_r(u) = \sum_{v \in C} n_r(v), n_r'(v) = n_r(v), \forall v \in C \cup \{u\}$;
for each $v \in C \cup \{u\}$ and $l = 0, 1, \ldots$ do
    if $l = 0$ then
        $r_l(v, v) = 1, r_l(v, w) = 0, \forall w \neq v$;
    else
        $\hat{\pi}_l(v, w) = 0, r_l(v, w) = 0, \forall w \in V$;
Let $H_l(v) = \{w | \hat{\pi}_l(v, w) > 0\}, R_l(v) = \{w | r_l(v, w) > 0\}$;
Replace Line 12 of Algorithm 4 into:
$\hat{s}(u, v) = 0$;
for $l = 0, 1, \ldots$ do
    for each $w \in H_l(u) \cap H_l(v)$ do
        Increase $\hat{s}(u, v)$ by $\hat{\pi}_l(u, w)\hat{\pi}_l(v, w)\hat{d}(w)$;
Sample an item $(x, r_{l_1}(u, x))$ from $\{R_0(u), R_1(u), \ldots\}$;
Sample an $\sqrt{c}$-walk $W(x)$ from $x$, which stops at node $x'$;
Let $l_1' = l_1 + |W(x)|$;
Sample an item $(y, r_{l_2}(v, y))$ from $\{R_0(v), R_1(v), \ldots\}$;
Sample an $\sqrt{c}$-walk $W(y)$ from $y$, which stops at node $y'$;
Let $l_2' = l_2 + |W(y)|$;
Let $r_{sum}(u) = \sum_{R_0(u), R_1(u), \ldots} \sum_{x \in V} r_l(u, x)$;
Let $r_{sum}(v) = \sum_{R_0(v), R_1(v), \ldots} \sum_{y \in V} r_l(v, y)$;
if $\hat{\pi}_{l_1'}(v, x') \neq 0$ then
    Increase $\hat{s}(u, v)$ by $r_{sum}(u)\hat{\pi}_{l_1'}(v, x')\hat{d}(x')$;
if $\hat{\pi}_{l_2'}(u, y') \neq 0$ then
    Increase $\hat{s}(u, v)$ by $\hat{\pi}_{l_2'}(u, y') r_{sum}(v)\hat{d}(y')$;
if $l_1' = l_2'$ and $x' = y'$ then
    Increase $\hat{s}(u, v)$ by $r_{sum}(u) r_{sum}(v)\hat{d}(x')$;
Use $\hat{s}(u, v)$ as an estimation of $s(u, v)$;
Insert after Line 13 of Algorithm 4:
Increase $n_r(v_h)$ and $n_r(v_l)$ by 1, respectively;
Increase $n_r(u)$ by 2;
for any $v \in \{v_h, v_l, u\}$ do
    if $n_r(v) = 2n_r'(v)$ then
        ForwardPush($G, v, \frac{1}{n_r(v)}$);
        $n_r'(v) = n_r(v)$;
ForwardPush($G, v, r_{max}$)
for $l = 0, 1, \ldots$ do
    for each item $(w, r_l(v, w)) \in R_l(v)$ do
        if $\frac{r_l(v,w)}{|I(w)|} > r_{max}$ then
            $\hat{\pi}_l(v, w) = \hat{\pi}_l(v, w) + (1 - \sqrt{c}) \cdot r_l(v, w)$;
            Update $\hat{\pi}_l(v, w)$ in $H_l(v)$;
            for each $x \in I(w)$ do
                $r_{l+1}(v, x) = r_{l+1}(v, x) + \frac{\sqrt{c}}{|I(w)|} \cdot r_l(v, w)$;
                Update $r_{l+1}(v, x)$ in $R_{l+1}(v)$;
            $r_l(v, w) = 0$;
```

4. THRESHOLDING QUERIES

In this section, we demonstrate how to adapt our arm sampling strategies and the techniques for confidence bound to the thresholding query, which returns all nodes with SimRank values larger than a threshold τ for a given query node u. Firstly, by tackling the problem from the perspective of MAB, a greedy arm sampling strategy [26] is adopted for the sample-one-arm operations[3]. Secondly, to enhance practical efficiency, we also employ the *sample-all-arms* strategy, which is implemented by the prefiltering algorithm and with a modification of the pruning rule. Our algorithm, denoted as *SimTab-Thres*, is shown in Algorithm 6. Similar to *SimTab-Top-k*, the algorithm contains a prefiltering phase followed by a refinement phase.

Algorithm 6: *SimTab-Thres*

```
Input: Directed graph $G = (V, E)$; $u \in V$; threshold $\tau$;
       failure probability $\delta$;
Output: $V_\tau$, the estimated node set with SimRank value no
        smaller than $\tau$
1  $V_\tau = \emptyset$;
2  $C$ = Prefiltering($G, u, \tau, \frac{\delta}{2}$);
3  Initialize minimum queue $Q = \emptyset$;
4  Let $n_r(v)$ be the number of samples of node $v$;
5  for each $v \in C$ do
6      Add $(v, \sqrt{n_r(v)} \cdot |\hat{s}(u,v) - \tau|)$ to $Q$;
7  while $Q \neq \emptyset$ do
8      $(v, p(v))$ = pop($Q$);
9      Sample a pair of $\sqrt{c}$-walks from $u$ and $v$ to update $\hat{s}(u,v)$
       and $\beta(v)$;
10     if $\hat{s}(u,v) - \beta(v) \geq \tau$ then
11         Add $v$ to $V_\tau$;
12     else if $\beta(v) \leq \varepsilon_{min}$ and $\hat{s}(u,v) \geq \tau$ then
13         Add $v$ to $V_\tau$;
14     else if $\beta(v) > \varepsilon_{min}$ and $\hat{s}(u,v) + \beta(v) \geq \tau$ then
15         Update $p(v)$, and put $(v, p(v))$ back into $Q$;
16 return $V_\tau$;
```

The prefiltering phase. In the prefiltering phase (Line 2), we filter out nodes that can not be in V_τ and get a candidate set C. With probability at least $1 - \frac{\delta}{2}$, C contains all nodes in V_τ. Each item in C is represented by a triple $(v, \hat{s}(u,v), \beta(v))$, where $\beta(v)$ denotes the confidence interval and can be updated later in the refinement phase according to the implementation described in Section 3. The prefiltering algorithm is analogous to Algorithm 2 but with the modified pruning rule as follows.

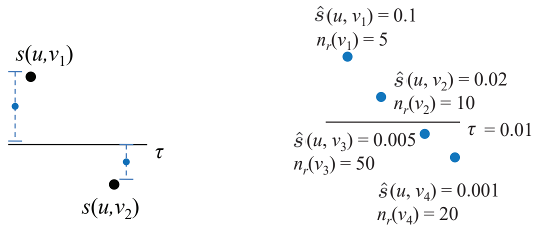
The pruning rule. For node v satisfying that $\hat{s}(u,v) - \beta(v) \geq \tau$, we directly add it to V_τ; node v satisfying that $\hat{s}(u,v) + \beta(v) < \tau$ will be discarded. If node v satisfies that $\hat{s}(u,v) - \beta(v) < \tau \leq \hat{s}(u,v) + \beta(v)$, we can not decide whether $v \in V_\tau$. Therefore, it is referred to as the candidate node.

The refinement phase. In the refinement phase, we identify the nodes with SimRank values above τ from C. Intuitively, this can be done by applying the Monte Carlo procedure for each candidate node. For node v, the procedure stops as long as the confidence bound $\beta(v)$ falls below $\frac{|s(u,v)-\tau|}{2}$ (see Figure 4(a)). Moreover, the time complexity asymptotically matches the hardness of the thresholding bandit problem. Nonetheless, we adopt a heuristic adaptive sampling strategy [26], which facilitates the optimization techniques and approximate version of the query. Details are referred to the full version of our paper [1].

[3]More precisely, in [26] the proposed method solves the thresholding bandit problem under the fixed budget setting, which minimizes the error of estimation given the number of samples. On the other hand, our problem belongs to the fixed confidence setting, i.e., minimizing the number of samples for a predefined failure probability.

The sampling strategy. At every step, we sample the node $v \in C$ with the minimum weight $p(v) = \sqrt{n_r(v)} \cdot |\hat{s}(u,v) - \tau|$.

Once we are sure that $v \in V_\tau$ (Lines 10-11) or the estimation is as accurate as of the ground truth (Lines 12-13), v is moved to V_τ or discarded based on the estimation value. Here, we use a very small error parameter ε_{min} (e.g. 10^{-6}) to handle nodes with $s(u,v) = \tau$. Otherwise, the sampling-based algorithm will not stop for these nodes. The setting of ε_{min} guarantees that our result is at least as good as the ground truth used in [24, 32].



(a) The complexity of *MC*. (b) A toy example for the sampling strategy.

Figure 4: An example for the thresholding query.

Runinng example for Algorithm 6. Consider the toy example in Figure 4(b), where we have four candidates $\{v_1, v_2, v_3, v_4\}$ for the thresholding query with $\tau = 0.01$. According to the sampling strategy, we compute $p(v_1) = \sqrt{5} \cdot 0.09 \approx 0.201$, $p(v_2) = \sqrt{10} \cdot 0.01 \approx 0.032$, $p(v_3) = \sqrt{50} \cdot 0.005 \approx 0.035$, and $p(v_4) = \sqrt{20} \cdot 0.009 \approx 0.040$. Therefore, v_2 will be chosen since it has the minimum weight. Note that the sampling strategy implies that a candidate v which is sampled few times whereas having a small gap between the estimated value and the threshold should be re-considered. Intuitively, since the gap between v_1, v_4 and τ is sufficiently large, no more sample is needed; meanwhile, a large number of samples have already given an accurate estimation for v_3.

We theoretically prove the correctness and complexity of the *SimTab-Thres* algorithm.

THEOREM 3. *With at least* $1 - \delta$ *probability,* SimTab-Thres *returns the true* V_τ*, i.e. the node set with SimRank value no smaller than the threshold* τ*, with expected time complexity* $\min\left(O(\frac{\theta}{\varepsilon_{min}{}^2} \log\frac{n}{\delta}), O(\sum_{v \in V \setminus \{u\}} \frac{1}{(\Delta_{\tau,v} + \varepsilon_{min})^2} \log\frac{n}{\delta})\right)$.

Optimizations. Recall that during the top-k identification phase of *SimTab-Top-k*, we employ an adaptive forward push to tighten the confidence bounds without additional cost. This technique can be integrated into the refinement phase of *SimTab-Thres*, since both queries apply the sample-one-arm strategy in which arms are sampled independently and repeatedly.

5. RELATED WORK

To the best of our knowledge, very few methods are directly developed for the top-k and thresholding SimRank queries. Therefore, we classify the related work into (1) algorithms for top-k queries [21], and (2) the single-source algorithms with the state-of-the-art empirical performance on these queries. We also include a brief discussion of other related work.

5.1 Algorithms for Top-k Queries

TopSim [21] is the only known method that exactly solves the top-k SimRank query. Given the query node u, *TopSim* firstly finds all w reachable from u with exact l steps following *in-edges*, where l starts from 1 and increases by 1 once all such w are found. For each w, it finds all v reachable from w in exact l steps following *out-edges*. Moreover, v must not be any node in path $w \rightsquigarrow u$. Then the probability of $w \rightsquigarrow u, w \rightsquigarrow v$ is aggregated into $\hat{s}(u,v)$, an estimation of $s(u,v)$. The algorithm stops when the gap between k-th and $(k+1)$-th largest score exceeds the heuristic upper bound of the score any node can gain with more than l steps, in fact, $(\frac{c}{\bar{d}})^{l+1}$, where c is the decay factor and $\bar{d}$ is the average degree of the graph. The authors also propose several optimization algorithms to improve the speed, some with trade of accuracy.

5.2 State-of-the-art Single-Source Queries

The recently proposed single-source algorithms [24,32] are state of the art for both top-k and thresholding queries. They all employ the *return all and postprocessing* paradigm, i.e., they first derive an approximate estimation for each node in the graph, followed by returning the set of nodes satisfying the query constraint.

5.2.1 The index-free algorithm

ProbeSim [24] is the state-of-the-art index-free algorithm that is able to compute single-source SimRank queries on large graphs. Given a query node u, *ProbeSim* samples n_r $\sqrt{c}$-walks from u following in-edges in the forward stage. For each walk $W(u) = (w_0 = u, w_1, \ldots)$, let w_l be the node at the l-th step. The algorithm then performs a *probe* procedure from each $w_l (l = 1, \ldots)$ in the backward stage, where the meeting probabilities for different w_i are aggregated to derive an unbiased estimation of $s(u,v)$. They provide both a deterministic and a randomized version of the probe procedure, which is essentially based on deterministic or randomized similar path enumeration.

5.2.2 Index-based algorithms

Most single-source algorithms [17,28,30,32] use index to improve the query performance. In general, they pre-compute a fraction of the results of similarity path enumeration or a large number of random walks during index construction, while the index can be reused in the query phase. *TSF* [28] constructs a set of *one-way graphs* as index via random in-neighbor sampling, and answers single-source queries by online random walk sampling and traversal on one-way graphs. On the contrary, *SLING* [30] implements the forward and backward stage as similarity path enumeration. It computes the *hitting probabilities* from each node v, which denotes the probability of an $\sqrt{c}$-walk from v passing w. Only non-negligible probabilities are indexed to reduce space costs. *READS* [17] proposes an indexing scheme along with the *SimRank-aware random walk* to enhance both theoretical and practical query efficiency of *MC*. As the state-of-the-art index-based approach, *PRSim* [32] improves the efficiency of the backward stage via a probability-guided search of similarity paths. To limit the index space, it only conducts backward search [4] from nodes with high reverse PageRank scores (i.e., PageRank following in-edges). In the query phase, given a $\sqrt{c}$-walk generated in the forward stage, if it stops at a hub node, the indexed estimation values are directly retrieved; otherwise a randomized searching procedure is invoked to estimate the SimRank values, which only incurs sub-linear cost on power-law graphs.

Although pre-computing partial intermediate results as the index benefits the query time performance, it also has several fatal drawbacks. Firstly, all known methods except [17,28] construct *static* index, which means the index can not be updated once the graph changes. Secondly, for *every* index-based method, the preprocessing phase (e.g., the size of the index) is determined by the error parameter ε, which is user-defined during the query phase. Therefore, these algorithms suffer from low flexibility.

5.3 Other Related Work

The algorithms for SimRank can be broadly divided into the iterative methods and the random walk based methods. [16] proposes the *Power Method*, an iterative method to compute the all-pair SimRank matrix S. It is based on the matrix formulation of SimRank [20]: $S = (cP^\top SP) \vee I$, where I is the identity matrix of compatible size, P is a *transition matrix* defined by the edges in G, and $\vee$ is the element-wise maximum operator. A bunch of follow-up work [25,31,35,37] improves its efficiency or accuracy; however, as all methods incur $O(n^2)$ space overheads, the cost is prohibitive for large-sized graphs. We also note a line of research [11,13,20,22,33,34,36] has been proposed with a modified definition of SimRank that makes it easier to compute: $S = cP^\top SP + (1-c) \cdot I$. However, [20,30] prove that this definition is rather different from the original SimRank.

6. EXPERIMENTS

This section experimentally evaluates our proposed algorithms against the state-of-the-art methods for top-k and thresholding SimRank queries.

Table 5: Datasets.

Dataset	Type	n	m
DBLP (DB)	undirected	5,425,963	17,298,033
LiveJournal (LJ)	directed	4,847,571	68,993,773
IT-2004 (IT)	directed	41,291,594	1,150,725,436
Friendster (FD)	directed	68,349,466	2,586,147,869

6.1 Experimental Setup

Datasets. We use four large graph datasets [2,3] with edge numbers varying from tens of millions to billions, as shown in Table 5.

Query generation. We consider two different strategies for query node generation. 1) *Uniformly at random*. Following all previous studies [17,24,28,30,32], for each dataset, we randomly generate 1,000 query nodes for top-k and thresholding queries, respectively. 2) *Stratified sampling*. Due to the definition of SimRank, a query node with higher in-degree tends to have more nodes with smaller similarities. We split the in-degrees into intervals $[1,10), [10,10^2), [10^2,10^3), [10^3,10^4)$ and $[10^4,\infty)$, and generate 100 random queries for each interval. For space constraints, we only report the empirical results under the uniformly-at-random setting, for that the conclusions of two different settings are similar.

Methods. We evaluate our top-k and thresholding algorithm with *ProbeSim* [24], the state-of-the-art index-free algorithm, and the state-of-the-art index-based algorithms, including *PRSim* [32] and *READS* [17]. We include *TopSim* [21] and *TSF* [28] as baselines. Since *PRSim* can be easily modified into an index-free algorithm by setting the number of indexed hubs as 0, we also take it into consideration, and denote it as *PRSim-IF* (for index-free). We also compare our methods to *Opt-LP*, the state-of-the-art all-pair SimRank algorithm [31]. We obtain the code of *ProbeSim*, *READS*, *PRSim*, and *Opt-LP* from the authors, and implement all other algorithms in C++ and compile the codes with the -O3 option. All experiments are conducted on a machine with a 2.6GHz CPU and 128GB memory. Following previous works [24, 30, 32], we set $c = 0.6$ through all experiments.

Parameters. For each method, we choose two parameter settings: the *typical* parameter setting following the original paper, and a *precise* parameter setting to achieve the most accurate estimation,

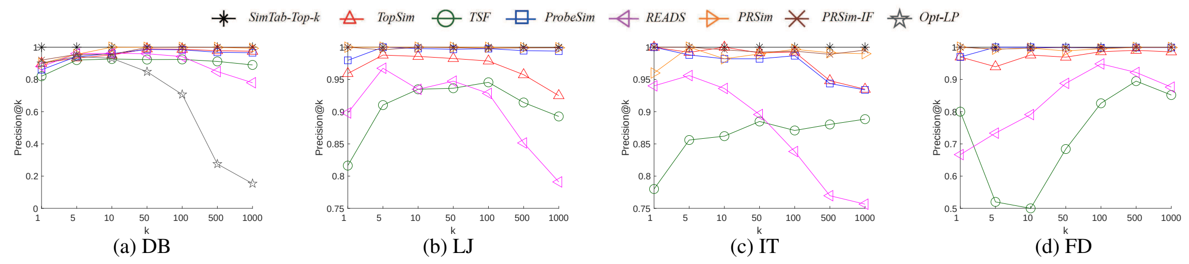


Figure 5: Precision@k for top-k queries, varying $k \in \{1, 5, 10, 50, 100, 500, 1000\}$

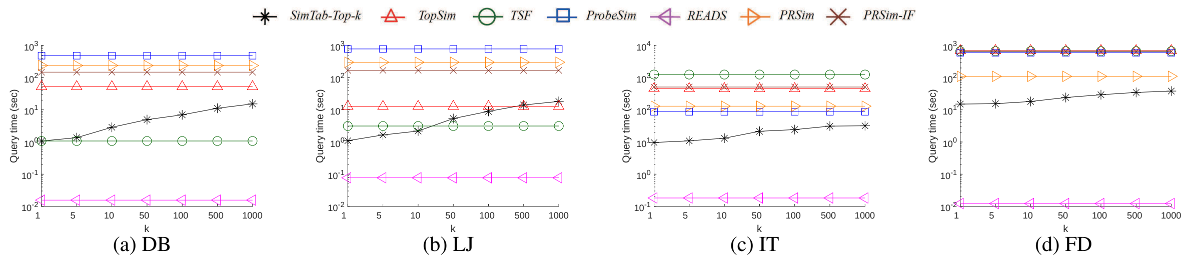


Figure 6: Query time for top-k queries, varying $k \in \{1, 5, 10, 50, 100, 500, 1000\}$

on condition that the method is not out-of-time[4]. *ProbeSim, PRSim, and PRSim-IF* all use an absolute error parameter ε for single-source queries. We set $\varepsilon \in \{0.1, 0.05, 0.01, 0.005, ...\}$, where $\varepsilon = 0.1$ is the typical setting for *ProbeSim* as in [24], and $\varepsilon = 0.01$ is the typical setting for *PRSim* and *PRSim-IF*. For *TopSim*, since the baseline algorithm is extremely slow and has inferior answer quality, we use *Prio-TopSim*, the optimized algorithm for evaluation. We vary T , the depth of the traversal from 3 to 6 with default value as 3, and H, the size of priority pool from $1,000$ to $4,000$ with default value as $1,000$. *TSF* has two parameters R_g and R_q, which denotes the number of indexed one-way graphs and the number of times each one-way graph is reused in the query stage, respectively. We set $(R_g, R_q) \in \{(100, 20), (300, 40), (600, 80), (900, 120)\}$, where $(100, 20)$ is the default value. When the size of the index exceeds the memory capacity, we use its external memory version, i.e., *EXT-TSF*, which may incur longer query time for disk I/O. For *READS*, we vary parameter r, the number of indexes (i.e. SA forests) constructed during preprocessing, and rq, the number of random walks generated for each index at query time. Specifically, we set $(r, rq) \in \{(100, 10), (500, 10), (500, 20), (1000, 20)\}$ while by default $(r, rq) = (100, 10)$. For our algorithms, we set $\varepsilon_{min} = 10^{-6}$. The failure probability is set to 0.0001 for all sampling-based algorithms. Since all baselines under the typical parameter settings give significantly inferior performance compared to their precise parameter settings, we omit them in Figure 5-8. More details can be referred to the full version of the paper [1].

We adopt the idea of *pooling* [24] to evaluate the relative effectiveness of different algorithms. We take *MC* as the ground truth for each candidate in the pool and guarantee that the estimation error is below 10^{-6} with confidence over 99.999%.

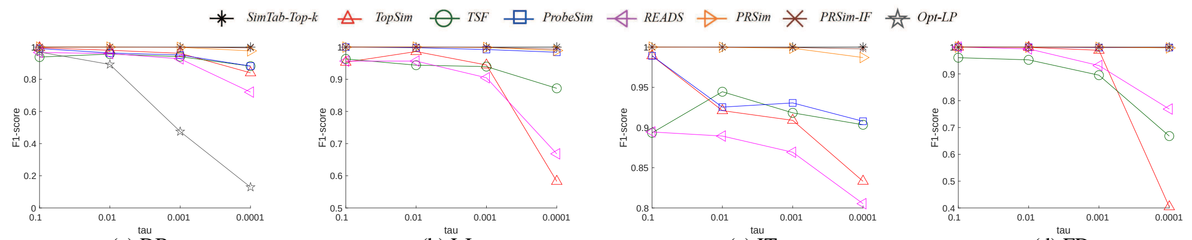
6.2 Evaluation of Top-k Queries

Metrics. We use Precision@k to evaluate the accuracy of top-k queries. Given a query node u, denote by V_k the set of top-k nodes with the largest SimRank values, and V'_k the estimated node set of a specific algorithm. The metric is defined as $Precision@k = \frac{|V_k \cap V'_k|}{k}$. In the evaluation, we vary k in $\{1, 5, 10, 50, 100, 500, 1000\}$. Intuitively, as k increases, the problem incurs higher computational cost because the top-k answers are harder to be distinguished from other nodes.

Our query results are illustrated in Figure 5 & 6. Generally speaking, all baselines achieve better performance for the medium values of k, while the precision decreases as k goes towards 1 or $1,000$. In particular, the two algorithms that adopt random walk sampling in both the forward and backward stage, i.e., *TSF* and *READS*, achieve the highest query speed with the help of index, but give significantly inferior query accuracy compared to other baselines. This is attributed to that the number of indices constructed are determined heuristically, partially because of the tremendous time and space overhead. Hence, the estimation error can not be guaranteed in practice. Methods with graph-traversal based searching strategies, including *TopSim*, *ProbeSim*, *PRSim*, and *PRSim-IF*, incur much higher query cost but achieve relatively satisfying accuracy. *TopSim* acquires surprisingly good performance in terms of Precision@k, since intuitively the most similar nodes locate close to the query node. Overall, *PRSim-IF* and *PRSim* are the two best baseline algorithms with a good tradeoff between query accuracy, computational cost, and index size (for *PRSim*). In comparison, the precision of *SimTab-Top-k* is always 1 in our experiments. This is achieved by our multi-armed bandit modeling of the problem, which enables us to treat the nodes adaptively in the sampling process. Meanwhile, our algorithm is orders of magnitude faster than those baselines with acceptable query accuracy, e.g, *PRSim-IF*. Note that the precision of these algorithms still have a gap from 1, and remain non-stable for different datasets and k. On the other hand, the answer quality can hardly be improved, because these algorithms cannot efficiently query with smaller parameters for the tremendous time and index cost. Lastly, the query time of *SimTab-Top-k* varies with k, which implies that our algorithm is adaptive to the hardness of the queries.

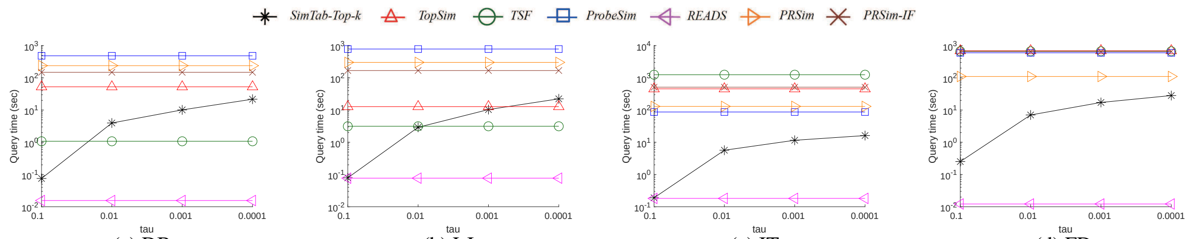
To demonstrate the uniqueness and hardness of the top-k query, we also compare with *Opt-LP*, the state-of-the-art all-pair SimRank algorithm, which is also the most scalable one for large graphs.

[4]We say an algorithm is out-of-time if the query time for some node is over one hour or the index construction time is beyond 10,000 seconds. This indicates that the algorithm is not suitable for dynamic graphs.



(a) DB (b) LJ (c) IT (d) FD

Figure 7: F1-score for thresholding queries, varying $\tau \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$



(a) DB (b) LJ (c) IT (d) FD

Figure 8: Query time for thresholding queries, varying $\tau \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$

Following [31], we set the absolute error parameter ε to 0.01. Unfortunately, even under such a loose parameter setting, *Opt-LP* can only compute the SimRank similarities on DB, the smallest dataset, and fails on three other datasets due to out of memory. Besides, the query accuracy in terms of Precision@k is inferior to all compared single-source algorithms. This indicates that the top-k SimRank query cannot be easily answered via direct SimRank computation and should be specially considered.

6.3 Evaluation of Thresholding Queries

Metrics. For the thresholding query, we vary $\tau \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$. We use precision, recall and F1-score as the evaluation metrics. Specifically, let S be the set of exact answers for thresholding queries, and S' the node set returned by some specific algorithm. We define $Precision = \frac{|S \cap S'|}{|S'|}$, $Recall = \frac{|S \cap S'|}{|S|}$, and $F1-score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$.

The query results are demonstrated in Figure 7 & 8. For space constraints, we only show F1-Score in Figure 7. The following conclusions are easily derived. First, the query is harder for smaller τ. Since most nodes have low SimRank values w.r.t. the query node, more nodes need to be estimated precisely for smaller τ. Consequently, the F1-score of baseline algorithms decreases as τ shrinks. Second, at the cost of longer query time, *PRSim-IF* achieves the best performance among all index-free baselines with F1-score close to 1, while *PRSim* is the best index-based algorithm. As Figure 7 shows, *TopSim* is not suitable for thresholding queries especially for small τ, because a large fraction of the answer is missed due to the truncated search and the heuristic pruning rule. Similar to the top-k query, the answer quality of *TSF* and *READS* is significantly inferior to those algorithms with graph-traversal based strategies, such as *PRSim-IF* and *PRSim*. Therefore, they should not be recommended to answer these queries. Again, our *SimTab-Thres* algorithm enables to answer all queries exactly, with query efficiency significantly outperforming any baseline that achieves acceptable answer quality.

7. CONCLUSIONS

In this paper, we propose algorithms to improve both the efficiency and the effectiveness of the state-of-the-art methods over top-k and thresholding SimRank queries. Specifically, we integrate several techniques to tighten the confidence bounds of SimRank estimation, including the empirical Bernstein inequality, variance reduction tricks, and careful algorithm design. Moreover, we model the top-k and thresholding SimRank queries as the multi-armed bandits problems, so that the algorithmic complexity is determined by the hardness of the query instance. By proposing novel arm sampling strategies, we are able to significantly enhance the practical efficiency of the algorithms. We conduct extensive experiments for top-k and thresholding queries on large-scale graphs, and the results indicate that our algorithms are the only acceptable methods to achieve stable and satisfying performance in terms of answer quality and with reasonable query efficiency.

8. ACKNOWLEDGMENTS

This work was supported by The National Key Research and Development Program of China under grant 2018YFB1003504, NSFC (No. 61932001), and Peking University Medicine Seed Fund for Interdisciplinary Research supported by the Fundamental Research Funds for the Central Universities (No. BMU2018MI015). It was also supported by NSFC (No. 61961130390, No. 61972401, and No. 61832017), by Beijing Outstanding Young Scientist Program NO. BJJWZYJH012019100020098, and by the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China under Grant 18XNLG21. This work was also supported by Beijing Academy of Artificial Intelligence (BAAI). Zhewei Wei is the corresponding author. This work was partially done at Beijing Key Laboratory of Big Data Management and Analysis Methods, and at Key Laboratory of Data Engineering and Knowledge Engineering, MOE, Renmin University of China.

9. REFERENCES

[1] `https://github.com/dokirabbithole/SimTab_Technical_Report/blob/master/SimTab_TR.pdf`.

[2] `http://snap.stanford.edu/data/index.html`.

[3] `http://konect.uni-koblenz.de/`.

[4] R. Andersen, C. Borgs, J. Chayes, J. Hopcraft, V. S. Mirrokni, and S.-H. Teng. Local computation of pagerank contributions. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 150–165. Springer, 2007.

[5] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486. IEEE, 2006.

[6] P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *J. Mach. Learn. Res.*, 3:397–422, 2002.

[7] S. Chen, T. Lin, I. King, M. R. Lyu, and W. Chen. Combinatorial pure exploration of multi-armed bandits. In *Advances in Neural Information Processing Systems*, pages 379–387, 2014.

[8] F. R. K. Chung and L. Lu. Concentration inequalities and martingale inequalities: A survey. *Internet Mathematics*, 3(1):79–127, 2006.

[9] D. Fogaras and B. Rácz. Scaling link-based similarity search. In *WWW*, pages 641–650, 2005.

[10] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2(3):333–358, 2005.

[11] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, and M. Onizuka. Efficient search algorithm for simrank. In *ICDE*, pages 589–600, 2013.

[12] V. Gabillon, M. Ghavamzadeh, and A. Lazaric. Best arm identification: A unified approach to fixed budget and fixed confidence. In *Advances in Neural Information Processing Systems*, pages 3212–3220, 2012.

[13] G. He, H. Feng, C. Li, and H. Chen. Parallel simrank computation on large graphs with iterative aggregation. In *KDD*, pages 543–552, 2010.

[14] W. Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[15] K. Jamieson, M. Malloy, R. Nowak, and S. Bubeck. lilucb: An optimal exploration algorithm for multi-armed bandits. In *Conference on Learning Theory*, pages 423–439, 2014.

[16] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *SIGKDD*, pages 538–543, 2002.

[17] M. Jiang, A. W.-C. Fu, and R. C.-W. Wong. Reads: a random walk approach for efficient and accurate dynamic simrank. *PVLDB*, 10(9):937–948, 2017.

[18] R. Jin, V. E. Lee, and H. Hong. Axiomatic ranking of network role similarity. In *KDD*, pages 922–930, 2011.

[19] S. Kalyanakrishnan, A. Tewari, P. Auer, and P. Stone. Pac subset selection in stochastic multi-armed bandits. In *ICML*, volume 12, pages 655–662, 2012.

[20] M. Kusumoto, T. Maehara, and K. Kawarabayashi. Scalable similarity search for simrank. In *SIGMOD*, pages 325–336, 2014.

[21] P. Lee, L. V. Lakshmanan, and J. X. Yu. On top-k structural similarity search. In *2012 IEEE 28th International Conference on Data Engineering*, pages 774–785. IEEE, 2012.

[22] C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, and T. Wu. Fast computation of simrank for static and dynamic information networks. In *EDBT*, pages 465–476, 2010.

[23] D. Liben-Nowell and J. M. Kleinberg. The link-prediction problem for social networks. *JASIST*, 58(7):1019–1031, 2007.

[24] Y. Liu, B. Zheng, X. He, Z. Wei, X. Xiao, K. Zheng, and J. Lu. Probesim: scalable single-source and top-k simrank computations on dynamic graphs. *arXiv preprint arXiv:1709.06955*, 2017.

[25] D. Lizorkin, P. Velikhov, M. N. Grinev, and D. Turdakov. Accuracy estimate and optimization techniques for simrank computation. *VLDB J.*, 19(1):45–66, 2010.

[26] A. Locatelli, M. Gutzeit, and A. Carpentier. An optimal algorithm for the thresholding bandit problem. *arXiv preprint arXiv:1605.08671*, 2016.

[27] V. Mnih, C. Szepesvári, and J.-Y. Audibert. Empirical bernstein stopping. In *Proceedings of the 25th international conference on Machine learning*, pages 672–679, 2008.

[28] Y. Shao, B. Cui, L. Chen, M. Liu, and X. Xie. An efficient similarity search framework for simrank over large dynamic graphs. *PVLDB*, 8(8):838–849, 2015.

[29] N. Spirin and J. Han. Survey on web spam detection: principles and algorithms. *SIGKDD Explorations*, 13(2):50–64, 2011.

[30] B. Tian and X. Xiao. Sling: A near-optimal index structure for simrank. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1859–1874, 2016.

[31] Y. Wang, X. Lian, and L. Chen. Efficient simrank tracking in dynamic graphs. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 545–556. IEEE Computer Society, 2018.

[32] 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 *Proceedings of the 2019 International Conference on Management of Data*, pages 1042–1059, 2019.

[33] W. Yu, X. Lin, and W. Zhang. Fast incremental simrank on link-evolving graphs. In *ICDE*, pages 304–315, 2014.

[34] W. Yu, X. Lin, W. Zhang, L. Chang, and J. Pei. More is simpler: Effectively and efficiently assessing node-pair similarities based on hyperlinks. *PVLDB*, 7(1):13–24, 2013.

[35] W. Yu and J. McCann. Gauging correct relative rankings for similarity search. In *CIKM*, pages 1791–1794, 2015.

[36] W. Yu and J. A. McCann. Efficient partial-pairs simrank search for large networks. *PVLDB*, 8(5):569–580, 2015.

[37] W. Yu, W. Zhang, X. Lin, Q. Zhang, and J. Le. A space and time efficient algorithm for simrank computation. *World Wide Web*, 15(3):327–353, 2012.