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

Yu Liu[1], Lei Zou[1], Qian Ge[1], Zhewei Wei[2]
[1]Wangxuan Institute of Computer Technology, Peking University, China
[2]School of Information, Renmin University of China, 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 an unified manner. First, we integrate several techniques with random walk sampling to tighten the confidence bound of SimRank estimation, which enhance 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 the state of the art, whereas the algorithm complexity closely matches the hardness of the problem. We further propose novel sampling strategy specially tailored for node similarity queries to dramatically improve the practical query efficiency of MAB 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.

1. INTRODUCTION

SimRank [15] is a widely adopted measure of the similarities of graph nodes, with numerous applications such as web mining [17], social network analysis [23], and spam detection [30]. 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:

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

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 [15, 25]. Due to the recursive definition of SimRank, its exact values can not be computed in limited time. The result of the *Power Method* [15], which is based on matrix iteration, is considered as the *de facto* ground truth as long as the incurred estimation error ε_{min} for any pair of nodes is sufficiently small, e.g., $\varepsilon_{min} = 10^{-10}$ [24, 31, 34].

A plethora of techniques have been proposed for the efficient computation of SimRank. Since computing all-pair SimRank incurs $\Omega(n^2)$ time and space cost, which is excessive 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. Given a error parameter ε and a failure probability δ, we say the query result achieves *absolute error guarantee*, 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 [17, 23, 30], 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, and let v_i be the node in G whose SimRank similarity to u is the i-th largest, $i = 1, \ldots, k$. A top-k SimRank query returns the set of k nodes $V_k = \{v_1, \ldots, v_k\}$, with at least $1 - \delta$ probability for any $k \in [1, n)$.*

DEFINITION 2 (THRESHOLDING QUERY). *Given a node u in G, a real number $\tau \in [0, 1]$, and a failure probability δ, a thresholding SimRank query returns a set of nodes V_τ, such that with $1-\delta$ probability, $V_\tau = \{v | s(u, v) \geq \tau, v \in V\}$.*

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 [20], which is specially designed for top-k queries. TopSim estimates similarity values by forward (i.e., following in-edges) and then backward (i.e.,

[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, 31, 34].

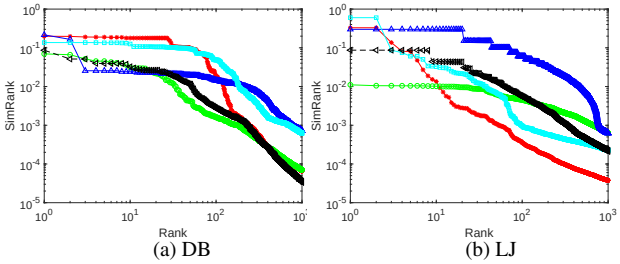
Proceedings of the VLDB Endowment, Vol. 11, No. 1

DOI: https://doi.org/10.14778/3136610.3136612

Table 1: A toy example, with query node u_1 and u_2 ($\varepsilon = 0.05$)

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



(a) DB (b) LJ

Figure 1: The distribution of top-1000 largest SimRank values for 5 random query nodes. For each SimRank value, the estimation error is below 10^{-6}.

following out-edges) traversals level by level from the query node. A heuristic stopping rule is used to speculate if the k-th largest estimated SimRank value is larger than the upper bound of the $k+1$-th to the $|V|$-th largest ones, which, however, to a large extent sacrifices the query accuracy. On the other hand, algorithms for single-source queries [16, 24, 29, 31, 34] 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 return the nodes with top-k largest SimRank values. Among them, the algorithms [24, 31, 34] with absolute error guarantee achieves the 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 are index-based, which precomputes 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 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 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.

In this paper, we improve both the query efficiency and effectiveness of top-k and thresholding 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 employ several techniques to tighten the confidence bound in SimRank estimation, including the empirical Bernstein inequality [28], 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 both theoretical and practical algorithm efficiency. Moreover, we model the top-k and thresholding queries from the perspective of the Multi-Armed Bandit (MAB) problems. By integrating the MAB algorithms, our algorithm treats each node differently by their SimRank values in the sampling procedure, coupled with the corresponding pruning rules. For our proposed method, the query complexity is dependent on the hardness of query instance, and closely matchs 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 similarities for different query nodes vary dramatically (shown in Figure 1). This phenomenon produces great influence on the problem complexity as the query parameter varies. For example, it is easier to answer top-1 queries than top-1000 queries, because the gap between the largest similarity value and values of other nodes is significantly larger than that of the 1000-th largest one (See Figure 1.). To improve the practical efficiency, we propose novel sampling strategy specially tailored for SimRank queries, for that the MAB algorithms have severe scalability problems on large-sized graphs. Several optimization techniques are also integrated to our algorithm design, which dramatically improves the efficiency and effectiveness. 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 [15] which is taken as the ground truth [24, 31]. Lastly, our algorithms are index-free, which can be naturally applied to dynamic graphs. Table 2 compares our proposed algorithms with the state of the art.

Finally, we conduct extensive experiments to evaluate our algorithms against the 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). Specifically, we adopt Precision@k to evaluate the quality of top-k query results, while for thresholding queries we measure the precision, recall, and the F1-score. We also empirically demonstrate that the 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, and are the only acceptable ones achieving query accuracy guarantee and satisfying answer quality.

2. PRELIMINARIES

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

2.1 SimRank with Random Walks

As indicated in [15], 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 that follows a randomly selected incoming edge at each step. Let t be 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 [16, 32] demonstrate that Equation 2 can be simplified by considering the probabilistic stop at each step. For example, the $\sqrt{c}$*-walk* [32] is defined as follows.

[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).

Method	Query Complexity	Index Cost (Space, Time)	Theoretical Guarantee	Empirical Performance
TopSim [20]	$O(\bar{d}^{2l})$	N/A	N/A	Poor
TSF [29]	$O(R_g R_q T^2)$	$O(R_g \lvert V\rvert)$	N/A	Non-stable
SLING [31]	$O(\frac{n}{\varepsilon})$ 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 [16]	$O(\frac{n}{\varepsilon^2}\log\frac{n}{\delta})$	$O(\frac{n}{\varepsilon^2}\log\frac{n}{\delta})$	Absolute error	Non-stable
PRSim [34]	$O(\frac{n}{\varepsilon^2}\log\frac{n}{\delta}\cdot\sum_{w\in V}\pi(w)^2)$	$O(\min(\frac{n}{\varepsilon},m))$, $O(\frac{m}{\varepsilon})$	Absolute error	Non-stable
SimTab-Top-k	$O(H_k\log\frac{n}{\delta})$	N/A	Precision@k	Good and stable
SimTab-Thres	$O(H_\tau\log\frac{n}{\delta})$	N/A	Precision and recall	Good and stable

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 $s(u, v)$
$W(u)$	An $\sqrt{c}$-walk from a node u
c	Decay factor in the definition of SimRank
ε, δ	Additive error parameter and failure probability
$\beta(v)$	Confidence bound of the estimated similarity of node v, i.e., with high probability $\hat{s}(u, v) - \beta(v) \le s(u, v) \le \hat{s}(u, v) + \beta(v)$

DEFINITION 3 ($\sqrt{c}$-WALK). *Given a node u in G, an $\sqrt{c}$-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 exist some l such that $w_l = w'_l$. According to [32],

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

Based on this random walk interpretation of SimRank, the Monte Carlo approach [8, 16, 32] 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 [16, 32].

LEMMA 1 (CHERNOFF-HOEFFDING INEQUALITY [13]). *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 $\bar{X} = \sum_{i=1}^{n_r} X_i$. Then*

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

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., $\Pr[|\bar{X} - s(u, v)| \ge \varepsilon] \le \delta$, we have $\beta_{n_r} = |\bar{X} - s(u, v)| \ge \sqrt{\frac{1}{2n_r}\log\frac{2}{\delta}}$. We refer to β_{n_r} as the *confidence bound*, 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 $n_r \ge \frac{1}{2\varepsilon^2}\log\frac{2}{\delta}$, with at least $1-\delta$ probability we have $\left|\frac{n_{r,meet}}{n_r} - s(u, v)\right| \le \varepsilon$. 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 size of the vertex set of 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 [16, 20, 24, 29, 31, 34] 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* [20] 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 phase 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 Relation between SimRank and Personalized PageRank

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

Method	The forward stage	The backward stage
TopSim [20]	similarity path enumeration	similarity path enumeration
TSF [29]	random walk sampling	similarity path enumeration (on the indexed one-way graph)
SLING [31]	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 [16]	(SimRank-aware) random walk sampling	(SimRank-aware) random walk sampling and indexing
PRSim [34]	$\sqrt{c}$-walk sampling	deterministic or randomized path enumeration, e.g., *Variance Bounded Backward Walk*

Inspired by [32], *PRSim* [34] 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:

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

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 [32]. 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 and backward push [3, 4].

Forward push [4] and backward push [3]. The forward push [4] is proposed to deterministically compute the Personalized PageRank. 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 our neighbors. This process can be formulated by the following Equation: $\hat{\pi}_f(s, v) = \hat{\pi}_f(s, v) + (1 - \sqrt{c})r_f(s, v), r_f(s, u) = r_f(s, u) + \frac{\sqrt{c}}{|O(v)|} r_f(s, v), \forall u \in O(v)$. As more push operations are conducted, the residues are transferred into the reserves, resulting in more accurate estimation of $\pi(s, t)$. The following Equation [4] holds in any step of forward push: $\pi(s, t) = \hat{\pi}_f(s, t) + \sum_{v \in V} r_f(s, v)\pi(v, t)$.

Instead of pushing the probabilities starting from the source node s, backward push [3] estimates PPR by reversely pushing the probabilities from t. The reserve $\hat{\pi}_b(v, t)$ and residue $r_b(v, t)$ can be defined analogous to the forward push. Meanwhile, we have the following Equation [3] during the push procedure: $\pi(s, t) = \hat{\pi}_b(s, t) + \sum_{v \in V} \pi(s, v)r_b(v, t)$. We omit the subscripts f and b in the following when the context is clear.

2.4 The Multi-Armed Bandits Problem

The multi-armed bandits (MAB) problems have been extensively studied in the field of theoretical computer science. Specifically, it considers an arbitrary instance of an n-armed bandit ($n \geq 2$). Each sample (or "pull") of arm a yields a reward generated randomly from a fixed distribution with mean $p_a \in [0, 1]$. Indeed each bandit instance is completely determined by the distributions corresponding to its arms. The random variables generating rewards for the arms are mutually independent. Among the well-studied MAB problems, the top-k arm identification [10, 14, 18] is arguably the one that draws the most attention.

DEFINITION 4 (THE TOP-k ARM IDENTIFICATION PROBLEM [5]). *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.*

In this paper, we also demonstrate that the thresholding SimRank queries can be modeled as the following MAB problem.

DEFINITION 5 (THE THRESHOLDING BANDITS PROBLEM [5, 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. We define the sampling complexity of the top-k problem as follow [18].

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

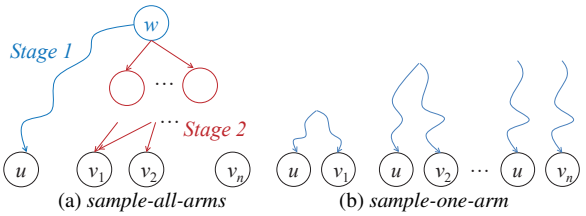
Intuitively, Δ_i characterizes the hardness to differentiate p_i from the actual top-k results. It has been proven in [18] that the sampling complexity of the top-k arm identification problem is at least $H_k = \sum_{i=1}^{n} \frac{1}{\Delta_i{}^2}$. Similarly, the sample complexity of the thresholding bandits problem can be defined as $H_\tau = \sum_{i=1}^{n} \frac{1}{\Delta_{\tau,i}{}^2}$ [5, 26], where $\Delta_{\tau,i} = |p_i - \tau|$ is the gap between the reward of the i-th arm and the given threshold.

3. TOP-K QUERIES

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

According to the random walk interpretation of SimRank, for each node pair (u, v), the SimRank similarity can be computed without relying on the similarities of other nodes, e.g., following the Monte Carlo approach. This motivates us to model the top-k query as the Multi-Armed Bandits (MAB) problem [10, 18]. To be precise, given a query node u, for all nodes in $V \backslash \{u\}$, we construct $n - 1$ arms with mean $s(u, v_1), \ldots, s(u, v_{n-1})$, where v_i is the node with the i-th largest SimRank values w.r.t u. Of course, the mean of each arm is not known to the algorithm and can only be estimated by sampling. For each v_i, a "pull" of the arm is implemented by sampling a pair of $\sqrt{c}$-walks from u and v_i respectively, which gives an unbiased estimation of $s(u, v_i)$ (Figure 2(b)). Note that the result of each sampling is either 0 or 1. To this end, designing an efficient top-k algorithm is equivalent to finding top-k arms with largest rewards by using as few samples as possible. As an example, for the Monte Carlo method, which falls into this paradigm, each node in $V \backslash \{u\}$ is sampled identical times. A better solution is to adopt some top-k arm sampling strategy [10, 14, 18] that achieves optimal theoretical complexity by treating arms differently. Since each arm (i.e., node pair) is independently sampled, we refer to this sampling strategy as *sample-one-arm*.

Unfortunately, due to the large amount of arms (i.e., nodes), any known MAB algorithm even with theoretically optimal performance may suffer from inferior practical efficiency. Since each arm needs to be sampled at least once, the cost is unacceptable for large-sized graphs. In fact, as our empirical analysis shows, even the state-of-the-art top-k and thresholding MAB algorithms can not finish in a reasonable time on a graph with million-sized nodes.



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

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

On the other hand, we notice that there exists another sampling strategy specially tailored for SimRank queries. Recall that a few existing solutions [24, 34] adopt random walk sampling in the forward stage of the random walk computation scheme (See Section 5). For each sampled walk, the backward stage computes an estimation of SimRank similarities for *all* nodes. Take *ProbeSim* [24] as an example, each invocation of *RandomizedProbe* gives an 0/1 estimation of $s(u, v)$ for all $v \in V \backslash \{u\}$ with $O(n)$ expected computation cost. They also show that the estimation is unbiased (Theorem 1 of [24]). Due to the power law distribution of real-world graphs, the SimRank similarities are highly skewed in most cases. As a consequence, these algorithms achieves much superior practical efficiency than applying sample-one-arm strategy for all nodes (i.e. *MC*), because only a fraction of nodes can be reached during the backward searching stage. We model this forward and backward procedure as the *sample-all-arms* strategy, which is only applicable for SimRank similarity search on graphs, thanks to its random walk interpretation.

Now we have a sample-one-arm strategy that achieves optimal theoretical efficiency, and a sample-all-arm strategy that performs well in practice, we will show how to combine them to get the best of both worlds.

3.2 Algorithm Overview

In this section, 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 economic, 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 an identical complexity to the top-k bandits algorithm asymptotically, and 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

Algorithm 1: *SimTab-Top-k*

Input: Directed graph $G = (V, E)$; $u \in V$; $k \in [1, n)$; failure probability δ

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;

which finds top-k nodes among the candidates, again with at most $\frac{\delta}{2}$ failure probability. Hence, with high probability, *SimTab-Top-k* can return 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. Intuitively, we want to prune as many nodes as possible through approximate single-source query and with low cost. Therefore, we adopt an iterative process to find the appropriate error parameter. Specifically, at the first round we make a guess of the sample error parameter $\varepsilon_1 = \frac{1}{2}$, and set the failure probability of each iteration (denoted as δ') to be $\frac{\delta}{\lceil \log_2 \frac{2}{\varepsilon_{min}} \rceil}$, where ε_{min} is a very small error parameter (e.g. 10^{-6}). We halve the error parameter in each iteration, and guarantee that the iteration takes no more than $\lceil \log_2 \frac{2}{\varepsilon_{min}} \rceil$ rounds, thus by the union bound, the single-source estimation fails with probability less than δ. Note that if the error parameter decreases to ε_{min} (Lines 9-11), the result of the single-source query will be regarded as ground truth. For each iteration i, we invoke the *Single-source* algorithm (described later), which gives an unbiased estimation $\hat{s}_i(u, v)$ for each v with additive error ε_i (Line 3). The top-k nodes with largest empirical means are added to the candidate set (Line 4). Next, we find the node v' with smallest lower confidence bound, i.e., $\hat{s}_i(u, v) - \beta_i(v)$, where $\beta_i(v)$ denotes the confidence bound of the estimation during the i-th iteration. Then we check for each $v \in V \backslash (\{u\} \cup C)$ if it can be pruned from the top-k answers. Specifically, we have the following pruning rule, of which the correctness can be easily derived.

The pruning rule. If $\hat{s}_i(u, v') - \beta_i(v') + \varepsilon_{min} > \hat{s}_i(u, v) + \beta_i(v)$ 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 bounds, we use the empirical Bernstein inequality [6]. Intuitively, it states that if the empirical variance is small, then the confidence bound is reversely related with the number of samples t. This bound is significantly tighter than that of the Chernoff-Hoeffding inequality, which is in reverse proportional to $\sqrt{t}$.

LEMMA 2 (EMPIRICAL BERNSTEIN INEQUALITY [6]). *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,*

$$|\bar{x}_t - \mu| \leq \bar{\sigma}_t \sqrt{\frac{2 \ln 3/\delta}{t}} + \frac{3R \ln 3/\delta}{t},$$

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

Hence, the confidence bound is set to $\beta_i(v) = \bar{\sigma}_i(v) \sqrt{\frac{2 \ln 3/\delta'}{n_{r_i}}} + \frac{3 \ln 3/\delta'}{n_{r_i}}$, where n_{r_i} is the number of sampled $\sqrt{c}$-walks during the

Algorithm 2: Prefiltering

Input: $G = (V, E)$; $u \in V$; k; failure probability δ;
Output: Candidate node set C

1 $\varepsilon_1 = \frac{1}{2}, i = 1, \delta' = \frac{\delta}{2\lceil \log_2 \frac{2}{\varepsilon_{min}} \rceil}$;
2 **while** *true* **do**
3 $\{(v, \hat{s}_i(u, v), \beta_i(v)) | v \in V \backslash \{u\}\}$ = Single-source($G, u, \varepsilon_i, \delta'$);
4 Let $C = \{v_1, \ldots, v_k\}$ be the nodes with top-k empirical means;
5 Let $v' = \text{argmin}_v\{\hat{s}_i(u, v) - \beta_i(v)\}, v \in C$;
6 **for** *each* $v \in V \backslash (\{u\} \cup C)$ **do**
7 **if** $\hat{s}_i(u, v') - \beta_i(v') + \varepsilon_{min} \leq \hat{s}_i(u, v) + \beta_i(v)$ **then**
8 $C = C \cup \{v\}$;
9 **if** $\varepsilon_i \leq \frac{\varepsilon_{min}}{2}$ **then**
10 Sort C according to the empirical means;
11 **return** the first k nodes in C, and skip the racing phase;
12 **else if** $|C| \leq t_a/t_o$ **then**
13 **return** C;
14 **else**
15 $\varepsilon_{i+1} = \varepsilon_i/2$;
16 i++;
17 `Single-source` (*G, u, ε, δ*)
18 Initialize $\hat{s}(u, v) = 0$ for each $v \in V \backslash (\{u\}$;
19 $n_r = \frac{n \log n}{\varepsilon^2}$;
20 **for** $i = 1$ *to* n_r **do**
21 Sample an $\sqrt{c}$-walk $W(u) = (w_0 = u, \ldots, w_l)$ from u;
22 Sample two independent $\sqrt{c}$-walks $W_1(w_l)$ and $W_2(w_l)$ from w_l;
23 **if** $W_1(w_l)$ *and* $W_2(w_l)$ *do not meet* **then**
24 $\{Score(u, v), \forall v \in V \backslash \{u\}\}$ = BackwardProbe(G, u, w, l, ε);
25 $\hat{s}(u, v) = \hat{s}(u, v) + \frac{Score(u,v)}{n_r}$;
26 **return** $\hat{s}(u, v)$;
27 `BackwardProbe` (G, u, w, l, ε)
28 Initialize $r_j(v, w) = 0, \hat{\pi}_j(v, w) = 0, Score_j(u, v) = 0$, and queue $Q_j = \emptyset$, hash set $H_j = \emptyset, R_j = \emptyset$ for $j \in [1, l]$;
29 $r_0(w, w) = 1, r_{max} = \varepsilon, numVisit = 0, Q_0 = \{w\}$;
30 **for** $j = 0$ *to* $l - 1$ **do**
31 **while** $Q_j \neq \emptyset$ **do**
32 x =pop(Q_j);
33 **if** $r_j(x, w) > r_{max}$ *and* $numVisit + |O(x)| < \alpha n$ **then**
34 **for** *each* $y \in O(x)$ **do**
35 $r_{j+1}(y, w) = r_{j+1}(y) + \sqrt{c} \cdot \frac{r_j(x,w)}{|I(y)|}$;
36 Add y to Q_{j+1};
37 $\hat{\pi}_j(x, w) = \hat{\pi}_j(x, w) + (1 - \sqrt{c}) \cdot r_j(x, w)$;
38 Add $(x, \hat{\pi}_j(x, w))$ to H_j;
39 $r_j(x, w) = 0, numVisit = numVisit + |O(x)|$;
40 **else**
41 Add $(x, r_j(x, w))$ to R_j;
42 **for** $j = 0$ *to* $l - 1$ **do**
43 **if** $\sum_{x \in R_j} |O(x)| \leq n$ **then**
44 $U = \cup_{x \in H_j} O(x)$;
45 **else**
46 $U = V \backslash \{u\}$;
47 **for** *each* $y \in U \cup H_{j+1}$ **do**
48 $Score_{j+1}(u, y) = \hat{\pi}_{j+1}(y, w)$;
49 Uniformly sample an edge (x, y) from $I(y)$;
50 **if** $x \in H_j$ **then**
51 Increase $Score_{j+1}(u, y)$ by $r_j(u, x)$ with probability $\sqrt{c}$;
52 **return** $\{(y, Score_l(u, y)) | y \in V\}$;

i-th iteration, and $\bar{\sigma}_i(v)$ denotes the empirical standard variance over the n_{r_i} estimations of $s(u, v)$.

Once the number of candidates is below a threshold such that apply single pair estimation for every arm independently incurs lower cost, we terminate the prefiltering phase and return C. (The stopping condition is shown later.) Otherwise, we half the value of the error parameter and proceed to the next iteration.

The *Single-source* algorithm. Now we turn to the single-source algorithm which is invoked for each iteration (Lines 17-26). Our algorithm follows the state-of-the-art forward and backward random walk scheme, and adopts the interpretation of SimRank by Equation 6. The algorithm repeatedly generates $n_r = \frac{n \log n}{\varepsilon^2}$ walks from the query node u. For each walk $W(u) = (w_0 = u, ..., w_l)$ that stops at w_l, we first sample two $\sqrt{c}$-walks to get an unbiased estimation of $d(w)$. If the walks do not meet, we invoke *BackwardProbe*, a backward traversal procedure from w to derive an unbiased estimation of $s(u, v)$ for *each* v. The average of n_r trials will be returned as the estimated similarities.

The *BackwardProbe* algorithm. Given graph G, the query node u, the end node w of an $\sqrt{c}$-walk from u and of length l, and the error parameter ε of the single-source algorithm, *BackwardProbe* computes an unbiased estimation of $s(u, v)$ for each node v. Specifically, we first conduct backward search as long as the incurred cost is less than αn, where α is a small constant (e.g. 0.01). Note that we do not push nodes whose residues are too small, which is costly but contributes little to the accuracy of estimation. When all push operations are finished, we get two sets $H_j = \{\hat{\pi}_j(v, w), v \in V\}$ and $R_j = \{r_j(v, w), v \in V\}$ for each level j.

Since $\hat{\pi}_j(v, w)$ is always an underestimation of $\pi_j(v, w)$, we adopt a procedure analogous to *RandomizedProbe* [24], which transmits the residues in a randomized way to get the unbiased estimation of SimRank of v, denoted by $Score_l(u, v)$. The procedure iterates level by level. Note that for each level j, we are sufficient to consider at most n nodes (Lines 43-46). For each node y, we first add $\hat{\pi}_{j+1}(y, w)$ to $Score_{j+1}(u, y)$. Next, we randomly sample an in-neighbor x from $I(y)$, and add $r_j(x, w)$ to $Score_{j+1}(u, y)$ with probability $\sqrt{c}$. As we will see, $Score_l(u, y)$ is in fact an unbiased estimation of $s(u, y)$. As a comparison to *RandomizedProbe*, note that its estimation is either 0 or 1, which makes the variance of $\hat{s}(u, v)$ large, and thus the confidence bound is not sufficiently tight. We remedy this problem by incorporating backward search [27] to reduce the variance in practice, while the expected time complexity remains unaffected.

The stopping condition. Now we return back to the prefiltering algorithm and explain its stopping condition. Generally speaking, the prefiltering algorithm stops when it is not as efficient as applying sample-one-arm strategy on *all* candidates. In this way, adding the prefiltering phase does not deteriorate the theoretical optimality of the top-k identification algorithm. Let t_a (resp. t_o) denote the expected running time of a single *BackwardProbe* procedure (resp. $\sqrt{c}$-walk generation). Therefore, the stopping condition is set as $|C| \leq t_a/t_o$.

Note that the small error parameter ε_{min} is necessary to guarantee that the prefiltering algorithm finally stops. This special case usually occurs when the k-th and $k + 1$-th SimRank have the same value (i.e. $s(u, v_i) = s(u, v_{i+1})$). Consider an extreme case where $s(u, v_i) = c$ for every $i \in [1, n)$, and $\hat{s}(u, v_i) = s(u, v_i)$. Then, without ε_{min} any sampling-based algorithm will never terminate. On the other hand, the setting of ε_{min} guarantees that our result is at least as good as the result computed by the *Power Method*.

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 []. The following two Lemmas guarantee the effectiveness of *BackwardProbe* and *Single-source*, respectively.

LEMMA 3. *For each node $v \in V\backslash\{u\}$, $Score_l(u,v)$ returned by* BackwardProbe *is an unbiased estimation of the SimRank similarity $s(u,v)$.*

LEMMA 4. *At the i-th iteration of the prefiltering algorithm,* Single-source *returns $\hat{s}_i(u,v)$ for each node $v \in V\backslash\{u\}$, an unbiased estimation of $s(u,v)$ satisfying that with $1-\delta'$ probability, $|\hat{s}_i(u,v) - s(u,v)| \leq \varepsilon_i$.*

From the above Lemmas 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 [10] as the example. (We will explain the reason later.) The pseudo-code is demonstrated in Algorithm 3. 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 bound $\beta(v)$ for each candidate v, and a failure probability δ, *Top-k-Identification* returns the actual top-k nodes contained in C. Note that to efficient update the confidence bound we do not explicitly store $\beta(v)$. For the Chernoff bound, it is sufficient to record n_r, the number of samples for node v in the prefiltering phase. If the empirical Bernstein inequality is applied, to compute the empirical variance we also need to store $\sum_{i=1}^{n_r} \hat{s}_i(u,v)^2$, 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 $B(v) = \max\{UB(w) - LB(v), w \in C\backslash\{v\}\}$. Intuitively, $B(v)$ represents the upper bound of the gap between the best arm and arm v. To efficiently compute $B(v)$ for each candidate v, we first scan the upper confidence bound $UB(v)$ of each node v, and find the top-2 bounds. Then, each $B(v)$ can be computed in $O(1)$ time. Next, we take $O(|C|\log k)$ time to find the k nodes with smallest $B(v)$, denoted as V_k. Let v_h be the node in V_k having smallest $LB(v)$, and v_l the node in $C\backslash V_k$ having largest $UB(v)$. They represent the worst possible arm in V_k and the best possible arm in $C\backslash V_k$, respectively. If $LB(v_h)+\varepsilon_{min} > UB(v_l)$, we are confident that V_k is exactly the actual top-k nodes (with additive error ε_{min}). Otherwise, we sample the arm in $\{v_h, v_l\}$ with the larger confidence bound. This greedy strategy helps the algorithm to achieve optimal complexity. Note that each arm sampling is implemented by coupled random walk, from the query node u and arm v respectively. For the confidence bound, following [10], the confidence bound is computed based on the empirical Bernstein inequality:

$$\beta(v, n_r(v)) = \sqrt{\frac{\hat{\sigma}(v, n_r(v))\log\frac{|C|r^3}{\delta}}{n_r(v)}} + \frac{\frac{7}{6}\log\frac{|C|r^3}{\delta}}{n_r(v)-1}, \quad (8)$$

Algorithm 3: Top-k-Identification

Input: $G=(V,E)$; $u \in V$; k; candidate set $C = \{v, \hat{s}(u,v), \beta(v)\}$; failure probability δ;
Output: V_k as the top-k results

```
while true do
    for each v ∈ C do
        Compute UB(v) = ŝ(u,v) + β(v);
        Compute LB(v) = ŝ(u,v) − β(v);
    for each v ∈ C do
        Compute
        B(v) = max{UB(w) − LB(v), w ∈ C\{v}};
    Let V_k be the set of k nodes with smallest B(v);
    Let v_h ∈ V_k be the node with smallest LB;
    Let v_l ∈ C\V_k be the node with largest UB;
    if LB(v_h) + ε_min > UB(v_l) then
        return V_k;
    else
        Sample node v ∈ {v_h, v_l} with the larger β(v);
        Update ŝ(u,v) and β(v);
```

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

Remarks. We explain the reason of choosing UGapEc-V [10] as the MAB algorithm. Existing algorithms for top-k arm identification can be categorized into two classes, i.e., the uniformly sampling strategy [7, 12] and the adaptive sampling strategy. The former class includes the median elimination based methods [7] and the racing algorithms [12]. They rely on keep sampling all arms in the candidate set and eliminate arms that are not among the top-k. On the contrary, the adaptive sampling methods, such as LUCB [18] and UGapE [10] first identify the critical arms, and then sample them until the top-k estimated arms are separable from the others. Although both types of methods achieve comparable theoretical complexity, the empirical performance of the latter is significantly better when the number of arms are large. Moreover, it is unaware whether LUCB can integrate with the empirical Bernstein inequality. As our experiment shows, the tightness of the confidence bound also plays a critical role in the algorithm efficiency. Therefore, we choose a variant of the UGapE algorithm, which has been proved to nicely incorporate with the tighter confidence bounds.

Our Top-k-Identification algorithm bears one difference from the original UGapEc-V, i.e. the stopping condition. The original algorithm is designed for the approximate top-k MAB problems[3]. Instead, we aim to directly answer the exact top-k query.

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

LEMMA 5. Top-k-Identification *returns the actual top-k nodes with at least $1-\delta$ probability and with expected time complexity $O(H_k \log\frac{n}{\delta})$, where $H_k = \sum_{i=1}^{n} \frac{1}{\Delta_i{}^2}$ denotes the sample complexity of top-k queries.*

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

[3]In fact, the guarantee of answer quality is even weaker than the approximate single-source queries in [24, 31, 34].

THEOREM 2. *The* SimTab-Top-k *algorithm answers the top-k SimRank queries with at least $1-\delta$ success probability, using expected query time $O(H_k \log \frac{n}{\delta})$.*

3.4.2 Optimizations

The practical efficiency of SimTab-Top-k heavily relies on the tightness of the confidence bounds, for a tighter confidence bound effectively prunes more nodes and leads to fewer sample operations. In the prefiltering phase, we have adopted *BackwardProbe* which combines backward push [3] and *RandomizedProbe* [24] to significantly reduce the confidence bounds. For the top-k identification phase, we demonstrate how to integrate the idea of forward push [4], which has been employed to improve the performance of the Monte Carlo approach in computing Personalized PageRank [33, 35].

The algorithm, denoted by *Adaptive-Top-k*, is an optimization of *Top-k-Identification*. Recall that Top-k-Identification uses $\sqrt{c}$-walk sampling to estimate the reward of an arm (i.e. the SimRank similarity of a pair of nodes), which results in 0/1 estimations. Usually, a large amount of walks are sampled for each node until the algorithm terminates. We note that SimRank can be interpreted from the perspective of reverse Personalized PageRank, 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 [4], which generates estimators with smaller variances but with higher computation cost. Therefore, a straightforward idea of is to conduct as many forward push as possible until the asymptotical complexity of the push operation matches the cost of walk sampling. As more walks are generated for a candidate node v, more push operations are conducted to tighten the confidence bound. Hence, the number of push operation of each candidate varies in a adaptive way, which guarantees the efficiency of the algorithm.

The pseudo-code is demonstrated in Algorithm 4. For each level $l = [0, \infty)$, we first initialize $\hat{\pi}_l(v, w) = 0$ for each $v \in C \cup \{u\}$ and $w \in V$, and $r(v, w) = 0, r(v, v) = 1$ for each $v \in C \cup \{u\}$ and $w \in V \backslash \{v\}$. 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. In particular, we rewrite the Equation as follows.

$$s(u,v) = a \sum_{l=0}^{\infty} \sum_{w \in V} (\hat{\pi}_l(u,w) + \delta_l(u,w))(\hat{\pi}_l(v,w) + \delta_l(v,w))d(w), \tag{9}$$

where $a = \frac{1}{(1-\sqrt{c})^2}$, $\hat{\pi}_l(s, w)$ is the reserve by forward push, and $\delta_l(s, w)$ denotes an unbiased estimation of $\sum_{v \in V} r(s, v)\pi(v, w)$. 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 9. Next, for both u and v we sample a $\sqrt{c}$-walk as follows. 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 an $\sqrt{c}$-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 at w', we add $r_{sum}(u)r_{sum}(v)\hat{d}(w')$ to $\hat{s}(u, v)$. It can be proved that the estimator is unbiased for $s(u, v)$. For the estimation of $d(w)$, it can

Algorithm 4: Adaptive-Top-k Algorithm

Input: $G = (V, E)$; $u \in V$; k; candidate set $C = \{v, \hat{s}(u, v), \beta(v)\}$; failure probability δ;
Output: V_k, the estimated top-k nodes with largest SimRank values

Denote by $n_r(v)$ the number of sampled walks from node v in the top-k arm identification phase;
$n_r(u) = \sum_{v \in C} n_r(v), n'_r(v) = n_r(v), \forall v \in C \cup \{u\}$;
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
    $\hat{\pi}_l(v, w) = 0, \forall w \in V$;
Replace Line 13 of Algorithm ?? into:
$\hat{s}(u, v) = 0$;
for $l = 0, 1, \ldots$ **do**
  for *each* $w \in R_l(u) \cap R_l(v)$ **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 $\hat{\pi}_{l'_1}(x', w) \neq 0$ **then**
  Increase $\hat{s}(u, v)$ by $\hat{\pi}_{l'_1}(x', w)r_{sum}(v)\hat{d}(w)$;
if $\hat{\pi}_{l'_1}(y', w) \neq 0$ **then**
  Increase $\hat{s}(u, v)$ by $r_{sum}(u)\hat{\pi}_{l'_1}(y', w)\hat{d}(w)$;
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 14 of Algorithm ??:
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($G, v, \frac{\alpha}{n_r(v)}$);
    $n'_r(v) = n_r(v)$;
`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 $R_{l+1}(v)$;
      $r_l(v, w) = 0$;

be implemented by sampling two $\sqrt{c}$-walks from w, which gives 0/1 estimation.

LEMMA 6. *The time and space complexity of Adaptive-Top-k is asymptotically identical to those of the Top-k-Identification algorithm.*

4. THRESHOLDING QUERIES

In this section, we demonstrate how to adapt our arm sampling strategies and the techniques for confidence bound to thresholding queries, which returns all nodes with SimRank values larger than a threshold τ for a given query node u. By tackling the problem from the perspective of MAB, a greedy arm sampling strategy [26] is adopted for the sample-one-arm operations, which has been proved optimal[4]. Analogous to the top-k queries, to enhance practical efficiency we also employ the *sample-all-arms* strategy, which is implemented by the prefiltering algorithm and with a little modification of the pruning rule.

Our algorithm, denoted as *SimTab-Thres*, is shown in Algorithm 5. Given a directed graph $G = (V, E)$, a query node u, the threshold τ, and the failure probability δ, *SimTab-Thres* returns a node set V_τ such that with probability at least $1 - \delta$, each node v with $s(u, v) \geq \tau$ is included in the set, while nodes with SimRank values smaller than τ are excluded from V_τ. Similar to *SimTab-Top-k*, the algorithm contains a prefiltering phase followed by a refinement phase.

Algorithm 5: *SimTab-Thres*

Input: Directed graph $G = (V, E)$; $u \in V$; threshold τ; failure probability δ;
Output: V_τ, the estimated node set with SimRank value no smaller than τ

1 $V_\tau = \emptyset$;
2 C = Prefiltering($G, u, \tau, \frac{\delta}{2}$);
3 Initialize maximum queue $Q = \emptyset$;
4 Let $n_r(v)$ be the number of samples of node v in the prefiltering phase;
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** $\hat{s}(u, v) - \beta(v) > \tau$ **then**
11 Add v to V_τ;
12 **else if** $\beta(v) \leq \varepsilon_{min}$ *and* $\hat{s}(u, v) > \tau$ **then**
13 Add v to V_τ;
14 **else if** $\beta(v) > \varepsilon_{min}$ *and* $\hat{s}(u, v) + \beta(v) > \tau$ **then**
15 Update $p(v)$, and put $(v, p(v))$ back into Q;
16 **return** V_τ;

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 implemented as a triple $(v, \hat{s}(u, v), \beta(v))$, where $\beta(v)$ denotes the confidence bound and can be updated later in the refinement phase, according to the implementation introduced in Section 3. The prefiltering algorithm is analogous to Algorithm 2 but with a modified pruning rule.

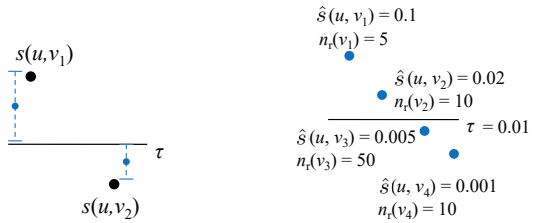
The pruning rule. For node v satisfying that $\hat{s}(u, v) - \beta(v) \geq \tau$, we can directly add it to V_τ; node v satisfying that $\hat{s}(u, v) + \beta(v) < \tau$ is discarded. If node v satisfies that $\hat{s}(u, v) - \beta(v) \leq \tau < \hat{s}(u, v) + \beta(v)$, we can not decide whether $v \in V_\tau$. 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 3(a)). Moreover, the time complexity asymptotically matches the hardness of the thresholding bandit problem. Nonetheless, we adopt a heuristic adaptive sampling strategy [], for which the reason will be amplified later.

The sampling strategy. At every step, we only sample the node with largest priority $p(v) = \sqrt{n_r(v)} \cdot |\hat{s}(u, v) - \tau|$ (Line 8).

This implies that a candidate v which is sample few times whereas having a large gap between the estimated value and the threshold should be re-considered. Consider the example in Figure 3(b), in which the estimated SimRank and number of samples are illustrated. We have $p(v_1) = \sqrt{5} \cdot 0.09 \approx 0.201$, $p(v_2) = \sqrt{10} \cdot 0.01 \approx 0.032$, $p(v_3) = \sqrt{50} \cdot 0.005 \approx 0.035$, $p(v_4) = \sqrt{20} \cdot 0.009 \approx 0.056$. Therefore, v_2 will be chosen. Intuitively, since the gap between v_1, v_4 and τ is sufficiently large, no more sample is needed presently; meanwhile, a large number of samples gives an accurate estimation for v_3.

Once we are sure that $v \in V_\tau$ (Lines 10-11) or the estimation is as accurate as the ground truth (Lines 12-13), v is moved to V_τ or discarded based on the estimation. 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, 34].



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

Figure 3: An example for the thresholding query.

Remarks. The advantage of the greedy sampling strategy over a naive MC algorithm lies in the approximate thresholding queries, which takes two threshold p and r as input, and guarantees that the returned node set has precision at least p and recall at least r. Note that our algorithm can be easily extended to the approximate setting with a slight modification of the pruning rule and stopping condition. In this case, the greedy arm sampling strategy achieves much superior empirical efficiency compared to *MC*, since each candidate are treated differently. Pleaser refer more details to the full version of our paper [].

[4]More precisely, in [26] the proposed method solves the thresholding bandit problem under the fixed budget setting, which minimizes the error of estimation under a given 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.

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* $O(H_\tau \log \frac{n}{\delta})$*, where* $H_\tau = \sum_{v \in V} \frac{1}{|s(u,v)-\tau|^2}$ *represents the hardness of the thresholding query.*

Optimizations. Recall that during the top-k identification phase of *SimTab-Top-*k, we employ adaptive forward push to tighten the confidence bounds without additional cost. This technique can be integrated to 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 which 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 increase 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 upper bound of score any node can gain with more than l steps, in fact, $(\frac{c}{d})^{l+1}$, 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.

We also note that a few existing work (such that [19]) also claims to answer top-k queries. However, they share similar heuristics with *TopSim*, including the distance-based upper bound for pruning, and a modified definition of the SimRank. Therefore, the empirical performance of these algorithms is less competitive.

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

Presently, the recently proposed single-source algorithms are the 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, the ProbeSim algorithm 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 meting 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 use index to improve the query performance. In general, they pre-compute a fraction of the results of similarity path enumeration or large amount of random walks during index construction, while the index can be reused in the query phase. *TSF* [29] constructs a set of *one-way graphs* via random in-neighbor sampling as the index, and answers single-source queries by online random walk sampling and traversal on one-way graphs. In the contrary, *SLING* [31] 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 the space cost. *READS* [16] 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* [34] improves the efficiency of backward stage via probability-guided search of similarity paths. To limit the index space, it only conducts backward search [27] 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 RPPR 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 [16, 29] 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 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. [15] proposes the *Power Method*, an iterative method to compute the all-pair SimRank matrix S. It is based on the matrix formulation of SimRank [19]: $S = (cP^\top SP) \vee I$, 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, 38, 40] 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 [9, 11, 19, 22, 36, 37, 39] has been proposed with a modified definition of SimRank that makes it easier to compute: $S = cP^\top SP + (1-c) \cdot I$. However, [19, 32] 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 [1,2] with edge number varying from tens of millions to billions, shown in Table. For each dataset, we randomly generate 1,000 query nodes for top-k and thresholding queries, respectively.

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* [34] and *READS* [16]. We include *TopSim* [20] and *TSF* [29] 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 obtain the code of *ProbeSim* [24], *READS* [16], and *PRSim* [34] from the authors, and implement all other algorithms in C++ and compile the codes with -O3 option. All experiments are conducted on a machine with a 2.6GHz CPU and 64GB memory. Following previous works [24, 31, 34], 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, on condition that the method is not out-of-time[5]. *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 $\varepsilon = 0.01$ is the typical setting. For *TopSim*, since the baseline algorithm is both slow and has inferior answer quality, we use *Prio-TopSim*, the optimized algorithm for evaluation. We vary T , the depth of the random walks from 3 to 6, whose default value is 3. *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 $(R_g, Rq) \in \{(100, 20), (300, 40), (600, 80), (900, 120)\}$ where $(100, 20)$ is the default value. For READS, we vary parameter r, the number of random walks generated for each node during preprocessing, and t, the maximum depth of the walks. Specifically, we set $(r, t) \in \{(100, 10), (500, 10), (500, 20), (1000, 20)\}$ while by default $(r, t) = (100, 10)$. For our algorithms, we set $\varepsilon_{min} = 10^{-6}$. The failure probability is set to 0.0001 for all sampling-based algorithms.

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 accuaracy of top-k queries. Given a query node u, denote by V_k the set of top-k nodes with 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 varying k in $\{1, 5, 10, 50, 100, 500, 1000\}$. Intuitively, as k increases, the problem incurs higher cost because the top-k answers are harder to be distinguished from other nodes.

Our query results is illustrated in Fig. 4. First of all, we are regretful to point out that all evaluated baselines fail to give the stable query performance in terms of Precision@k, as the precision fluctuates as k varies. Precisely, these algorithms achieve best performance around $k = 50$, while the precision decreases dramatically as k goes towards 1 or $1,000$, regardless of the error parameter ε. For large k, this is reasonable since the gap between the k-th largest and $k+1$-th largest SimRank values quickly decreases with k on real-world graphs, partially due to the power law degree distribution. To explain the performance degeneration on small k, note that the SimRank values of top-k answers also vary with the query node, and an error is incurred once the value is below the predefined error parameter for single-source algorithms. Since k is small, each error weights more than the queries with large k. On the contrary, 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. For all compared baselines, the index-based algorithms suffer from excessive indexing and querying cost, even can not finish in reasonable time on the two largest graphs (IT and FD). Meanwhile, algorithms without absolute error guarantee (such as TopSim and TSF) have significant inferior performance compared to other methods.

We also compare the query efficiency of different algorithms in Fig. 5. For all baselines, the query time heavily depends on the input parameter. Though the query speed under typical parameter setting is extremely fast, especially for index-based algorithms, the answer quality is unacceptable. Nonetheless, the precision of query results improves by a significant margin if the precise parameter is chosen, at the cost of much loner querying time. Still, the precision has a large gap from 1 for most methods, and remain non-stable for different k. On the other hand, the answer quality can not even be improved, because these algorithms can not efficiently query with smaller parameters for the tremendous time cost. As for *SimTab-Top-k*, the query time significantly outperforms *PRSim-IF*, the index-free algorithm with best performance among the baselines. Note that the query time of fig:exp-topk-qtime varies with k, which implies that our algorithm is adaptive to the hardness of the queries.

6.3 Evaluation of Thresholding Queries

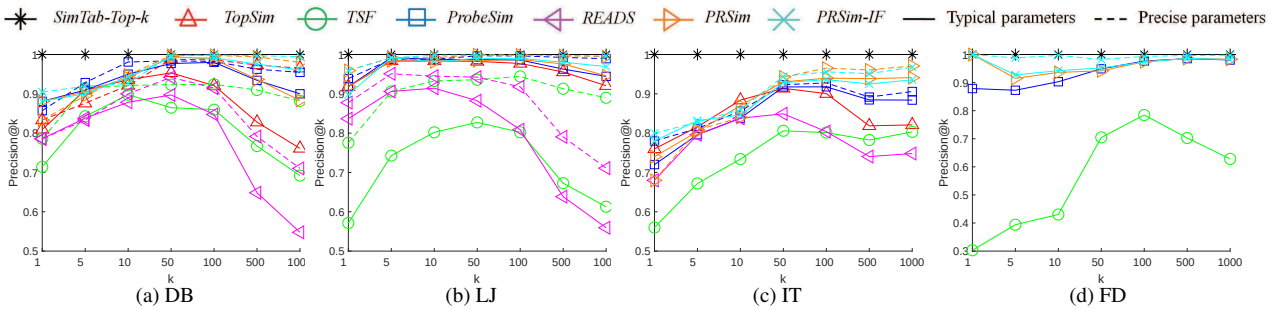
Metrics. To evaluate the thresholding queries, we use precision, recall and F1-score as the metrics. Specifically, let S be the set of exact answers for thresholding queries, and S' the node set returned by some specific algorithm. We define $Precision = \frac{|S \cap S'|}{|S'|}$, $Recall = \frac{|S \cap S'|}{|S|}$, and $F1-score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$.

The query results are demonstrated in Fig. 6 and 7. For space constraints, we only show the F1-Score in Figure 6. The following conclusions are easily derived. First, the query is harder for smaller τ, since most nodes have low SimRank values. Consequently, the F1-score of baseline algorithms decrease sharply. Second, *PRSim-IF* achieves the best performance among all index-free baselines with F1-score varying in 96% 99%, while *PRSim* is the best index-free algorithm. However, all index-based algorithms including *PRSim* can not scale well to more precise parameter settings. Again, our *SimTab-Thres* algorithm significantly outperforms any baselines in terms of query effectiveness, while the efficiency is superior to the baselines with precise parameters.

7. CONCLUSIONS

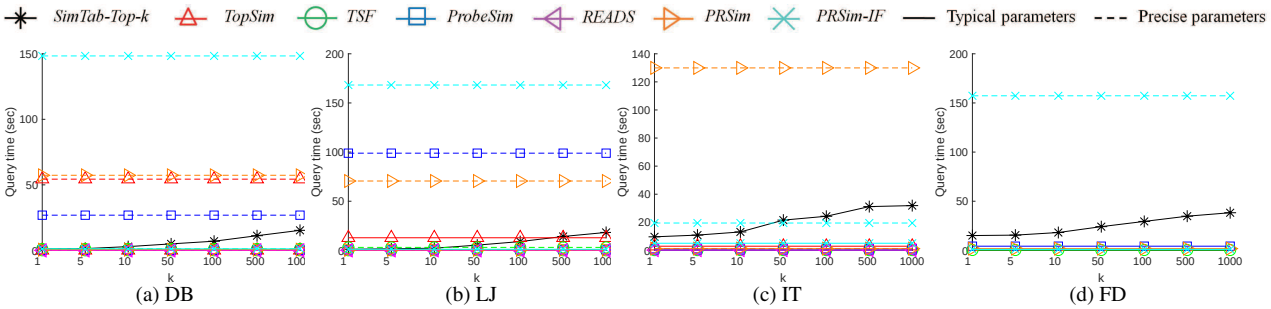
In this paper, we propose algorithms to improve both the efficiency and the effectiveness of the state-of-the-art methods over top-k and thresholding queries. Specifically, we integrate several techniques to tighten the confidence bounds of SimRank estimation, including the empirical Bernstein inequality, variance reduction tricks, and careful algorithm design. Moreover, we model the top-k and thresholding queries as the multi-armed bandits problems, so that the algorithm complexity is determined by the hardness

[5]We say an algorithm is out-of-time if the query time for some node is over 1,000 seconds or the index construction takes more than 1 hour. This indicates that the algorithm is not suitable for dynamic graphs.



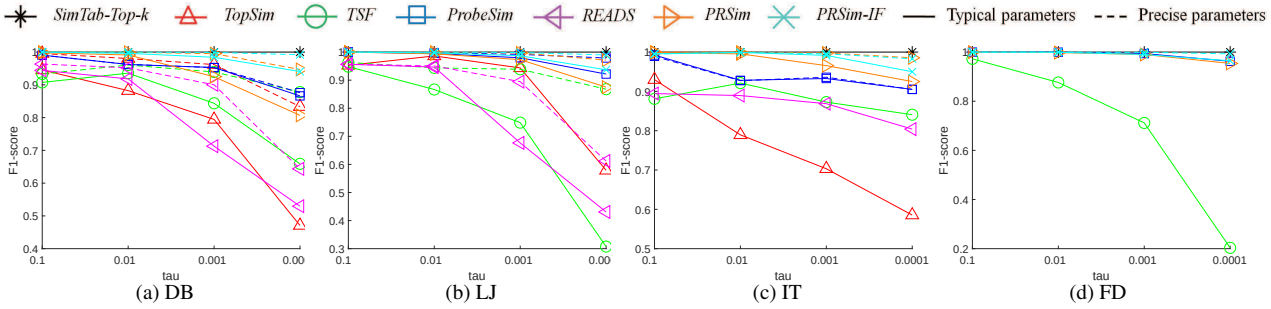
(a) DB (b) LJ (c) IT (d) FD

Figure 4: Precision@k for top-k queries, varying $k \in \{1, 5, 10, 50, 100, 500, 1000\}$



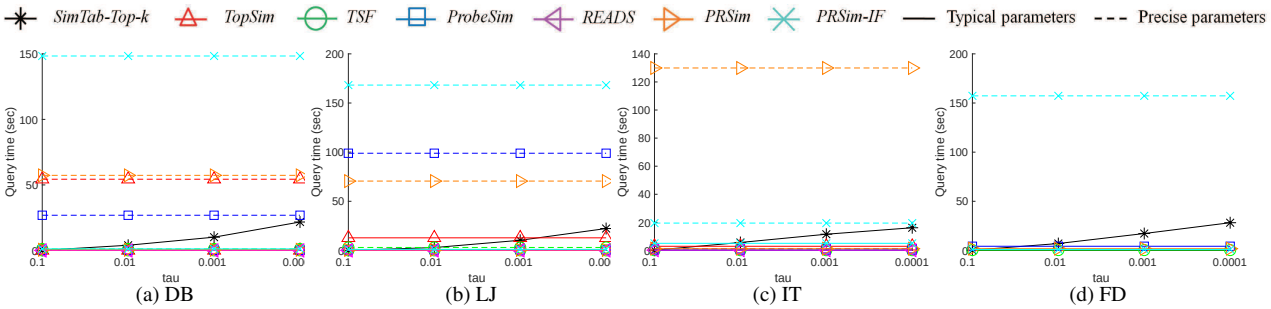
(a) DB (b) LJ (c) IT (d) FD

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



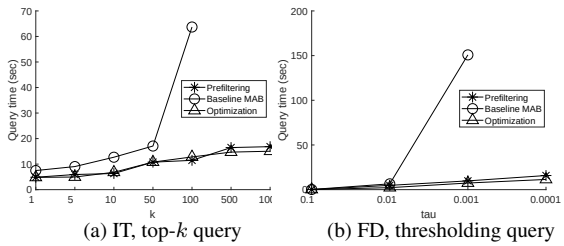
(a) DB (b) LJ (c) IT (d) FD

Figure 6: F1-score in thresholding queries, varying $\tau \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$



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

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



(a) IT, top-k query (b) FD, thresholding query

Figure 8: An example of our optimization techniques over top-k and thresholding SimRank queries.

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 which achieve stable and satisfying performance in terms of answer quality, and significantly outperform any known solutions.

8. REFERENCES

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

[2] http://konect.uni-koblenz.de/.

[3] 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.

[4] 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.

[5] 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.

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

[7] E. Even-Dar, S. Mannor, and Y. Mansour. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Journal of machine learning research*, 7(Jun):1079–1105, 2006.

[8] 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.

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

[10] 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.

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

[12] V. Heidrich-Meisner and C. Igel. Hoeffding and bernstein races for selecting policies in evolutionary direct policy search. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 401–408, 2009.

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

[14] 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.

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

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

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

[18] 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.

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

[20] 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.

[21] P. Lee, L. V. S. Lakshmanan, and J. X. Yu. On top-k structural similarity search. In *ICDE*, pages 774–785, 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] P. Lofgren, S. Banerjee, and A. Goel. Personalized pagerank estimation and search: A bidirectional approach. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 163–172, 2016.

[28] 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.

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

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

[31] 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.

[32] B. Tian and X. Xiao. SLING: A near-optimal index structure for simrank. In *SIGMOD*, pages 1859–1874, 2016.

[33] S. Wang, R. Yang, X. Xiao, Z. Wei, and Y. Yang. Fora: Simple and effective approximate single-source personalized pagerank. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 505–514, 2017.

[34] 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.

[35] 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 *Proceedings of the 2018 International Conference on Management of Data*, pages 441–456, 2018.

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

[37] 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.

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

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

[40] 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.

9. APPENDIX

9.1 Approximate Queries

9.1.1 Approximate top-k queries

SimTab-Top-k is able to answer the approximate top-k query with a slight modification of the algorithm. We first give the formal definition as below.

DEFINITION 6 (APPROXIMATE TOP-k QUERIES). *We are given a node u in G, a positive integer $k < n$, a precision guarantee p, and a failure probability δ, and let V_k be the set of nodes in G whose SimRank similarity to u is the top-k largest. An approximate top-k SimRank query returns a set of k nodes V'_k, such that $|V'_k \cap V_k| \geq pk$.*

Note that the definition is directly based on precision, rather than absolute error of the approximate single-source query or the (ε, k)-optimality of the top-k arm identification problem. This is because according to our empirical analysis, there does not exist a direct correlation between the top-k precision and the approximate error of SimRank similarities.

Our prefiltering algorithm is modified as follows. In each iteration i, after we compute an estimation $\hat{s}_i(u, v)$ (and $\beta_i(v)$) for each v, we find the node v' with the $\lfloor p'k \rfloor$-th smallest lower bound, and use it to select the candidate set (Lines 5-8 of Algorithm ?). This guarantees that at least $\lceil (1-p')k \rceil$ actual top-k answers are included in the candidates. Note that any choice of $p' \geq p$ is sufficient; in this paper, we set $p' = \frac{p+1}{2}$. For the top-k identification phase, we choose v_h as the node with the $\lfloor (p'-p)k \rfloor$-th smallest LB (Line 8 of Algorithm ?). Intuitively, since the stopping conditions are more relaxed, the approximate algorithm should terminate earlier than the exact counterpart.

9.1.2 Approximate thresholding queries

Our algorithm can be easily extended to answer the approximate thresholding query, which is defined as follows.

DEFINITION 7 (APPROXIMATE THRESHOLDING QUERIES). *We are given a node u in G, a real number $\tau \in [0, 1]$, a failure probability δ, and the approximation bound $p, r \in (0, 1)$ for precision and recall, respectively. An approximate thresholding SimRank query returns a set of nodes, denoted as V_{tau}, such that with $1 - \delta$ probability, $\frac{|V_{tau} \cap S|}{|V_{tau}|} \geq p$, $\frac{|V_{tau} \cap S|}{|S|} \geq r$. Here, we denote by S the exact answer for the thresholding query.*

Compared to the previously studied approximate queries that depend on an additional error parameter ε for the estimated SimRank value, the parameters p and r directly describe the quality of approximation for thresholding queries.

The approximate algorithm only has slight differences with the exact version. Firstly, note that we are able to determine a fraction of nodes in the prefilteing phase, i.e., putting them into V_τ or D. Otherwise, the node belongs to C. We assign a parameter $r' > r$ as the recall in the prefiltering phase. Once $\frac{|V_\tau|}{|V_\tau \cup C|} \geq r'$, the prefiltering algorithm terminates. Also note that the precision is not affected because it is deferred to the refinement phase.

Secondly, in the refinement phase, denote by C' the remained nodes in the candidate set with estimated SimRank value above the threshold. Once it holds that $\frac{|V_\tau|}{|V_\tau \cup C'|} \geq \max\{p, r\}$, we return all nodes in V_τ, plus the nodes in C' with empirical mean above τ. It is theoretically guaranteed that our proposed method returns an answer satisfying the approximation bound. As indicated by the empirical analysis, we are able to identify the answers with large SimRank values using far less cost than the exact algorithm, with the sacrifice of possibly losing those nodes with similarities close to τ.

9.1.3 SimTab-Single: The algorithm for approximate single-pair queries

In this section, we show that how to use the empirical Bernstein inequality to speed up approximate single-pair SimRank queries. Comparing to the commonly used Chernoff-Hoeffding inequality, the empirical Bernstein inequality takes the empirical variance into consideration, and is proven to be tighter when the esitimation is close to 0.

Empirical Bernstein Inequality. Previous work such as the Monte Carlo method employs Chernoff-Hoeffding inequality to bound the estimated error between the empirical and the true means. For single-pair SimRank queries and for any $u, v \in V$, denote by X_i the estimated SimRank value of the i-th pair of $\sqrt{c}$-walk. We have $\mathrm{E}[X_i] = s(u, v)$ and $X_i \in [0, 1]$ for each i. If we limit the failure probability to be no larger than δ, according to Lemma ?, after n_r samples we have

$$\varepsilon \geq \sqrt{\frac{1}{2n_r} \log \frac{2}{\delta}}. \tag{10}$$

Next, we consider applying the empirical Bernstein inequality instead. Intuitively, the empirical Bernstein inequality states that if the empirical variance is small, then the confidence bound is reversely related with the number of samples t.

LEMMA 7 (EMPIRICAL BERNSTEIN INEQUALITY [6]). *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,*

$$|\bar{x}_t - \mu| \leq \bar{\sigma}_t \sqrt{\frac{2 \ln 3/\delta}{t}} + \frac{3R \ln 3/\delta}{t},$$

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

For random walk based single-pair algorithms, we have $R = O(1)$. If the true mean $s(u, v)$ is close to zero, empirically $\bar{\sigma}$ is very small because each X_i is an unbiased sample of $s(u, v)$. Therefore, the confidence bound is approximately in reverse proportional to the sample size t, and is significantly tighter than the bound based on Chernoff-Hoeffding inequality which is in reverse proportional to $\sqrt{t}$. This suggests that algorithms based on the empirical Bernstein inequality requires fewer samples given an error threshold ε.

Our algorithm proceeds as follows (Algorithm 6). We start by setting the empirical sum X to be 0, and the number of samples t to be 1 (Line 1). For each iteration, we check if the stopping rule is satisfied (Line 2). If not, we take t more samples and double the value of t. Each sample is taken as follow. We set x to be u and y to be v (Line 4). On each step, we terminate the random walks with probability $1 - c$. With the remaining probability c, the two random walks proceed to an in-neighour of x and an in-neighour of y uniformly at random (Line 6). If the two random walks meet (Line 7), we increase the value of X by 1 and stop the random walks (Lines 8-9). Otherwise, the random walks proceed to the next step. When the stopping rule is satisfied, the algorithm stops and output $\hat{s}(u, v) = X/t$ as the estimation (Line 11).

To set the proper stopping time for single-pair algorithm, we need to make sure that when the algorithm stops, the condition

Algorithm 6: EBSim: The Single-Pair SimRank Algorithm

Input: Directed graph $G = (V, E)$; $u, v \in V$; Error ε and failure probability δ

Output: $\hat{s}(u, v)$, the estimation for the SimRank $s(u, v)$

1 $X \leftarrow 0, t \leftarrow 1$;
2 **while** $\sqrt{2X \ln \frac{3 \ln^2 t}{\delta}} + 3 \ln \frac{3 \ln^2 t}{\delta} \geq \varepsilon t$ **do**
3 **for** $s = 1$ *to* t **do**
4 $x \leftarrow u, y \leftarrow v$;
5 **while** $rand() \leq c$ **do**
6 $x \leftarrow rand(\mathcal{I}(x))$ and $y \leftarrow rand(\mathcal{I}(y))$;
7 **if** $x = y$ **then**
8 $X \leftarrow X + 1$;
9 Break;
10 $t \leftarrow 2 * t$;
11 **return** $\hat{s}(u, v) = X/t$;

$|\hat{s}(u, v) - s(u, v)| < \varepsilon$ holds with probability $1 - \delta$. Let t be the smallest t such that

$$\sqrt{2X \ln \frac{3 \ln^2 t}{\delta}} + 3 \ln \frac{3 \ln^2 t}{\delta} \leq \varepsilon t.$$

In particular, we have the following Lemma.

THEOREM 4. *With probability $1 - \delta$, Algorithm 6 outputs $\hat{s}(u, v) = X/t$ such that $|\hat{s}(u, v) - s(u, v)| < \varepsilon$.*

PROOF. Please refer to the full version of our paper []. □

Next, we analyze the time complexity of EBSim with the following theorem.

THEOREM 5. *The EBSim algorithm runs in $O(\frac{1}{\varepsilon} + \frac{s(u,v)}{\varepsilon^2} \log \frac{1}{\delta})$ time in expectation.*

PROOF. Please refer to the full version of our paper []. □

9.2 Additional Lemmas, Theorems, and Proofs

PROOF OF LEMMA ?. Note that the algorithm can only stop at $t = 2^i$ for $i = 0, 1, 2, \ldots$ Let $\mathcal{E}_i$ denote the event that the algorithm stops at $t = 2^i$, but gives wrong estimation, i.e., $|\hat{s}(u, v) - s(u, v)| \geq \varepsilon$. Since Algorithm 6 stops if and only if

$$\sqrt{2X \ln \frac{3 \ln^2 t}{\delta}} + 3 \ln \frac{3 \ln^2 t}{\delta} < \varepsilon t,$$

it follows that

$$\sqrt{\frac{X}{t} \cdot \frac{2 \ln \frac{3 \ln^2 t}{\delta}}{t}} + \frac{3 \ln \frac{3 \ln^2 t}{\delta}}{t} < \varepsilon.$$

Note that

$$X = \sum_{i=1}^{t} X_i = \sum_{i=1}^{t} X_i^2 \geq \sum_{i}^{t} (X_i - X/t)^2 = \bar{\sigma}_t^2,$$

Thus, we have

$$\bar{\sigma}_t^2 \cdot \sqrt{\cdot \frac{2 \ln \frac{3 \ln^2 t}{\delta}}{t}} + \frac{3 \ln \frac{3 \ln^2 t}{\delta}}{t} < \varepsilon.$$

By Lemma 7, it follows that with probability $1 - \frac{\delta}{\ln^2 t}$,

$$\Pr[\mathcal{E}_i] = \Pr[|\hat{s}(u, v) - s(u, v)| \geq \varepsilon] \leq \frac{\delta}{\ln^2 t} = \frac{\delta}{\ln^2 2^i} = \frac{1}{i^2} \cdot \frac{\delta}{\ln^2 2}.$$

By union bound, the probability that none of $\mathcal{E}_1, \mathcal{E}_2, \ldots, \ldots$ happens is bounded by

$$1 - \sum_{i=1}^{\infty} \Pr[\mathcal{E}_i] \geq 1 - \sum_{i=1}^{\infty} \frac{1}{i^2} \cdot \frac{\delta}{\ln^2 2} \geq 1 - \delta.$$

This proves that when Algorithm 6 stops, it outputs $\hat{s}(u, v) = X/t$ such that with probability $1 - \delta$, $|\hat{s}(u, v) - s(u, v)| < \varepsilon$. □

PROOF OF THEOREM ?. First note that for reasonable setting of δ (e.g., $\delta = \frac{1}{n}$), when $t = \Theta(\frac{1}{\varepsilon})$ it satisfies that

$$3 \ln \frac{3 \ln^2 t}{\delta} \leq \varepsilon t / 2.$$

Therefore, the probability that the algorithm does not stop at t is at most

$$\Pr\left[\sqrt{2X \ln \frac{3 \ln^2 t}{\delta}} > \frac{\varepsilon t}{2}\right].$$

We bound the probability for $t_i = O(2^i \cdot \frac{s(u,v)}{\varepsilon^2} \log \frac{1}{\delta}))$ for $i = 1, 2, \ldots$, repsectively. Note that the theorem holds if $t \leq \frac{s(u,v)}{\varepsilon^2} \log \frac{1}{\delta}$. Since

$$\Pr\left[\sqrt{2X \ln \frac{3 \ln^2 t}{\delta}} > \frac{\varepsilon t}{2}\right] = \Pr\left[2X \ln \frac{3 \ln^2 t}{\delta} > \frac{\varepsilon^2 t^2}{4}\right] = \Pr\left[\frac{X}{t} > \frac{\varepsilon^2 t}{8 \ln \frac{3 \ln^2 t}{\delta}}\right],$$

when $t = t_i = \frac{2^i \cdot s(u,v)}{\varepsilon^2} \log \frac{1}{\delta}$, the probability is

$$\Pr\left[\frac{X}{t} > s(u, v) \cdot \frac{2^i \log \frac{1}{\delta}}{8 \ln \frac{3 \ln^2 t}{\delta}}\right].$$

It can be proved that for reasonable setting of δ, $\frac{\log \frac{1}{\delta}}{8 \ln \frac{3 \ln^2 t}{\delta}} > \frac{1}{64}$ for all i and $s(u, v) \in [0, 1]$. By Markov's inequality,

$$\Pr\left[\frac{X}{t} > s(u, v) \cdot \frac{2^i}{64}\right] \leq \frac{64}{2^i}.$$

Therefore, for any $t = t_i$, we have $t \Pr\left[\sqrt{2X \ln \frac{3 \ln^2 t}{\delta}} > \frac{\varepsilon t}{2}\right] \leq \frac{64 s(u,v)}{\varepsilon^2} \log \frac{1}{\delta}$. Also note that $t = t_i$ and $t = t_j$ are mutually exclusive for $i \neq j$. Hence the expected running time of Algorithm 6 is bounded by $O(\frac{1}{\varepsilon} + \frac{s(u,v)}{\varepsilon^2} \log \frac{1}{\delta})$. □

PROOF OF THEOREM ?, TO BE REFINED. We first show that the prefiltering algorithm always terminates. Since the confidence bound $\beta_i(v)$ is in reverse proportion to $\sqrt{t_i}$ by Chernoff-Hoeffding inequality, or roughly in reverse proportion to t_i by empirical Bernstein inequality, it continuously shrinks. Precisely, we have $\beta_i(v) \leq \varepsilon_i$ for each $v \in V$ and each iteration i. When $\varepsilon_i \leq \frac{\varepsilon_p}{2}$, the pruning rule of Line 8 is essentially sorting all $\hat{s}(u, v)$, and the estimated similarities can be immediately returned and viewed as ground truth.

Second, we demonstrate that the algorithm fails with at most $\frac{\delta}{2}$ probability, i.e. some $(\varepsilon_p, k, \frac{\delta}{2})$-optimal node v is not included in the final C. This is only possible if during some iteration i it holds that $|\hat{s}(u, v) - s(u, v)| > \beta_i(v)$, which occurs with at most δ' probability. Note that we first set $\varepsilon_1 = \frac{1}{2}$ and halve the sample error parameter in each iteration, the algorithm has at most $\lceil \log_2 \frac{2}{\varepsilon_p} \rceil$

loops. Therefore the total failure probability can be bounded below $\frac{\delta}{2}$.

Finally, it is easy to see that by the pruning rule in Line 7, we guarantee that all nodes in C is $(\varepsilon_p, k, \frac{\delta}{2})$-optimal, and the Lemma follows. □

LEMMA 8. *The expected time and space complexity of* BackwardProbe *are bounded by* $O(n)$.

9.3 More Experiments