ABSTRACT

Given a graph $G$ and a node $u \in G$, a single source SimRank query evaluates the similarity between $u$ and every node $v \in G$. Existing approaches to single source SimRank computation incur either long query response time, or expensive pre-computation, which needs to be performed again whenever the graph $G$ changes. Consequently, to our knowledge none of them is ideal for scenarios in which (i) query processing must be done in realtime, and (ii) the underlying graph $G$ is massive, with frequent updates.

Motivated by this, we propose SimPush, a novel algorithm that answers single source SimRank queries without any pre-computation, and at the same time achieves significantly higher query processing speed than even the fastest known index-based solutions. Further, SimPush provides rigorous result quality guarantees, and its high performance does not rely on any strong assumption of the underlying graph. Specifically, compared to existing methods, SimPush employs a radically different algorithmic design that focuses on (i) identifying a small number of nodes relevant to the query, and subsequently (ii) computing statistics and performing residue push from these nodes only.

We prove the correctness of SimPush, analyze its time complexity, and compare its asymptotic performance with that of existing methods. Meanwhile, we evaluate the practical performance of SimPush through extensive experiments on 9 real datasets. The results demonstrate that SimPush consistently outperforms all existing solutions, often by an order of magnitude. In particular, on a commodity machine, SimPush answers a single source SimRank query on a web graph containing over 133 million nodes and 5.4 billion edges in under 62 milliseconds, with 0.00035 empirical error, while the fastest index-based competitor needs 1.18 seconds.

1. INTRODUCTION

SimRank is a popular similarity measure between nodes in a graph, with numerous potential applications, e.g., in recommendation systems [26], schema matching [25], spam detection [2], and graph mining [13, 19, 29]. The main idea of SimRank is that two nodes that are referenced by many similar nodes are themselves similar to each other. For instance, in a social network, two key opinion leaders who are followed by similar fans are expected to be similar in some way, e.g., sharing similar political positions or life experiences. Formally, given a graph $G$ and nodes $u, v \in G$, the SimRank value $s(u, v)$ between $u$ and $v$ is defined as follows:

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

where $I(u)$ and $I(v)$ are the sets of in-neighbors of $u$ and $v$, respectively, and $c \in [0, 1]$ is a decay factor commonly fixed to a constant, e.g., $c = 0.6$ [21, 31, 33].

This paper focuses on single-source SimRank processing, which takes as input a node $u \in G$, and computes the SimRank $s(u, v)$ between $u$ and every node $v \in G$. This can be applied, for example, in a search engine that retrieves web pages similar to a given one, or in a social networking site that recommends new connections to a user. We focus on online scenarios, in which (i) query execution needs to be done in realtime, and (ii) the underlying graph can change frequently and unpredictably, meaning that query processing must not rely on heavy pre-computations whose results are expensive to update. For large graphs, this problem is highly challenging, since computing SimRank values is immensely expensive: its original definition, presented above, is recursive and requires numerous iterations over the entire graph to converge, which is clearly unscalable.

Several recent approaches, notably [12, 15, 21, 28, 31, 33], have demonstrated promising results for single source SimRank processing, by solving the approximate version of the problem with rigorous result quality guarantees, as elaborated in Section 2. The majority of these methods, however, require extensive pre-processing to index the input graph $G$, as explained in Section 2.2, such indexes cannot be easily updated when the underlying graph $G$ changes, meaning that these methods are not suitable for our target scenarios described above. Specifically, the current state of the art for offline single source SimRank is PRSim [33], which achieves
Table 1: Comparison of single-source SimRank algorithms with error tolerance $\epsilon$ and failure probability $\delta$

| Algorithm | Query Time | Index Size | Preprocessing Time |
|-----------|------------|------------|-------------------|
| SimPush   | $O(m \log \frac{n}{\epsilon} + \log \frac{n}{\epsilon^2} + 1/\epsilon^3)$ | -          | -                 |
| TSF [28]  | $O(n \log \frac{n}{\epsilon^2})$ | $O(n \log \frac{n}{\epsilon^2})$ | $O(n \log \frac{n}{\epsilon^2})$ |
| READS [12]| $O(n \log \frac{n}{\epsilon^2})$ | $O(n \log \frac{n}{\epsilon^2})$ | $O(n \log \frac{n}{\epsilon^2})$ |
| ProbeSim  | $O(n \log \frac{n}{\epsilon^2})$ | -          | -                 |
| PRSim [33]$^1$ | $O(n/\epsilon)$ | $O(n/\epsilon)$ | $O(m/\epsilon + n \log \frac{n}{\epsilon^2})$ |

$^1$We simply convert each undirected edge $(u, v)$ to a pair of directed ones $(u, v)$ and $(v, u)$ with opposing directions.

1. We make no assumptions about the data distribution of the underlying graph $G$ (e.g., as in PRSim), which assumes that $G$ is a power-law graph, as elaborated in Section 2.2.

2. We experimentally evaluate our method against 6 recent solutions using 9 real graphs. The results demonstrate the high practical performance of SimPush. In particular, SimPush outperforms all existing methods (both indexed and index-free) in terms of query processing time, and SimPush is usually over an order of magnitude faster than the previous best index-free method ProbeSim, on comparable result accuracy levels.

We now describe our approximate SimRank algorithms, starting from attention nodes only. In particular, to ensure $\epsilon$-approximate result quality (defined in Section 2.1), it suffices to identify $O(\frac{1}{\epsilon})$ attention nodes. Existing solutions need to perform similar computations on a far larger set of nodes, covering the entire graph $G$ in the worst case.

Table 1 compares the asymptotic performance of SimPush against several recent approaches, where $n$ and $m$ denote the number of nodes and edges in $G$, respectively, and $\epsilon$ and $\delta$ are parameters for the error guarantee. For sparse graphs, $m$ is comparable to $O(n \log n)$; hence, compared to ProbeSim, the complexity of SimPush is lower for common values of $\epsilon$ and $\delta$. Further, SimPush does not involve large hidden constant factors (e.g., as in SLING), and makes no assumption on the data distribution of the underlying graph $G$ (e.g., as in PRSim), which assumes that $G$ is a power-law graph, as elaborated in Section 2.2.

We experimentally evaluate our method against 6 recent solutions using 9 real graphs. The results demonstrate the high practical performance of SimPush. In particular, SimPush outperforms all existing methods (both indexed and index-free) in terms of query processing time, and SimPush is usually over an order of magnitude faster than the previous best index-free method ProbeSim, on comparable result accuracy levels. Further, on UK graph with 133 million nodes and 5.4 billion edges, SimPush obtains 0.00035 empirical error within 62 milliseconds.

2. PRELIMINARIES

2.1 Problem Definition

Let $G = (V, E)$ be a directed graph, where $V$ is the set of nodes with cardinality $n = |V|$, and $E$ is the set of edges with cardinality $m = |E|$. If the input graph is undirected, $O(n \log \frac{n}{\epsilon^2} - \sum_{w \in V} \pi(w)^3)$ is the detailed time complexity of PRSim, where $\sum_{w \in V} \pi(w)^2 = 1$ in the worst case [33].
Approximate hitting probability from set of attention nodes at the \( \ell \)-th level in residue of attention node \( u \), and focus computations around these nodes only. As we show soon, the number of attention nodes is bounded by \( O(\frac{1}{\epsilon}) \), and they are mostly within the close vicinity of the query node \( u \), meaning that this is equivalent to the probability that two independent \( \sqrt{c} \)-walks starting from \( u \) never meet at any step.

**Table 2: Frequently used notations.**

| Notation | Description |
|----------|-------------|
| \( G = (V,E) \) | Input graph \( G \) with nodes \( V \) and edges \( E \) |
| \( n, m \) | \( n = |V|, m = |E| \) |
| \( \mathcal{O}(v), \mathcal{I}(v) \) | Out-neighbors and in-neighbors of node \( v \) |
| \( d_G(v), d_{\mathcal{I}}(v) \) | Out-degree and in-degree of node \( v \) |
| \( c \) | Decay factor in SimRank |
| \( \epsilon, \delta \) | Maximum absolute error and failure probability in approximate SimRank |
| \( \epsilon_b \) | Error parameter decided by \( \epsilon \) and \( c \) |
| \( G_u \) | Source graph generated for query node \( u \) |
| \( A_u \) | Set of all attention nodes with respect to \( u \) |
| \( A_u(\ell) \) | Set of attention nodes at the \( \ell \)-th level of \( G_u \), where \( \ell = 1, \ldots, L \) |
| \( L \) | Max level in \( G_u \) |
| \( w, w_1, w_j \) | Attention nodes at the \( \ell \)-th level, \((\ell+i)\)-th level, and \((\ell+j)\)-th level of \( G_u \), respectively, where \( \ell = 1, \ldots, L \) and \( i, j = 0, \ldots, L - \ell \) |
| \( h^{(\ell)}(u,w) \) | \( \ell \)-step hitting probability from \( u \) to \( w \) in \( G \) |
| \( h^{(\ell)}(u,w) \) | \( \ell \)-step hitting probability from \( u \) to \( w \) in \( G_u \) |
| \( h^{(\ell)}(v,w) \) | Approximate hitting probability from \( v \) to \( w \) in \( G \) |
| \( h^{(\ell)}(v,w) \) | Approximate hitting probability from \( v \) to \( w \) in \( G_u \) |
| \( \gamma^{(\ell)}(w) \) | Last-meeting probability of attention node \( w \) at the \( \ell \)-th level of \( G_u \) |
| \( r^{(\ell)}(w) \) | Residue of attention node \( w \), \( r^{(\ell)}(w) = h^{(\ell)}(u,w) \cdot \gamma^{(\ell)}(w) \) |
| \( \kappa^{(\ell)}(u,v,w) \) | The probability that two \( \sqrt{c} \)-walks from \( u \) and \( v \) meet at \( w \) at the \( \ell \)-th step, and never meet again afterwards. |

PRSim \cite{33}. PRSim is based on the main concepts of SLING, and further optimizes performance, especially for power-law graphs. PRSim builds a connection between SimRank and personalized PageRank \cite{11}: let \( \pi^{(\ell)}(u,v) \) be the \( \ell \)-hop reverse personalized PageRank (RPPR) between \( u \) and \( v \), we have \( \pi^{(\ell)}(u,v) = h^{(\ell)}(u,v) \cdot (1 - \sqrt{c}) \). PRSim uses Equation (4) for SimRank estimation:

\[
s(u,v) = \frac{1}{(1 - \sqrt{c})^2} \sum_{\ell=0}^{\infty} \sum_{w \in V} \pi^{(\ell)}(u,w) \cdot \pi^{(\ell)}(v,w).
\]  

Given query node \( u \), ProbeSim first samples a \( \sqrt{c} \)-walk \( W(u) \) from \( u \). For every node \( w \) at the \( \ell \)-th step of the walk, ProbeSim performs a probing procedure, in order to compute the first meeting probabilities at all levels. In particular, ProbeSim probes nodes in the order of increasing steps, so that when probing \( w \) at the \( \ell \)-th step of \( W(u) \), the method excludes the nodes visited in previous probeings, in order to compute the first meeting probabilities in Equation (5). Such inefficiency leads to long query response time, which may put off users who wait online for query results.

**Other methods.** READS \cite{12} precomputes \( \sqrt{c} \)-walks and compresses the walks into trees. During query processing, READS retrieves the walks originating from the query node \( u \), and finds all the \( \sqrt{c} \)-walks that meet with the \( \sqrt{c} \)-walks of \( u \). TSF \cite{28} builds an index consisting of one-way graphs by sampling one in-neighbor from each node’s in-coming edges. During query processing, the one-way graphs are used to simulate random walks to estimate SimRank. According to \cite{33}, PRSim subsumes both READS and TSF; further, \cite{33} points out that the result quality guarantee of TSF is questionable, since (i) TSF allows two walks to meet multiple times, leading to overestimated SimRank values and (ii) TSF assumes that a random walk has no cycles, which may not hold in practice. Finally, TopSim \cite{15} is another index-free method, which is subsumed by ProbeSim according to \cite{21}. Meanwhile, according to \cite{21,33}, the result quality guarantee of TopSim is problematic as the method truncates random walks with a maximum number of steps.

### 3. OVERVIEW OF SIMPUSH

We overview the proposed solution SimPush in this section, and present the detailed algorithm later in Section 4. As mentioned before, the main idea of SimPush is to identify a small set of attention nodes, and focus computations around these nodes only. As we show soon, the number of attention nodes is bounded by \( O(\frac{1}{\epsilon}) \), and they are mostly within the close vicinity of the query node \( u \), meaning that...
they can be efficiently identified. Meanwhile, we prove that the error introduced by neglecting non-attention nodes is negligible and bounded within the error guarantee $\epsilon$ in inequality (1). This design significantly reduces the computational overhead in SimPush.

Specifically, given the input graph $G$ and query node $u$, SimPush computes the approximate single source SimRank results for $u$ in three stages. The first stage identifies the set of attention nodes, denoted as $A_u$, through a Source-Push algorithm. Besides $A_u$, Source-Push also returns a graph $G_u$ (referred to as the source graph of $u$) consisting of nodes in $G$ that are visited during the algorithm. In the second stage, SimPush follows a similar (and yet much improved) framework as SLING, and computes the hitting probabilities between the query node $u$ and each attention node $w \in A_u$, as well as the last-meeting probability of $w$. Note that in SimPush, the computation of hitting probabilities is restricted to attention nodes, and heavily reuses the intermediate results obtained in the first stage, which drastically reduces the computational overhead compared to existing methods such as SLING, which precomputes hitting probabilities for all nodes in a graph by following out-going edges. Further, SimPush defines last-meeting probabilities over attention nodes only, and computes the probabilities in a deterministic way over a small source graph generated when computing the attention nodes (details in Section 4.1), while previous methods such as SLING defines its last-meeting probabilities over the whole graph, and precomputes the probabilities by sampling numerous $\sqrt{c}$-walks. Finally, in the third stage, SimPush employs a Reverse-Push approach to complete the estimates of probabilities between the query node $u$ and every node $v \in G$ via an attention node $w \in A_u$, yielding the final estimate of the SimRank between $u$ and $v$.

In the following, we elaborate on the three stages using the running example in Figure 1.

**Figure 1:** Running Example of SimPush

In the above definition of $s'(u, v)$, we enumerate all possible levels $\ell$. Next we show that this is not necessary, since attention nodes only exist in the first few levels within close vicinity of query node $u$, according to the following lemma.

**Lemma 1.** Given nodes $u, v \in G$, their exact SimRank $s(u, v)$ and estimated value $s'(u, v)$ in Equation (6) satisfy

$$s(u, v) - \frac{\sqrt{c} \cdot \epsilon_h}{1 - \epsilon} \leq s'(u, v) \leq s(u, v).$$

In the above definition of $s'(u, v)$, we enumerate all possible levels $\ell$. Next we show that this is not necessary, since attention nodes only exist in the first few levels within close vicinity of query node $u$, according to the following lemma.

**Lemma 2.** Given query node $u$, decay factor $c$ and parameter $\epsilon_h$, the number of attention nodes with respect to $u$ is at most $\left\lceil \frac{\sqrt{\gamma}}{(1 - \sqrt{c}) \cdot \epsilon_h} \right\rceil$. Meanwhile, all attention nodes exist within $L^* = \left\lceil \log \frac{1}{\epsilon} \cdot \frac{1}{\epsilon_h} \right\rceil$ steps from $u$.

According to Lemma 2, to discover all attention nodes, it suffices to explore $L^*$ steps around the query node $u$. Further, in SimPush, attention node discovery is performed by exploring $L \leq L^*$ steps from $u$, through the proposed Source-Push algorithm, detailed in Section 4.1. In particular, Source-Push samples a sufficient number of random walks to determine $L$, such that with high probability (according to parameter $\delta$), all attention nodes exist within $L$ steps from $u$. The specific value of $L$ depends on the input graph $G$. As our experiments demonstrate $L$ is usually small for real graphs. For instance, when $\epsilon = 0.02$, on a billion-edge Twitter graph, the average $L$ is merely 2.76, and the number of attention nodes is no more than a few hundred.

Next, to identify attention nodes, SimPush also needs to compute the hitting probabilities from $u$. This is done through a residue propagation procedure in the Source-Push algorithm, detailed in Section 4.1. Specifically, $h^{(0)}(u, u)$ is set to 1, and all other hitting probabilities are initialized to zero. Starting from the 0-th level, Source-Push pushes hitting probabilities of nodes from the current level to their in-neighbors on the next level, until reaching the $L$-th level. As mentioned earlier, SimPush also records the nodes and edges traversed during the propagation in a source graph $G_u$. Specifically, $G_u$ is organized by levels (with max level $L$), and there are only edges between adjacent levels, i.e., incoming edges from the $(\ell + 1)$-th level to the $\ell$-th level. $G_u$ itself, as well as the computed hitting probabilities of attention nodes, are reused in subsequent stages of SimPush.

**Figure 1(a)** shows an example of the propagation process, assuming $L = 3$ and $\epsilon_h = 0.12$. Attention (resp. non-attention) nodes are shown as solid circles (resp. empty circles) in the figure. Symbols with a superscript circle (e.g.,

---

1 All proofs can be found in the appendix.
Estimation of $\kappa^{(t)}(u,v,w)$. After identifying attention nodes, SimPush needs to estimate each $\kappa^{(t)}(u,v,w)$, according to Equation (6). Existing solutions mostly estimate it by running numerous $\sqrt{e}$-walks on the whole graph $G$, which is costly. Instead, SimPush incorporates a novel algorithm that mostly operates within the source graph $G_u$, obtained in the first phase. $G_u$ is far smaller than $G$.

Specifically, the hitting probabilities from $u$ to all attention nodes are already obtained Phase 1. Next, we focus on the last meeting probability for a given node $w$. In order to achieve high efficiency, SimPush only computes last meeting probabilities for attention nodes, and limits the computations within the source graph $G_u$. Towards this end, SimPush defines a new last meeting probability, as follows.

Definition 4. (Last-Meeting Probability in $G_u$). Given attention node $w$ on the $\ell$-th level of $G_u$, where $\ell = 1, \ldots, L$, the last-meeting probability of $w$ within $G_u$, $\gamma^{(t)}(w)$, is the probability that two $\sqrt{e}$-walks from $u$ and walking within $G_u$ do not meet at any attention node on the $(\ell + i)$-th level within $G_u$, for $1 \leq i \leq L - \ell$.

We emphasize that $\gamma^{(t)}(w)$ has vital differences compared to the last-meeting probability $\eta(w)$ used in SLING and PRSim, explained in Section 2.2. First, $\gamma^{(t)}(w)$ is defined based on the attention sets and source graph $G_u$, instead of the whole graph. Second, $\gamma^{(t)}(w)$ does not take into account whether or not two walks meet at any non-attention node; the rationale here is that non-attention nodes have negligible impact on the SimRank estimation of SimPush, and, thus, can be safely ignored. Third, $\gamma^{(t)}(w)$ is level-specific and we only consider $L - \ell$ steps in $G_u$ since there are only incoming edges between consecutive levels in $G_u$ and the levels are bounded by $L$. In Section 4.2, we present an efficient residue-push technique to compute the $\gamma^{(t)}(w)$ of all attention nodes, without performing any $\sqrt{e}$-walk.

Based on the above notion of last meeting probability, we design another estimate for the SimRank value $s(u,v)$ between the query node $u$ and a node $v \in G$, as follows.

$$s^+(u,v) = \sum_{\ell=1}^{L^*} \sum_{w \in A_u^{(t)}} h^{(t)}(u,w) \cdot \gamma^{(t)}(w) \cdot h^{(t)}(v,w),$$  

(7)

where $A_u^{(t)}$ is the set of attention nodes at the $\ell$-th level of $G_u$, obtained in the first phase. Note that here the trivial case of $\ell = 0$ is not considered, and we require $u \neq v$.

Compared to $s^+(u,v)$ defined in Equation (6), $s^+(u,v)$ uses an estimated $\kappa^{(t)}(u,v,w)$, computed using hitting probabilities and last-meeting probabilities in $G_u$. The following lemma establishes the approximation bound for $s^+(u,v)$.

**Lemma 3.** Given distinct nodes $u$ and $v$, their exact SimRank value $s(u,v)$ and estimate $s^+(u,v)$ satisfy

$$s(u,v) - \frac{2\sqrt{\epsilon} \cdot \epsilon}{1 - \sqrt{e}} \leq s^+(u,v) \leq s(u,v).$$

**Reverse-Push.** In Equation (7), it remains to clarify the computation of $h^{(t)}(v,w)$. Instead of estimating $h^{(t)}(v,w)$ independently (e.g., by simulating random walks), we propose a novel Reverse-Push algorithm, detailed in Section 4.3, which estimates $h^{(t)}(v,w) = h^{(t)}(u,v) \cdot \gamma^{(t)}(w)$ as the initial residue of attention node $w$, and keeps pushing the residue to each node $v \in G$, following out-going edges, until $b$ steps are performed.

For example, in Figure 1(b), given a 3rd level attention node $w$ with residue $r^{(3)}(w)$, Reverse-Push propagates the residue to the out-neighbors of $w$, i.e., $v_a$ and $v_b$, to obtain the residues at the 2nd level, i.e., $r^{(2)}(v_a)$ and $r^{(2)}(v_b)$. Then, all $r^{(2)}$ residues are pushed to their out-neighbors to get all $r^{(1)}$ residues. After that, all $r^{(1)}$ are pushed to get $r^{(0)}$ residues. It is clear that the nodes at the 0-th level, e.g., $v_g$ (as well as $v_h$ and $v_k$) meets with $u$ at $w$ in 3 steps. The residue $r^{(0)}(v_g)$ estimates $h^{(0)}(u,v) = h^{(0)}(u,w) \cdot \gamma^{(0)}(w) \cdot h^{(0)}(v_g, w)$ w.r.t. $r^{(0)}(w)$. The detailed push criteria is in Section 4.3.

Accordingly, our final SimRank estimate is

$$\tilde{s}(u,v) = \sum_{\ell=1}^{L^*} \sum_{w \in A_u^{(t)}} h^{(t)}(u,w) \cdot \gamma^{(t)}(w) \cdot h^{(t)}(v,w),$$

(8)

where $u$ and $v$ are distinct nodes in $G$. $A_u^{(t)}$ is the $\ell$-th level attention set. Here, the hitting probability $h^{(t)}(v,w)$ from $v$ to $w$ is hatted to signify that Reverse-Push introduces additional estimation error. Note that as described above, the estimation is over the entire product $h^{(t)}(u,w) \cdot \gamma^{(t)}(w) \cdot h^{(t)}(v,w)$ rather than the last term. Lemma 4 provides error guarantee for $\tilde{s}(u,v)$, and explains the value of $\epsilon_b$.

**Lemma 4.** Given distinct nodes $u$ and $v$ in $G$, error parameter $\epsilon$, and decay factor $c$, when $\epsilon_b \leq \frac{1 - \sqrt{\epsilon}}{\sqrt{\epsilon}}$, we have $s(u,v) - \tilde{s}(u,v) \leq \epsilon$.

Note that in Lemma 4, the error bound is deterministic, rather than probabilistic as in our problem definition in Inequality (1). This is due to the fact that in Equation (8), we enumerate up to $L^*$ levels instead of $L$ levels as in the actual algorithm, as mentioned earlier. The value of $L$, as well as the probabilistic error bound of the complete SimPush solution, are deferred to the next section. Finally, Table 3 lists the time complexity of the three stages of SimPush.

| Stage            | Time Complexity |
|------------------|-----------------|
| Source-Push      | $O(m \log \frac{1}{\epsilon} + \log \frac{1}{\epsilon^*})$ |
| All $\gamma^{(t)}(w)$ computation | $O(m \log \frac{1}{\epsilon} + 1/\epsilon^*)$ |
| Reverse-Push     | $O(m \log \frac{1}{\epsilon})$ |

**4. DETAILED SIMPUSH ALGORITHM**

Algorithm 1 shows the main SimPush algorithm, consisting of three stages. With $\epsilon_b$ set at Line 1, SimPush first
Algorithm 1: SimPush

Input: Graph $G = (V,E)$, query node $u$, decay factor $c$, error parameter $\epsilon$, failure probability $\delta$
Output: $\bar{s}(u,v)$ for each $v \in V$, w.r.t. query node $u$.
1. $\epsilon_h \leftarrow \frac{1}{\sqrt{\epsilon} \cdot \sqrt{\frac{1}{\epsilon} \cdot \frac{1}{\epsilon_c}}}$
2. Invoke Algorithm 2 (Source-Push) to obtain attention nodes and the source graph $G_u$.
3. Invoke Algorithm 3 to compute all nonzero hitting probabilities for attention nodes in $G_u$.
4. for $\ell = 1$ to $L$ do
5. for each attention node $w$ in $A^\ell_u$ do
6. Compute $\gamma^\ell(u,w)$ with Algorithm 4;
7. Invoke Algorithm 5 (Reverse-Push) to get $\tilde{s}(u,v)$ for each $v \in V$;
8. return $\bar{s}(u,v)$ for each $v \in V$;

Algorithm 2: Source-Push

Input: Graph $G$, query $u$, decay factor $c$, parameter $\epsilon_h$
Output: Source graph $G_u$ and attention node sets $A^\ell_u$ for $
\ell = 1, \ldots, L$.
1. $H^0(u,v) \leftarrow 0$, for $v \in V$ and $l = 1, 2, \ldots$
2. for $i = 1, \ldots, (2\log (1/\epsilon_c) + 1)/\epsilon_h^2)$ do
3. Generate a $\sqrt{\epsilon}$-walk from $u$ and for every visited node $v$ at the $l$-th step, $H^l(u,v) \leftarrow H^l(u,v) + 1$;
4. $L \leftarrow 0$
5. for every nonzero $H^l(u,v)$ do
6. if $l > L$ and $H^l(u,v) \geq \log \frac{1}{1-\sqrt{\epsilon} \cdot \sqrt{\frac{1}{\epsilon_c}}}$ then
7. $L \leftarrow l$;
8. $L \leftarrow \min(L, L^*)$;
9. $h^l(u,v) \leftarrow 0$ for $l = 1, \ldots, L$ and each $v \in V$;
10. $\ell \leftarrow 0$; $h^0(u,u) \leftarrow 1$;
11. Frontier set $F \leftarrow \{u\}$;
12. while $F \neq \emptyset$ and $\ell < L$ do
13. for each $v \in F$ do
14. for each node $v' \in I(v)$ do
15. $h^{(\ell+1)}(u,v') \leftarrow h^{(\ell+1)}(u,v') + \sqrt{\epsilon} h^l(u,v)$;
16. Insert $v$ to the $\ell$-th level and $v'$ to the $(\ell + 1)$-th level of $G_u$, and add edge from $v'$ to $v$ in $G_u$;
17. $F \leftarrow \emptyset$; $\ell \leftarrow \ell + 1$;
18. for each node $v$ with $h^\ell(u,v) > 0$ do
19. $F \leftarrow F \cup \{v\}$;
20. for $\ell = 1, \ldots, L$ do
21. Insert $w$ in $G_u$ with $h^\ell(u,v) \geq \epsilon_h$ into $A^\ell_u$;

Invokes Source-Push (Section 4.1) to obtain the attention nodes and source graph $G_u$ of $u$ (Line 2). Then (Lines 3-7), it computes the $\gamma^\ell(u,w)$ of all attention nodes $w$ (Section 4.2), and finally invokes Reverse-Push (Section 4.3) to compute the simple source SimRank values at Line 8.

4.1 Source-Push

Source-Push first samples a sufficient number of random walks to detect the max level $L$ from query node $u$, such that with high probability, all attention nodes appear within $L$ steps. Then, it performs residue propagation to compute the hitting probabilities from $u$, in order to identify attention nodes of $u$ and generate source graph $G_u$. Algorithm 2 displays Source-Push. At Lines 1-3, Source-Push first samples $2 \log (1/\sqrt{\epsilon}) \sqrt{\epsilon}$-walks from $u$, counts the visits of every node $v$ at every $\ell$-th step, $H^\ell(u,v)$, and then identifies the max level $L$ where there exists node $v$ with $H^\ell(u,v) \geq \log \frac{1}{1-\sqrt{\epsilon} \cdot \sqrt{\frac{1}{\epsilon_c}}}$, and $L$ is bounded by $L^*$ (Lines 4-8). Then, Algorithm 2 computes the hitting probabilities from $u$ for at most $L$ levels by propagation (Lines 9-19). Initially, at Lines 9-10, $h^0(u,v)$ is set to 1, all other hitting probabilities are initialized to zero. Starting from the 0-th level, Source-Push inserts $u$ into frontier set $F$ at Line 11, and then for each node $v$ in $F$ at the current $\ell$-th level, it pushes and increases the $(\ell + 1)$-level hitting probability of every in-neighbor $v'$ of $v$ by $\sqrt{\epsilon} h^\ell(u,v)$ and adds edge from $v'$ to $v$ in $G_u$ (Lines 12-16). Then, Source-Push moves to the $(\ell + 1)$-th level, and finds all the nodes to push (Lines 17-19). The whole process continues until the $L$-th level is reached or $F$ is empty (Line 12). At Lines 20-21, all attention nodes are identified. Lemma 5 states the accuracy guarantees and time complexity of Algorithm 2.

Lemma 5. Algorithm 2 runs in $O(m \log \frac{1}{\epsilon} + \log \frac{1}{\epsilon_c})$ expected time, and with probability at least $1 - \delta$, $G_u$ contains all nodes $w$ with $h^\ell(u,w) \geq \epsilon_h$ for all levels $\ell$.

Lastly, we define hitting probability within $G_u$, which is an important concept used in the next stages of SimPush.

Definition 5. (Hitting probability in $G_u$). Given nodes $w_u$ and $w_l$ in $G_u$, the hitting probability from $w_u$ to $w_l$ at the $i$-th step in $G_u$ is the probability that a $\sqrt{\epsilon}$-walk from $w_u$ and walking in $G_u$, visits $w_l$ at the $i$-th step, where $i \geq 0$.
\[ h^{(i)}(w, w_i) = \frac{\sqrt{c}}{d_2^i(w)} \sum_{w' \in \mathcal{I}^T(w)} h^{(i-1)}(w', w_i), \]

where \( \mathcal{I}^T(w) \) is the set of in-neighbors of \( w \) in \( G_u \) and \( d_2^i(w) \) is the indegree of \( w \) in \( G_u \), and \( i \geq 1 \). For example, in Figure 2, \( h^{(2)}(w_a, w_h) = \frac{\sqrt{3}}{4} \cdot \left( h^{(1)}(w_a', w_h) + h^{(1)}(w_e, w_h) \right) = 0.45 \).

Note that (i) in Equation (12), \( w' \) can be a non-attention node if it has nonzero hitting probabilities to attention nodes in \( G_u \), e.g., \( h^{(1)}(w_a', w_h) \) in the example; (ii) if node \( w \) has nonempty \( \mathcal{I}^T(w) \) in \( G_u \), \( \mathcal{I}^T(w) \) is the same as \( \mathcal{I}(w) \) in \( G \).

**Algorithms.** Next we present two algorithms: Algorithm 3 that computes the hitting probabilities between attention nodes within \( G_u \) using Equation (12), and Algorithm 4 that computes \( \gamma^{(\ell)}(w) \) using Equations (9), (10), and (11).

In Algorithm 3, all hitting probabilities are initialized to zero. Starting from \( \ell = L \) to 2, for each attention node \( w \), we first set \( h^{(0)}(w, w) \) to 1 at Lines 2-3 (i.e., the hitting probability to itself is 1). Then from Lines 4 to 7, for every node \( w' \) at the \( \ell \)-th level (including non-attention nodes), if it has nonzero hitting probabilities \( h^{(i)}(w', w) \) to any attention node \( w_i \), at the \( (\ell + i) \)-th level for \( i = 0, \ldots, L - \ell \), each of the probabilities \( h^{(i)}(w', w_i) \) is aggregated to every out-neighbor \( w_a' \) of \( w' \) in \( G_u \), where \( w_a' \) is at the \( (\ell - 1) \)-th level of \( G_u \) and can be a non-attention node. Apparently, from the perspective of \( w_a' \), we are aggregating its in-neighbors’ hitting probabilities to itself, i.e., Equation (12). Finally, only the hitting probabilities between attention nodes in \( G_u \) are returned and used by Algorithm 4 for computing \( \gamma^{(\ell)}(w) \).

Algorithm 4 computes \( \gamma^{(\ell)}(w) \) for attention node \( w \) at the \( \ell \)-th level of \( G_u \). At Line 1, \( \gamma^{(\ell)}(w) \) is initialized to 1. At Lines 2-4, when \( i = 1 \), all nonzero \( \rho^{(1)}(w, w_i) \) are computed according to Equation (10), and are subtracted from \( \gamma^{(\ell)}(w) \), based on Equation (9). Then all first-meeting probabilities \( \rho^{(i)}(w, w_i) \) for \( 2 \leq i < \Delta l \) are computed level by level from Lines 5 to 11, using Equation (11). Specifically, every \( \rho^{(i)}(w, w_i) \) is initialized as \( h^{(i)}(w, w_i) \) at Lines 6-7 and is subtracted by all non-first-meeting probabilities to \( w \) via \( w_j \) at Lines 8-11. Whenever the first-meeting probabilities \( \rho^{(i)}(w, w_i) \) for attention nodes \( w_i \) are obtained, they are subtracted from \( \gamma^{(\ell)}(w) \) at Line 12, accord-
reverse-push complexity for last-meeting computation is as a whole.

In this section, we call the (ℓ-1)-th level as the next level of the ℓ-th level. At current ℓ-th level, by pushing initial residue r(ℓ)(w) to the out-neighbors v of w in G, nodes v accumulate residue r(ℓ-1)(v) at the (ℓ-1)-th level. Then, Reverse-Push goes to the next level to push. After ℓ iterations, we have many nonzero r(ℓ)(v). Then r(ℓ)(v) estimates h(ℓ)(u,v) · γ(ℓ)(w) · h(ℓ)(w,v) with respect to r(ℓ)(w), and thus, r(ℓ)(v) is added to ˜s(u,v). Figure 1(b) shows an example that is already explained in Section 3. Further, instead of independently push for each attention node, we combine the push of the residues that are aggregated at the same node at the same level. For example, given node w with r(3)(w) at the 3-rd level of G and w’ with r(2)(w’) at the 2nd level, after pushing r(3)(w) to the out-neighbors v of w in G, we obtain many r(2)(v). If w’ happens to be an out-neighbor of w, the residue that gets from w and the residue of itself r(2)(w’) are combined and pushed together.

Algorithm 5 shows the pseudo code of Reverse-Push, which returns the estimated single source SimRank values. At Line 1, SimRank values ˜s(u,v) are initialized to zeros for v ∈ V. At Line 2, the initial residue of each attention node w is r(ℓ)(w), and the residues of all other nodes at all levels are zeros by default. Starting from level ℓ = L to 1, for every node v’ with residue r(ℓ')(v’) that satisfies √c · r(ℓ')(v’) ≥ ϵh, we propagate its residue to its out-neighbors v (Lines 3-5). If ℓ’ > 1, residue r(ℓ’-1)(v) is increased by √c · r(ℓ')(v’)/d2(v), where d2(v) is the indegree of v in G (Lines 6-7); if ℓ’ is 1, which means v is at the 0-th level, ˜s(u,v) is increased by √c · r(ℓ')(v)/d2(v) at Line 9. Finally, ˜s(u,v) is set to 1 and all SimRank values ˜s(u,v) for all v ∈ V are returned at Lines 10-11. The time complexity of Algorithm 5 is presented in Lemma 7.

**Lemma 7.** Algorithm 5 runs in O(m log 2/ϵ) time.

### 4.4 Correctness and Complexity Analysis

Theorems 1 and 2 present SimPush’s accuracy guarantee and time complexity, respectively.

**Theorem 1.** Given graph G, query node u, error parameter ϵ, and failure probability δ, Algorithm 1 returns an estimated SimRank value ˜s(u,v) that satisfies s(u,v) − ˜s(u,v) ≤ ϵ for each node v in G, with at least 1−δ probability, where s(u,v) is the exact SimRank value between u and v.

**Theorem 2.** In expectation, Algorithm 1 runs in O(m log 4/δ + log √c/δ/ε2 + 1/ε3) time.

### 5. EXPERIMENTS

We evaluate SimPush against the state of the art. All experiments are conducted on a Linux server with an Intel Xeon 2.60GHz CPU and 376GB RAM. All methods are in C++ and compiled by g++ 7.4 with -O3 optimization.

#### 5.1 Experimental Settings

**Methods.** SimPush is compared with six methods: PRSim [33], READS [12], TopSim [15], SLING [31], ProbeSim [21], and TSF [28]. ProbeSim and TopSim are index-free; PRSim, READS, TSF, SLING are index-based.

**Datasets and query sets.** We use 9 real-world graphs to evaluate SimPush and the competitors. The largest graph, ClueWeb, contains 1.68 billion nodes and 7.94 billion edges. The statistics of the graphs are shown in Table 4. There are 5 large graphs with billions of edges: ClueWeb, UK, Friendster, Twitter, and IT-2004, and 4 smaller graphs with millions of edges: In-2004, DBLP, Pokec, and LiveJournal.
### Table 4: Datasets used in the experiments.

| Name     | n       | m       | Type          |
|----------|---------|---------|---------------|
| In-2004  | 1,382,908 | 16,539,643 | directed     |
| DBLP     | 5,425,963 | 17,298,032 | undirected   |
| Pokec    | 1,632,803 | 30,622,564 | directed     |
| LiveJournal | 4,847,571 | 68,475,391 | directed     |
| TT-2004  | 41,291,594 | 1,315,718,909 | directed |
| Twitter  | 41,652,230 | 1,468,364,884 | directed     |
| Friendster | 65,608,366 | 3,612,134,270 | undirected   |
| UK       | 133,633,040 | 5,475,109,924 | directed     |
| ClueWeb  | 1,684,868,322 | 7,939,635,651 | directed     |

These graphs are of various types, including social networks, web graphs, and collaboration network. All datasets are available at [4, 16, 27]. For each dataset, we generate 100 queries by selecting nodes uniformly at random.

#### Parameters.
Following [21, 31, 33], we set the decay factor $c$ to 0.6, and fix the failure probability $\delta = 0.0001$. For SimPush, we vary $c$ in $(0.05, 0.2, 0.01, 0.005, 0.002)$. We set the parameters of all competitors following the settings in [33]. In particular, PRSim has two parameters: $e_a$, the absolute error threshold, and $j_0$, the number of hub nodes. We vary $e_a$ in $\{0.5, 0.1, 0.05, 0.01, 0.005\}$, and set $j_0$ to $\sqrt{n}$ by default [33]. We evaluate the static version of READS, which is the fastest among the three algorithms proposed in [12]. READS has two parameters: $r$, the number of $\sqrt{\epsilon}$-walks generated for each node in preprocessing stage, and $t$, the maximum depth of the $\sqrt{\epsilon}$-walks. We vary $(r, t)$ in $\{(10, 2), (50, 5), (100, 10), (500, 10), (1000, 20)\}$. TopSim has four parameters: $T$, the depth of random walks; $1/b$, the minimal degree threshold to identify a high degree node; $\eta$, the similarity threshold for trimming a random walk; $H$, the number of random walks to be expanded at each level. We fix $H$ and $\eta$ to their default values, i.e., 100 and 0.001, respectively. We vary $(T, 1/b)$ in $\{(1, 10), (3, 1000), (3, 10000), (4, 100000)\}$. SLING has a parameter $\epsilon_s$, which denotes the upper bound on the absolute error. We vary $\epsilon_s$ in $\{0.5, 0.1, 0.05, 0.01, 0.005\}$. ProbeSim also has an absolute error threshold $e_a$, which we vary in $\{0.5, 0.1, 0.05, 0.01, 0.005\}$. TSF has two parameters $R_g$ and $R_q$, which are the number of one-way graphs stored in the index and the times each one-way graph reused during query processing, respectively. We vary $(R_g, R_q)$ in $\{(10, 2), (100, 20), (200, 30), (300, 40), (600, 80)\}$. Note that every method is evaluated using its respective five parameter settings listed above; for each method, from its first to last parameter settings, it generates increasingly accurate results, with higher running time and memory usage.

#### Ground truth.
We get ground truth for the queries of all datasets by adopting the methods in [21, 33]. For small graphs, we directly apply Monte Carlo [5] to estimate SimRank for each query $u$ and each $v$ in $G$ with an absolute error less than 0.000001 and confidence over 99.999%, which is then used as the ground truth for $s(u,v)$. For large graphs, it is expensive to directly employ Monte Carlo to estimate $s(u,v)$ with high precision. We adopt the pooling method [21, 33] to generate ground truth for large graphs. Given query node $u$, we run each single-source algorithm, merge the top-$k$ nodes of each algorithm, remove the duplicates, and put them into a pool. For each node $v$ in the pool, we obtain the ground truth $s(u,v)$ by Monte Carlo. The ground truth top-$k$ node set $V_k$ is then the set of $k$ nodes with highest estimated SimRank values from the pool.

#### Metrics.
We adopt two metrics for accuracy evaluation, i.e., $\text{AvgError@}k$ and $\text{Precision@}k$, which are also used in [33]. We also evaluate the peak memory usage. $\text{AvgError@}k$ measures the average absolute error for approximating $s(u,v_i)$ for each node $v_i$ in the ground truth top-$k$ nodes $V_k$. Specifically, for each node $v_i$ in $V_k$, let $\hat{s}(u,v_i)$ denote the estimation of exact $s(u,v_i)$. $\text{AvgError@}k$ is defined as

$$\text{AvgError@}k = \frac{1}{k} \sum_{1 \leq i \leq k} |\hat{s}(u,v_i) - s(u,v_i)|.$$  

$\text{Precision@}k$ evaluates the ability to return the top-$k$ nodes for a query in terms of ground truth top-$k$ node set $V_k$. Suppose that $V_k' = \{v_{i_1}, \cdots, v_{i_k}\}$ is the top-$k$ nodes returned by the algorithm to be evaluated. $\text{Precision@}k$ is defined as

$$\text{Precision@}k = \frac{|V_k \cap V_k'|}{k}.$$  

#### Peak memory usage.
We enquiry Linux system resource files for `rusage.ru` for each method to get the peak memory usage of all methods over all datasets under all parameter settings.
5.2 Experimental Results
We evaluate the tradeoff between average error and query time, the tradeoff between average precision and query time, and the tradeoff between average error and peak memory usage for all methods over all graphs. We exclude a parameter of a method if it runs out of memory, or cannot finish preprocessing within 24 hours, or cannot finish a query in 1000 seconds. Given the query set of each graph, for each parameter setting of each method, we report the averages of query time, AvgError@50, Precision@50, and peak memory usage. Note that the preprocessing time of the index-based methods are not reported since our method SimPush is index-free.

Average error and query time. Figure 4 reports the tradeoff between AvgError@50 and query time of all methods over the first eight graphs in Table 4 (results on ClueWeb are reported separately later on). x-axis is AvgError@50 and y-axis is query time in seconds(s); both are in log-scale. For each method, the plot contains a curve with 5 points, which corresponds to results for its 5 settings (from right to left) described earlier. SimPush is superior over all methods by achieving lower error with less query time, and consistently outperforms existing solutions, especially on large graphs, no matter whether the competitor is index-free (e.g., ProbeSim) or index-based (e.g., PRSim). To reach the same level of empirical error, SimPush is much faster than the competitors, often by over an order of magnitude.

On graph UK, in Figure 4(h), to achieve $3.5 \times 10^{-4}$ AvgError@50, SimPush uses 0.062 seconds, while the index-based state-of-the-art PRSim needs 1.18 seconds, and the index-free ProbeSim uses 1.9 seconds and only achieves $9 \times 10^{-4}$ error. In Figure 4(f) for Twitter, which is known as a hard graph for SimRank computation due to its locally dense structure as analyzed in the paper of PRSim [33], SimPush also outperforms PRSim by a significant gap. To achieve $1.4 \times 10^{-4}$ error, PRSim requires 2.7 hours of precomputation and 9.1 seconds for query processing, while our online method SimPush only needs 1.5 seconds in total to achieve the same level of error. For ProbeSim, it needs 725 seconds to achieve such error on Twitter. As aforementioned, the max level $L$ of $G_u$ is usually small for real-world graphs. For instance, when $\varepsilon = 0.02$, on Twitter, $L$ is just 2.76 on average, and on DBLP, $L$ is 9.0. This indicates that the attention nodes that can largely contribute to the SimRank values are truly in the vicinity of query nodes. The number of attention nodes is usually in dozens or hundreds. Therefore, SimPush that first finds the attention nodes that can largely contribute to SimRank values and then focuses on such nodes for estimation, is rather efficient. On the graphs in Figures 4(a)-(d), SimPush also exceeds all competitors by a large gap. SLING, READS, TSF, and TopSim, are all inferior to SimPush over all these graphs.

Average precision and query time. Figure 5 reports the tradeoff between Precision@50 and query time of all methods over the first eight graphs (the evaluation on ClueWeb is presented later). x-axis is Precision@50, and y-axis is query time in seconds(s) and is in log-scale. For each method, the plot contains a curve with 5 points corresponding to its 5 parameter settings (from left to right). The overall observation is that SimPush provides the best precision and query time tradeoff in most settings, especially on large graphs. On the large graphs in Figures 5(e)-(h), to achieve the same level of precision, SimPush is much faster than all index-free and index-based competitors. For instance, on UK in Figure 5(h), SimPush achieves 96% precision in 0.062s, while both ProbeSim and PRSim requires 0.6s to achieve 96% precision. The performance gap between SimPush and the competitors remains for varying parameters. As analyzed, our method SimPush focuses computation only on the attention nodes of query $u$, and only explores the vicinity of $u$ to estimate SimRank values, which is highly efficient. For small graphs in Figures 5(a)-(d), to achieve the same level of precision, e.g., above 96%, TSF, TopSim, READS, and SLING, are consistently outperformed by SimPush.

Average error and peak memory usage. Figure 6 shows the peak memory usage of all methods. The memory usage includes the size of the input graphs, the indices (if any), and any other structures required by the methods. The x-axis is AvgError@50 and is in log-scale, and the y-axis is the peak memory usage in GigaBytes (GB). For each method, the plot contains a curve with 5 points correspond-
Results on Billion-Node ClueWeb. Figure 7 reports the evaluation results on the ClueWeb dataset, including AvgError@50, Precision@50, and peak memory usage measurements. TSF, TopSim, READS, and SLING are not reported since their memory requirements exceed that of our server (376GB). Figure 7(a) reports the tradeoff between AvgError@50 and query time. SimPush significantly outperforms PRSim and ProbeSim, often by orders of magnitude. Similarly, in Figure 7(b), SimPush achieves far more favorable tradeoff between Precision@50 and query time. For instance, to achieve 99.8% precision, SimPush takes 0.01s, while PRSim needs 1s and ProbeSim uses 0.122s. Figure 7(c) shows the tradeoff between peak memory usage and accuracy. SimPush uses about 147 GB memory, whereas PRSim and ProbeSim each consumes more than 250 GB memory.

6. RELATED WORK

We review existing work for SimRank computation, excluding SLING [31], ProbeSim [21], READS [12], TSF [28], TopSim [15] and PRSim [33], discussed in Section 2.2.

Power method [10] is the first for all-pair SimRank computation and it computes SimRank values of all node pairs in the input graph G by the matrix formulation in [14]:

\[ S = (c \mathbf{P}^\top \mathbf{S}) \mathbf{P} + (1 - c) \cdot \mathbf{I}. \]  

However, as pointed out by [14], the SimRank computed by Equation (14) are rather different from the correct values.

An early work [6] proposes a Monte Carlo approach to approximate SimRank by sampling conventional random walks. An index structure is also proposed to store random walks. However, the index incurs tremendous space and preprocessing overheads, which makes the Monte Carlo method inapplicable on sizable graphs [14,31]. Maehara et al. [23] propose an index structure for top-k SimRank queries, relying on heuristic assumptions about graphs, and thus, does not provide worst-case error guarantee [21,33]. A distributed version of the Monte Carlo approach is proposed by Li et al. [18], and the distributed method can scale to a billion-node graph at the significant cost of computation resources; the distributed environment is a different setting that is orthogonal to our study. There are also studies [1,5,20,38,40] on variants of SimRank, and SimRank similarity join [24,30]. However, these solutions are inapplicable for single-source SimRank queries.

7. CONCLUSION

In this paper, we propose SimPush, a novel index-free algorithm that answers single source SimRank queries with rigorous guarantees, and the method is significantly faster than even the fastest known index-based solutions, often by over an order of magnitude, which is confirmed by our extensive experimental evaluation on real-world web-scale graphs. In the future, we plan to study SimRank queries with relative error guarantees, batch SimRank processing, as well as SimRank computation on new hardware.
APPENDIX

Lemma 8. \( \sum_i X_i = N \) and for all i, \( X_i \in (0, \epsilon_b) \). \( \sum_i Y_i = M, M \in (0, 1) \) and for all i, \( Y_i > 0 \). Then \( \sum_i X_i Y_i \leq \epsilon_b M \).

**Proof of Lemma 1.** Obviously, \( s'(u, v) \leq s(u, v) \) holds.

Now we prove \( s(u, v) - \frac{1}{\sqrt{\epsilon_b}} \leq s'(u, v) \). Let \( G^{(\ell)} \) be the set of all nodes at \( \ell \)-th level of \( G_v \). \( G_v \) is source graph of \( v \) by pushing \( L^* \) levels from \( v \). Sum \( \sum_{w \in G^{(\ell)}_v} h^{(i)}(v, w) = \sqrt{\epsilon_b} \). The error of non-attention nodes at \( \ell \)-th level: \( \sum_{w \in G^{(\ell)}_v \setminus A^{(\ell)}_u} h^{(i)}(v, w) \leq \sqrt{\epsilon_b} \). For \( w \in G^{(\ell)}_v \setminus A^{(\ell)}_u \), we have \( h^{(i)}(v, w) \leq \epsilon_h \). Apply Lemma 8 and \( \eta(w) \leq 1 \), we have \( \sum_{w \in G^{(\ell)}_v \setminus A^{(\ell)}_u} h^{(i)}(v, w) \eta(w) \leq \epsilon_h \sqrt{\epsilon_b} \). Summing the error of all levels, \( \sum_{\ell=1}^\infty \epsilon_h \sqrt{\epsilon_b} \leq \frac{1}{\sqrt{\epsilon_b}} \). From Eq. (3), \( \sum_{\ell=1}^\infty \sum_{w \in A^{(\ell)}_u} \kappa^{(i)}(u, v, w) \) = \( \sum_{\ell=1}^\infty \sum_{w \in A^{(\ell)}_u} h^{(i)}(w, u) \), \( \eta(w) \cdot h^{(i)}(v, w) \). Thus, \( s(u, v) - \frac{1}{\sqrt{\epsilon_b}} \leq s'(u, v) \).

**Proof of Lemma 2.** At level \( \ell \), \( \sum_{w \in G^{(\ell)}_v} h^{(i)}(u, w) = \sqrt{\epsilon_b} \).

Hence, at level \( \ell \), there exists at most \( \frac{\sqrt{\epsilon_b}}{1 - \epsilon_b} \) attention nodes, and for \( \ell > L^* \), \( w \in G^{(\ell)}_v, h^{(i)}(u, w) \leq \epsilon_h \). Therefore, the size of attention set \( A_u \) is at most \( \sum_{\ell=1}^\infty \frac{\sqrt{\epsilon_b}}{1 - \epsilon_b} \leq \frac{1}{\epsilon_b} \).

**Proof of Lemma 3.** \( f^{(i)}(u, v, w) \) is the \( \ell \)-th step first meeting probability at \( w \), and we can write \( s(u, v) \) as \( s(u, v) = \sum_{\ell=1}^\infty \sum_{w \in G^{(\ell)}_v} f^{(i)}(u, v, w) \). Given \( A_u \), let \( s_1(u, v) = \sum_{\ell=1}^\infty \sum_{w \in A^{(\ell)}_u} f^{(i)}(u, v, w) \), and \( s_2(u, v) = \sum_{\ell=1}^\infty \sum_{w \in A^{(\ell)}_u} f^{(i)}(u, v, w) \). Obviously, \( s(u, v) = s_1(u, v) + s_2(u, v) \).

We want to prove \( s_1(u, v) \geq s_1(u, v) - s_2(u, v) \), and \( s_2(u, v) \leq \frac{1}{\sqrt{\epsilon_b}} \).

From Eq. (9),

\[
\begin{align*}
\frac{1}{\epsilon_b}
\end{align*}
\]

Consider the probability that two \( \sqrt{\epsilon_b} \)-walks from \( u \) and \( v \), first meet at attention node \( w' \) then meet at attention node \( w \). Given two events: (i) two \( \sqrt{\epsilon_b} \)-walks from \( u \) and \( v \) respectively, first meet at some attention node, then meet at \( w \), and (ii) these two walks meet at attention node \( w_a \), then two walks from \( w_a \) first meet at \( w \), the two events hold one-to-one correspondence. The probability of the first event corresponds to the last line of Eq. (16) and the latter event corresponds to the last line Eq. (15). Let \( \delta_1(u, v) = \sum_{\ell=1}^\infty \sum_{w \in G^{(\ell)}_v} f^{(i)}(u, v, w) \). Thus, \( s'(u, v) = s_1(u, v) - s_1(u, v) \), and \( s_1(u, v) \) is the probability that two \( \sqrt{\epsilon_b} \)-walks first meet at non-attention node, then meet at attention node. Thus, \( s_1(u, v) = s_1(u, v) \leq s_2(u, v) \).

Now we prove \( s_2(u, v) \leq \frac{1}{\epsilon_b} \). Based on Lemma 8, \( s_2(u, v) = \sum_{\ell=1}^\infty \sum_{w \in G^{(\ell)}_v \setminus A^{(\ell)}_u} f^{(i)}(u, v, w) \leq \sum_{\ell=1}^\infty \sum_{w \in G^{(\ell)}_v \setminus A^{(\ell)}_u} h^{(i)}(v, w) \leq \frac{1}{\epsilon_b} \).

**Proof of Lemma 4.** In Algorithm 5, consider the lose of Simrank at level \( \ell \). Similar to prove Lemma 1, the lose at level \( \ell \leq \epsilon_h \sqrt{\epsilon_b} \). Summing up all levels, the total loss is \( \leq \frac{1}{\epsilon_h \sqrt{\epsilon_b}} \).

**Proof of Lemma 5.** We push \( O(L^*) = O(\log \frac{1}{\epsilon}) \) levels and each level needs \( O(m) \) times, and thus the total time is \( O(m \log \frac{1}{\epsilon}) \). \( h^{(i)}(u, v) \) is the Monte Carlo estimation of \( h^{(i)}(u, v) \). From Hoeffding bound \( \delta \), \( \Pr(h^{(i)}(u, v) - \epsilon_b \leq 1 - \exp[-2\epsilon^2 / \epsilon^2]) = 1 - (1 - \sqrt{\epsilon_b}) \).

**Proof of Lemma 6.** Algorithm 3 costs \( O(m) \) per level of \( G_u \). Node \( w \) has \( O(1/\epsilon) \) hitting probabilities from \( w \). Thus the complexity of one level is \( O(m/\epsilon) \). There are \( O(\log \frac{1}{\epsilon}) \) levels. Total complexity is \( O(m \log \frac{1}{\epsilon}) \). Let \( Z_l \) be the number of nodes in \( G_u \) at level \( l \). For all \( w \in G^{(l+1)}_u \), the cost of all \( \rho^{(i)}(v, w_1) \) is \( O(Z_{l+1}) \). For all \( w \in G^{(l+2)}_u \), from Eq. (9), the cost of all \( \rho^{(i)}(v, w_2) \) is \( O(Z_{l+1} Z_{l+2}) \). Similarly, we can compute all \( w \) in \( G^{(l+1)}_u \) for all \( i > 0 \) in \( O(Z_{l+1} Z_{l+2}) \). The expected time of \( C^{(i)}(u, v) \) is \( O(Z_{l+1} + Z_{l+2}) \).

**Proof of Theorems 1 & 2.** Given Lemma 3, 4, 5, Theorem 1 follows. Given Lemma 5, 6, 7, Theorem 2 follows.
8. REFERENCES

[1] I. Antonellis, H. Garcia-Molina, and C. Chang. Simrank++: query rewriting through link analysis of the click graph. PVLDB, 1(1):408–421, 2008.
[2] A. A. Benczúr, K. Csalóğány, and T. Sarlós. Link-based similarity search to fight web spam. In AIRWEB, pages 9–16, 2006.
[3] A. D. Brody and A. Clauset. Scale-free networks are rare. Nature Communications, 10(1017), 2019.
[4] U. degli studi di Milano. http://law.di.unimi.it/datasets.php, 2004.
[5] D. Fogaras and B. Rácza. Scaling link-based similarity search. In WWW, pages 641–650, 2005.
[6] D. Fogaras, B. Rácz, K. Csalógány, and T. Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. Internet Mathematics, 2(3):333–358, 2005.
[7] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, and M. Onizuka. Efficient search algorithm for simrank. In ICDE, pages 589–600, 2013.
[8] G. He, H. Feng, C. Li, and H. Chen. Parallel simrank computation on large graphs with iterative aggregation. In SIGKDD, pages 543–552, 2010.
[9] W. Hoeffding. Probability inequalities for sums of bounded random variables. Journal of the American statistical association, 58(301):13–30, 1963.
[10] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In SIGKDD, pages 538–543, 2002.
[11] G. Jeh and J. Widom. Scaling personalized web search. In WWW, pages 271–279, 2003.
[12] M. Jiang, A. W. Fu, R. C. Wong, and K. Wang. READS: A random walk approach for efficient and accurate dynamic simrank. PVLDB, 10(9):937–948, 2017.
[13] R. Jin, V. E. Lee, and H. Hong. Axiomatic ranking of network role similarity. In SIGKDD, pages 922–930, 2011.
[14] M. Kusumoto, T. Maehara, and K. Kawarabayashi. Scalable similarity search for simrank. In SIGMOD, pages 325–336, 2014.
[15] P. Lee, L. V. S. Lakshmanan, and J. X. Yu. On top-k structural similarity search. In ICDE, pages 774–785, 2012.
[16] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, 2014.
[17] 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.
[18] Z. Li, Y. Fang, Q. Liu, J. Cheng, R. Cheng, and J. C. S. Lui. Walking in the cloud: Parallel simrank at scale. PVLDB, 9(1):24–35, 2015.
[19] D. Liben-Nowell and J. M. Kleinberg. The link-prediction problem for social networks. JASIST, 58(7):1019–1031, 2007.
[20] Z. Lin, M. R. Lyu, and I. King. Matchsim: a novel similarity measure based on maximum neighborhood matching. Knowl. Inf. Syst., 32(1):141–166, 2012.
[21] 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. PVLDB, 11(1):14–26, 2017.
[22] D. Lizorkin, P. Velikhov, M. N. Grinev, and D. Turdakov. Accuracy estimate and optimization techniques for simrank computation. PVLDB, 1(1):422–433, 2008.
[23] T. Maehara, M. Kusumoto, and K. Kawarabayashi. Efficient simrank computation via linearization. CoRR, abs/1411.7228, 2014.
[24] T. Maehara, M. Kusumoto, and K. Kawarabayashi. Scalable simrank join algorithm. In ICDE, pages 603–614, 2015.
[25] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In ICDE, pages 117–128, 2002.
[26] P. Nguyen, P. Tomceo, T. D. Noia, and E. D. Sciascio. An evaluation of simrank and personalized pagerank to build a recommender system for the web of data. In WWW, pages 1477–1482, 2015.
[27] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In AAAI, 2015.
[28] Y. Shao, B. Cui, L. Chen, M. Liu, and X. Xie. An efficient similarity search framework for simrank over large dynamic graphs. PVLDB, 8(8):838–849, 2015.
[29] J. Shi, N. Mamoulis, D. Wu, and D. W. Cheung. Density-based place clustering in geo-social networks. In SIGMOD, pages 99–110, 2014.
[30] W. Tao, M. Yu, and G. Li. Efficient top-k simrank based similarity join. PVLDB, 8(3):317–328, 2014.
[31] B. Tian and X. Xiao. Sling: a near-optimal index structure for simrank. In SIGMOD, pages 1859–1874, 2016.
[32] Y. Wang, X. Lian, and L. Chen. Efficient simrank tracking in dynamic graphs. In ICDE, page 545, 2018.
[33] Z. Wei, X. He, X. Xiao, S. Wang, Y. Liu, X. Du, and J. Wen. Prsim: Sublinear time simrank computation on large power-law graphs. In SIGMOD, pages 1042–1059, 2019.
[34] W. Yu, X. Lin, and W. Zhang. Fast incremental simrank on link-evolving graphs. In ICDE, pages 304–315, 2014.
[35] 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.
[36] W. Yu and J. A. McCann. Efficient partial-pairs simrank search for large networks. PVLDB, 8(5):569–580, 2015.
[37] W. Yu and J. A. McCann. Gauging correct relative rankings for similarity search. In CIKM, pages 1791–1794, 2015.
[38] W. Yu and J. A. McCann. High quality graph-based similarity search. In SIGIR, pages 83–92, 2015.
[39] 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.
[40] P. Zhao, J. Han, and Y. Sun. P-rank: a comprehensive structural similarity measure over information networks. In CIKM, pages 553–562, 2009.