Realtime Index-Free Single Source SimRank Processing on Web-Scale Graphs

Jieming Shi[†][*], Tianyuan Jin[‡][*], Renchi Yang[≀], Xiaokui Xiao[†], Yin Yang[§]
[†‡]School of Computing, National University of Singapore, Singapore
[≀]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[§]College of Science and Engineering, Hamad Bin Khalifa University, Qatar
[†]{shijm, xkxiao}@nus.edu.sg, [‡]tianyuan1044@gmail.com,
[≀]yang0461@e.ntu.edu.sg, [§]yyang@hbku.edu.qa

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 precomputation, and achieves significantly higher query 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 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 over 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.

PVLDB Reference Format:
Jieming Shi, Tianyuan Jin, Renchi Yang, Xiaokui Xiao, Yin Yang. Realtime Index-Free Single Source SimRank Processing on Web-Scale Graphs. *PVLDB*, 13(7): 966-978, 2020.
DOI: https://doi.org/10.14778/3384345.3384347

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, 40]. 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:

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

where $\mathcal{I}(u)$ and $\mathcal{I}(v)$ 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, 30, 32].

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-computions 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, 30, 32], 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 [32], which achieves efficient query processing with a relatively lightweight in-

[*]Equal contribution.

Proceedings of the VLDB Endowment, Vol. 13, No. 7
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3384345.3384347

Table 1: Comparison of single-source SimRank algorithms with error tolerance ϵ and failure probability δ

Algorithm	Query Time	Index Size	Preprocessing Time
SimPush	$O(m\log\frac{1}{\epsilon}/\epsilon + \log\frac{1}{\epsilon\delta}/\epsilon^2 + 1/\epsilon^3)$	-	-
TSF [28]	$O\left(n\log\frac{n}{\delta}/\epsilon^2\right)$	$O\left(n\log\frac{n}{\delta}/\epsilon^2\right)$	$O\left(n\log\frac{n}{\delta}/\epsilon^2\right)$
READS [12]	$O\left(n\log\frac{n}{\delta}/\epsilon^2\right)$	$O\left(n\log\frac{n}{\delta}/\epsilon^2\right)$	$O\left(n\log\frac{n}{\delta}/\epsilon^2\right)$
ProbeSim [21]	$O\left(n\log\frac{n}{\delta}/\epsilon^2\right)$	-	-
SLING [30]	$O(n/\epsilon)$	$O(n/\epsilon)$	$O\left(m/\epsilon + n\log\frac{n}{\delta}/\epsilon^2\right)$
PRSim [32][1]	$O\left(n\log\frac{n}{\delta}/\epsilon^2\right)$	$O(\min\{n/\epsilon, m\})$	$O(m/\epsilon)$

dex; nevertheless, it is clearly infeasible to rebuild index for every graph update or new query, as shown in our experiments in Section 5. The current best index-free solution is ProbeSim [32], whose query efficiency is far lower than that of PRSim. Consequently, ProbeSim yields poor response time for large graphs, adversely affecting user experience.

This paper proposes SimPush, a novel index-free solution for approximate single source SimRank processing that achieves significantly higher performance compared to all existing solutions (including index-based ones with heavy precomputation), while providing rigorous quality guarantees. This is achieved through a novel algorithmic design that (i) identifies a small subset of nodes in G that are most relevant to the query, called *attention nodes*, and subsequently (ii) computes important statistics and performs graph traversal starting from attention nodes only. In particular, to ensure ϵ-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 ϵ and δ 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 ϵ and δ. 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, we simply convert each undirected edge (u, v) to a pair of directed ones (u, v) and (v, u) with opposing directions. Following common practice in previous work [21, 32], we define the approximate single-source SimRank query as follows. Table 2 lists frequently used notations in the paper.

Definition 1. (Approximate Single Source SimRank Query) Given an input graph $G = (V, E)$, a query node $u \in V$, an absolute error threshold ϵ, a failure probability δ, and decay factor c, an approximate single source SimRank query returns an estimated value $\tilde{s}(u, v)$ for the exact SimRank $s(u, v)$ of each node $v \in V$, such that

$$|\tilde{s}(u,v) - s(u,v)| \le \epsilon \tag{1}$$

holds for any $v \in V$ with at least $1 - \delta$ probability.

2.2 State of the Art

SLING [30]. SimRank is well known to be linked to random walks [10]. Earlier work on SimRank processing generally use random walks without decay. More recent approaches are mostly based on a variant called $\sqrt{c}$-walks, as follows.

Definition 2. ($\sqrt{c}$-Walk [30]) Given node u and decay factor c, $\sqrt{c}$-walk from u is a random walk that (i) has $1 - \sqrt{c}$ probability to stop at current node, and (ii) has $\sqrt{c}$ probability to jump to a random in-neighbor of current node.

Given two $\sqrt{c}$-walks from distinct nodes u and v respectively, we say that these two $\sqrt{c}$-walks *meet*, if they both reach the same node after the same number of steps, say, the ℓ-th step. Let $\kappa^{(\ell)}(u, v, w)$ be the probability that two $\sqrt{c}$-walks from u and v meet at w at the ℓ-th step, and *never meet again afterwards*. Ref. [30] interprets the SimRank value $s(u, v)$ as follows:

$$s(u,v) = \sum_{\ell=0}^{+\infty} \sum_{w \in V} \kappa^{(\ell)}(u,v,w). \tag{2}$$

SLING [30] further decomposes $\kappa^{(\ell)}(u, v, w)$ into the product of three probabilities:

$$\kappa^{(\ell)}(u,v,w) = h^{(\ell)}(u,w) \cdot \eta(w) \cdot h^{(\ell)}(v,w), \tag{3}$$

where $h^{(\ell)}(u, w)$ denotes the probability (called *hitting probability*) that a $\sqrt{c}$-walk from node u reaches node w at the ℓ-th step. Since the random walks starting from nodes u and v are independent, the product $h^{(\ell)}(u, w) \cdot h^{(\ell)}(v, w)$ gives the probability (called *meeting probability*) that these two walks meet at node w (called the *meeting node*). The correction factor $\eta(w)$ (called the *last-meeting probability* of node w) is the probability that the above two $\sqrt{c}$-walks, after meeting at w, never meet again in the future. Clearly, this is equivalent to the probability that two independent $\sqrt{c}$-walks starting from w never meet at any step.

[1] $O\left(n\log\frac{n}{\delta}/\epsilon^2 \cdot \sum_{w\in V}\pi(w)^2\right)$ is the detailed time complexity of PRSim, where $\sum_{w\in V}\pi(w)^2 = 1$ in the worst case [32].

Table 2: Frequently used notations.

Notation	Description
$G=(V,E)$	Input graph G with nodes V and edges E
n, m	$n = \lvert V \rvert, m = \lvert E \rvert$
$\mathcal{O}(v), \mathcal{I}(v)$	Out-neighbors and in-neighbors of node v
$d_{\mathcal{O}}(v), d_{\mathcal{I}}(v)$	Out-degree and in-degree of node v
c	Decay factor in SimRank
ϵ, δ	Maximum absolute error and failure probability in approximate SimRank
ϵ_h	Error parameter decided by ϵ and c
G_u	Source graph generated for query node u
A_u	Set of all attention nodes of u
$A_u^{(\ell)}$	Set of attention nodes at the ℓ-th level of G_u, where $\ell = 1, ..., L$
L	Max level in G_u
w, w_i, w_j	Attention nodes at the ℓ-th level, $(\ell+i)$-th level, and $(\ell+j)$-th level of G_u respectively, where $\ell = 1, ..., L$ and $i = 0, ..., L-\ell$
$h^{(\ell)}(u,w)$	ℓ-step hitting probability from u to w in G
$\tilde{h}^{(\ell)}(u,w)$	ℓ-step hitting probability from u to w in G_u
$\hat{h}^{(\ell)}(v,w)$	Approximate hitting probability from v to w in G
$\gamma^{(\ell)}(w)$	Last-meeting probability of attention node w at the l-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 ℓ-th step, and never meet again afterwards.

SLING then pre-computes $h^{(\ell)}(u,w)$ and $\eta(w)$ with error up to ϵ, and materializes them in its index. Given a query node u, SLING retrieves all nodes at all levels with $h^{(\ell)}(u,w) \geq \epsilon$. Then, for each level ℓ and every node w on the ℓ-th level, SLING retrieves $\eta(w)$ and each node v with $h^{(\ell)}(v,w) \geq \epsilon$, and estimates $s(u,v)$ using Equation (3).

SLING incurs substantial pre-processing costs for computing $h^{(\ell)}(u,w)$ and $\eta(w)$, which need to be re-computed whenever graph G changes, as there is no clear way to efficiently update them. Consequently, SLING is not suitable for online processing. Further, although SLING achieves beautiful asymptotic bounds as shown in Table 1, its practical performance tends to be sub-par due to large hidden constant factors. For instance, Ref. [32] points out that the index size of SLING is over an order of magnitude larger than G itself, which leads to high retrieal costs at query time. Our experiments in Section 5 lead to similar conclusions.

PRSim [32]. 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 [11]: let $\pi^{(\ell)}(u,w)$ be the ℓ-hop reverse personalized PageRank (RPPR) between u and w, we have $\pi^{(\ell)}(u,w) = h^{(\ell)}(u,w) \cdot (1-\sqrt{c})$. PRSim uses Equation (4) for SimRank estimation:

$$s(u,v) = \frac{1}{(1-\sqrt{c})^2} \sum_{\ell=0}^{+\infty} \sum_{w \in V} \pi^{(\ell)}(u,w) \cdot \eta(w) \cdot \pi^{(\ell)}(v,w). \quad (4)$$

Then, based on the assumption that the input graph G is a power-law graph, PRSim selects a number of hub nodes, and pre-computes their RPPR values. At query time, PRSim estimates $\pi^{(\ell)}(u,w) \cdot \eta(w)$ by generating $\sqrt{c}$-walks from u and w. If w happens to be a hub, PRSim seeks the index for all possible $\pi^{(\ell)}(v,w)$ for any $v \in V$; otherwise, $\pi^{(\ell)}(v,w)$ is estimated online using a sampling based technique. Finally, PRSim estimates $s(u,v)$ based on Equation (4).

Similar to SLING, PRSim incurs considerable precomputation as explained above, and hence, it is not suitable for online SimRank processing. Further, PRSim heavily relies on the power-law graph assumption, both in algorithm design and in its asymptotic complexity analysis. In particular, in the best case that the underlying graph G strictly follows power-law, the query time complexity is sublinear to the graph size [32]. However, this assumption is rather strong and might be unrealistic: as reported in a recent study [3], strict power-law graphs are rare in practice.

ProbeSim [21]. The state-of-the-art index-free method is ProbeSim. Specifically, let $W(u)$ and $W(v)$ be two $\sqrt{c}$-walks from nodes u and v, respectively, and $f^{(\ell)}(u,v,w)$ be the probability that $W(u)$ and $W(v)$ *first meet* at w at the ℓ-th step. ProbeSim employs Equation (5) to estimate SimRank:

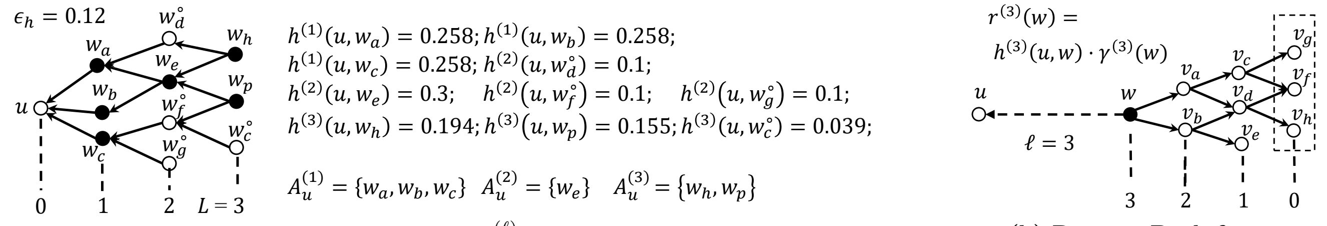
$$s(u,v) = \sum_{\ell=0}^{+\infty} \sum_{w \in V} f^{(\ell)}(u,v,w). \quad (5)$$

Given query node u, ProbeSim first samples a $\sqrt{c}$-walk $W(u)$ from u. For every node w at the ℓ-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 ℓ-th step of $W(u)$, the method excludes the nodes visited in previous probings, 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 [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 [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 simlulate random walks to estimate SimRank. According to [32], PRSim subsumes both READS and TSF; further, [32] 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 [15] is another index-free method, which is subsumed by ProbeSim according to [21]. Meanwhile, according to [21,32], the result quality guaranee 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



$\epsilon_h = 0.12$
w_d°
w_h
w_a
w_e
w_p
w_b
w_f°
u
w_g°
w_c°
w_c
0
1
2
$L = 3$
$h^{(1)}(u, w_a) = 0.258; h^{(1)}(u, w_b) = 0.258;$
$h^{(1)}(u, w_c) = 0.258; h^{(2)}(u, w_d^\circ) = 0.1;$
$h^{(2)}(u, w_e) = 0.3; \quad h^{(2)}(u, w_f^\circ) = 0.1; \quad h^{(2)}(u, w_g^\circ) = 0.1;$
$h^{(3)}(u, w_h) = 0.194; h^{(3)}(u, w_p) = 0.155; h^{(3)}(u, w_c^\circ) = 0.039;$
$A_u^{(1)} = \{w_a, w_b, w_c\} \quad A_u^{(2)} = \{w_e\} \quad A_u^{(3)} = \{w_h, w_p\}$
$r^{(3)}(w) =$
$h^{(3)}(u, w) \cdot \gamma^{(3)}(w)$
u
w
$\ell = 3$
v_a
v_b
v_c
v_d
v_e
v_g
v_f
v_h
3
2
1
0

(a) Source graph G_u and attention sets $A_u^{(\ell)}$: attention nodes are in black. **(b) Reverse-Push from w**

Figure 1: Running Example of SimPush

the error introduced by neglecting non-attention nodes is negligible and bounded within the error guarantee ϵ 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.

Discovery of attention nodes. First we clarify what qualifies a node as an attention node of query node u.

Definition 3. (Attention Nodes on Level ℓ). Given an input graph G and a query node $u \in G$, a node w is an attention node of u on the ℓ-th level, if and only if hitting probability $h^{(\ell)}(u, w) \geq \epsilon_h$, where $\epsilon_h = \frac{1-\sqrt{c}}{3\sqrt{c}} \cdot \epsilon$.

Parameter ϵ_h is explained in Lemma 4 towards the end of this subsection. Let $A_u^{(\ell)}$ denote the set of attention nodes on level ℓ, and A_u be the set of all attention nodes that appear in any level. Focusing on the attention nodes only, we employ the interpretation of SimRank $s(u, v)$ in Equation (2), and have the approximate $s'(u, v)$ in Equation (6). Lemma 1 provides the error guarantee for $s'(u, v)$ [1].

$$s'(u, v) = \sum_{\ell=0}^{+\infty} \sum_{w \in A_u^{(\ell)}} \kappa^{(\ell)}(u, v, w), \tag{6}$$

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

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

In the above definition of $s'(u, v)$, we enumerate all possible levels ℓ. 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 ϵ_h, the number of attention nodes with respect to u is at most $\left\lfloor \frac{\sqrt{c}}{(1-\sqrt{c}) \cdot \epsilon_h} \right\rfloor$. Meanwhile, all attention nodes exist within $L^* = \left\lfloor \log_{\frac{1}{\sqrt{c}}} \frac{1}{\epsilon_h} \right\rfloor$ 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 δ), 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 ℓ-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.

w_d°) denote non-attention nodes, which are used later in Section 4. Specifically, the propagation starts from u and traverses the graph in a level-wise manner, reaching nodes w_a, w_b, w_c on the first level, nodes w_d, w_e, w_f, w_g on the second level, and nodes w_h, w_p, w_c on the third level, which is the last level since $L = 3$. Note that a node can be visited multiple times on different levels, *e.g.*, w_c on both the first and third levels. In this case, it is also possible that a node is an attention node on one level (*e.g.*, w_c on Level 1) and non-attention node on another (*e.g.*, w_c on Level 3).

Estimation of $\kappa^{(\ell)}(u, v, w)$. After identifying attention nodes, SimPush needs to estimate each $\kappa^{(\ell)}(u, v, w)$, according to Equation (6). Existing solutions mostly estimate it by running numerous $\sqrt{c}$-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 ℓ-th level of G_u, where $\ell = 1, \ldots, L$, the last-meeting probability of w within G_u, $\gamma^{(\ell)}(w)$, is the probability that two $\sqrt{c}$-walks from w 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 $\gamma^{(\ell)}(w)$ has vital differences compared to the last-meeting probability $\eta(w)$ used in SLING and PRSim, explained in Section 2.2. First, $\gamma^{(\ell)}(w)$ is defined based on the attention sets and source graph G_u, instead of the whole graph. Second, $\gamma^{(\ell)}(w)$ 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, $\gamma^{(\ell)}(w)$ 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 $\gamma^{(\ell)}(w)$ of all attention nodes, without performing any $\sqrt{c}$-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.

$$s^+(u, v) = \sum_{\ell=1}^{L^*} \sum_{w \in A_u^{(\ell)}} h^{(\ell)}(u, w) \cdot \gamma^{(\ell)}(w) \cdot h^{(\ell)}(v, w), \quad (7)$$

where $A_u^{(\ell)}$ is the set of attention nodes at the ℓ-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 $\kappa^{(\ell)}(u, v, w)$, computed using hitting probabilities and last-meeting probabilities in G_u. The following lemma establishes the approximation bound for $s^+(u, v)$.

Table 3: Complexity of different stages in SimPush.

Stage	Time Complexity
Source-Push	$O(m \log \frac{1}{\epsilon} + \log \frac{1}{\epsilon\delta} / \epsilon^2)$
All $\gamma^{(\ell)}(w)$ computation	$O(m \log \frac{1}{\epsilon} / \epsilon + 1/\epsilon^3)$
Reverse-Push	$O(m \log \frac{1}{\epsilon})$

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

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

Reverse-Push. In Equation (7), it remains to clarify the computation of $h^{(\ell)}(v, w)$. Instead of estimating $h^{(\ell)}(v, w)$ independently (*e.g.*, by simulating random walks), we propose a novel Reverse-Push algorithm, detailed in Section 4.3, which estimates $h^{(\ell)}(u, w) \cdot \gamma^{(\ell)}(w) \cdot h^{(\ell)}(v, w)$ as a whole through residue push. Specifically, SimPush regards $r^{(\ell)}(w) = h^{(\ell)}(u, w) \cdot \gamma^{(\ell)}(w)$ as the initial residue of attention node w, and keeps pushing the residue to each node $v \in G$, following out-going edges, until ℓ 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 $h^{(3)}(u, w) \cdot \gamma^{(3)}(w) \cdot h^{(3)}(v_g, w)$ w.r.t., $r^{(3)}(w)$. The detailed push criteria is in Section 4.3.

Accordingly, our final SimRank estimate is

$$\tilde{s}(u, v) = \sum_{\ell=1}^{L^*} \sum_{w \in A_u^{(\ell)}} h^{(\ell)}(u, w) \cdot \gamma^{(\ell)}(w) \cdot \hat{h}^{(\ell)}(v, w), \quad (8)$$

where u and v are distinct nodes in G, $A_u^{(\ell)}$ is the ℓ-th level attention set. Here, the hitting probability $\hat{h}^{(\ell)}(v, w)$ 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 $h^{(\ell)}(u, w) \cdot \gamma^{(\ell)}(w) \cdot h^{(\ell)}(v, w)$ rather than the last term. Lemma 4 provides error guarantee for $\tilde{s}(u, v)$, and explains the value of ϵ_h.

Lemma 4. *Given distinct nodes u and v in G, error parameter ϵ, and decay factor c, when $\epsilon_h \leq \frac{1-\sqrt{c}}{3\sqrt{c}} \cdot \epsilon$, 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.

4. DETAILED SIMPUSH ALGORITHM

Algorithm 1 shows the main SimPush algorithm, consisting of three stages. With ϵ_h 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: $\tilde{s}(u, v)$ for each $v \in V$, w.r.t, query node $u$.
1 $\epsilon_h \leftarrow \frac{1-\sqrt{c}}{3\sqrt{c}} \cdot \epsilon$;
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 $A_u^{(\ell)}$ do
6         Compute $\gamma^{(\ell)}(w)$ with Algorithm 4;
7         $r^{(\ell)}(w) \leftarrow h^{(\ell)}(u, w) \cdot \gamma^{(\ell)}(w)$;
8 Invoke Algorithm 5 (Reverse-Push) to get $\tilde{s}(u, v)$ for each
    $v \in V$;
9 return $\tilde{s}(u, v)$ for each $v \in V$;
```

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 $\gamma^{(\ell)}(w)$ of all attention nodes w (Section 4.2), and finally invokes Reverse-Push (Section 4.3) to compute the single 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 $\left(2\cdot\log\frac{1}{(1-\sqrt{c})\epsilon_h\delta}/\epsilon_h^2\right)$ $\sqrt{c}$-walks from u, counts the visits of every node v at every l-th step, $H^{(l)}(u, v)$, and then identifies the max level L where there exists node v with $H^{(l)}(u, v) \geq \left(\log\frac{1}{(1-\sqrt{c})\epsilon_h\delta}/\epsilon_h^2\right)$, 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, $h^{(0)}(u, u)$ 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 ℓ-th level, it pushes and increases the $(\ell + 1)$-level hitting probability of every in-neighbor v' of v by $\frac{\sqrt{c}\cdot h^{(\ell)}(u,v)}{d_{\mathcal{I}}(v)}$ and adds edge from v' to v to 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 $O(m\log\frac{1}{\epsilon} + \log\frac{1}{\epsilon\delta}/\epsilon^2)$ 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.*

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_a and w_b in G_u, the hitting probability from w_a to w_b at the i-th step in G_u, is the probability that a $\sqrt{c}$-walk from w_a and walking in G_u, visits w_b at the i-th step, where $i \geq 0$.

```
Algorithm 2: Source-Push
   Input: Graph $G$, query $u$, decay factor $c$, parameter $\epsilon_h$
   Output: Source graph $G_u$ and attention node sets $A_u^{(\ell)}$
           for $\ell = 1, ..., L$.
 1 $H^{(l)}(u, v) \leftarrow 0$, for $v \in V$ and $l = 1, 2, ...$;
 2 for $i = 1, ..., \left(2\cdot\log\frac{1}{(1-\sqrt{c})\epsilon_h\delta}/\epsilon_h^2\right)$ do
 3     Generate a $\sqrt{c}$-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 $H^{(l)}(u, v) \geq \log\frac{1}{(1-\sqrt{c})\epsilon_h\delta}/\epsilon_h^2$ then
 7         $L \leftarrow l$;
 8 $L \leftarrow \min(L, L^*)$;
 9 $h^{(\ell)}(u, v) \leftarrow 0$ for $\ell = 1, ..., L$ 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 $v' \in \mathcal{I}(v)$ do
15             $h^{(\ell+1)}(u, v') \leftarrow h^{(\ell+1)}(u, v') + \frac{\sqrt{c}\cdot h^{(\ell)}(u,v)}{d_{\mathcal{I}}(v)}$;
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 $\ell = 1, ..., L$ do
21     Insert $w$ in $G_u$ with $h^{(\ell)}(u, w) \geq \epsilon_h$ into $A_u^{(\ell)}$;
```

Hereafter, we use $\tilde{h}^{(\ell)}(*, *)$ to denote the hitting probabilities in G_u, and use $h^{(\ell)}(*, *)$ to represent the hitting probabilities in G. For query node u, every $h^{(\ell)}(u, w)$ computed by Source-Push over G can be reproduced by pushing u over G_u, *i.e.*, $\tilde{h}^{(\ell)}(u, w)$ is the same as $h^{(\ell)}(u, w)$. For the ease of presentation, in the following sections, we denote w, w_i, and w_j as nodes at the ℓ-th, $(\ell + i)$-th, $(\ell + j)$-th levels of G_u respectively, and w, w_i, w_j are attention nodes by default, unless otherwise specified.

4.2 Last-Meeting Correction within G_u

As mentioned, given query u with attention sets $A_u^{(\ell)}$, SimPush computes last-meeting probability $\gamma^{(\ell)}(w)$ for each $w \in A_u^{(\ell)}$ in the source graph G_u (Definition 4). Utilizing G_u, we design a method that computes $\gamma^{(\ell)}(w)$ for *all* attention nodes in G_u without generating any $\sqrt{c}$-walks, in $O(m\log\frac{1}{\epsilon}/\epsilon + 1/\epsilon^3)$ time. We first clarify the formula to compute $\gamma^{(\ell)}(w)$, and then present the detailed algorithms.

Formula to compute $\gamma^{(\ell)}(w)$. Given attention nodes w and w_i, we define the i-step first-meeting probability $\rho^{(i)}(w, w_i)$ in G_u as follows.

Definition 6. (First-meeting probability in G_u). Given attention nodes w and w_i at the ℓ-th and $(\ell + i)$-th levels of G_u respectively, where $\ell = 1, ..., L$ and $0 < i \leq L - \ell$, $\rho^{(i)}(w, w_i)$ is the probability that two $\sqrt{c}$-walks from w walking in G_u first meet at attention node w_i.

Note that in Definition 6, it is allowed that the two walks first meet at some non-attention node in G_u, before meeting

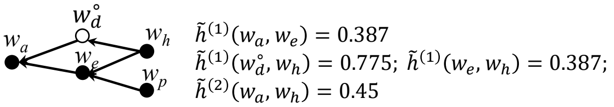

w_d°
w_a
w_e
w_h
w_p
$\tilde{h}^{(1)}(w_a, w_e) = 0.387$
$\tilde{h}^{(1)}(w_d^\circ, w_h) = 0.775$; $\tilde{h}^{(1)}(w_e, w_h) = 0.387$;
$\tilde{h}^{(2)}(w_a, w_h) = 0.45$

Figure 2: Hitting probabilities in a subgraph of G_u in Figure 1(a).

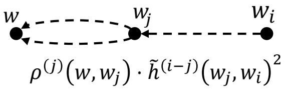


Figure 3: Non-first-meeting probability from attention nodes w to w_i via w_j.

at w_i. In this section, when we say that two walks first meet, it means that the two walks first meet at an attention node in G_u. According to Definitions 4 and 6, we have

$$\gamma^{(\ell)}(w) = 1 - \sum_{i=1}^{L-\ell} \sum_{w_i \in A_u^{(\ell+i)}} \rho^{(i)}(w, w_i), \tag{9}$$

where $A_u^{(\ell+i)}$ is the $(\ell+i)$-th level attention set and $\ell \leq L$. Now, the problem reduces to computing $\rho^{(i)}(w, w_i)$ in G_u. This requires the hitting probabilities between attention nodes within G_u (Definition 5), to be clarified soon.

When $i = 1$, $\rho^{(1)}(w, w_1)$ is nonzero only if attention node w_1 is an in-neighbor of w in G_u (obviously w_1 is at the $(\ell+1)$-th level of G_u). Given the 1-step hitting probability $\tilde{h}^{(1)}(w, w_1)$, the probability of two independent $\sqrt{c}$-walks from w walking in G_u and meeting at w_1 is $\tilde{h}^{(1)}(w, w_1)^2$. Further, since there is only one step from w to w_1, $\rho^{(1)}(w, w_1)$ is exactly $\tilde{h}^{(1)}(w, w_1)^2$, *i.e.*,

$$\forall w_1 \in A_u^{(\ell+1)}, \rho^{(1)}(w, w_1) = \tilde{h}^{(1)}(w, w_1)^2, \tag{10}$$

where $A_u^{(\ell+1)}$ is the $(\ell+1)$-th level attention set. For example, in Figure 2, $\rho^{(1)}(w_a, w_e) = \tilde{h}^{(1)}(w_a, v_e)^2 = 0.150$.

When $2 \leq i \leq L - \ell$, we compute $\rho^{(i)}(w, w_i)$ by utilizing $\rho^{(j)}(w, w_j)$ of the attention nodes w_j between w and w_i in G_u, where $1 \leq j \leq i-1$. Suppose that two $\sqrt{c}$-walks from w walking in G_u first meet at w_j and then meet at w_i. The *non-first-meeting probability* from w to w_i via w_j is $\rho^{(j)}(w, w_j) \cdot \tilde{h}^{(i-j)}(w_j, w_i)^2$. Figure 3 illustrates this concept, where first-meeting probability $\rho^{(j)}(w, w_j)$ is represented by two dashed lines, and meeting probability $\tilde{h}^{(i-j)}(w_j, w_i)^2$ is represented by one dashed line. Therefore, $\rho^{(i)}(w, w_i)$ equals the meeting probability from w to w_i, *i.e.*, $\tilde{h}^{(i)}(w, w_i)^2$, subtracted by all the non-first-meeting probabilities from w to w_i via any attention node w_j between w and w_i, *i.e.*,

$$\begin{aligned} \rho^{(i)}(w, w_i) &= \tilde{h}^{(i)}(w, w_i)^2 \\ &- \sum_{j=1}^{i-1} \sum_{w_j \in A_u^{(\ell+j)}} \rho^{(j)}(w, w_j) \cdot \tilde{h}^{(i-j)}(w_j, w_i)^2, \end{aligned} \tag{11}$$

where $i = 2, ..., L - \ell$. For example, in Figure 2, $\rho^{(2)}(w_a, w_h) = h^{(2)}(w_a, w_h)^2 - \rho^{(1)}(w_a, w_e) \cdot h^{(1)}(w_e, w_h)^2 = 0.45^2 - 0.15 \cdot 0.387^2 = 0.18$. w_d° is not considered since it is a non-attention node.

Hitting probabilities between attention nodes in G_u. Now we focus on computing hitting probabilities in G_u.

Algorithm 3: Hitting probabilities in G_u

```
   Input: Source graph $G_u$
   Output: All nonzero hitting probabilities between
           attention nodes in $G_u$
1  for $\ell = L, ..., 2$ do
2      for each attention node $w$ at the $\ell$-th level of $G_u$ do
3          $\tilde{h}^{(0)}(w, w) \leftarrow 1$;
4      for each node $w'$ at the $\ell$-th level of $G_u$ do
5          for each nonzero $\tilde{h}^{(i)}(w', w_i)$ do
6              for each $w'_a \in \mathcal{O}^T(w')$ at $(\ell-1)$-th level do
7                  $\tilde{h}^{(i+1)}(w'_a, w_i) \leftarrow$
                       $\tilde{h}^{(i+1)}(w'_a, w_i) + \frac{\sqrt{c}}{d_{\mathcal{I}}(w'_a)} \cdot \tilde{h}^{(i)}(w', w_i)$
8  return
```

Given nodes w and w_i (here w can be a non-attention node), $\tilde{h}^{(i)}(w, w_i)$ is computed by aggregating the hitting probabilities $\tilde{h}^{(i-1)}(w', w_i)$ from w's in-neighbors w' to w_i, as follows.

$$\tilde{h}^{(i)}(w, w_i) = \frac{\sqrt{c}}{d_{\mathcal{I}}^T(w)} \sum_{w' \in \mathcal{I}^T(w)} \tilde{h}^{(i-1)}(w', w_i), \tag{12}$$

where $\mathcal{I}^T(w)$ is the set of in-neighbors of w in G_u and $d_{\mathcal{I}}^T(w)$ is the indegree of w in G_u, and $i \geq 1$. For example, in Figure 2, $\tilde{h}^{(2)}(w_a, w_h) = \frac{\sqrt{c}}{2} \cdot \left(\tilde{h}^{(1)}(w_d^\circ, w_h) + \tilde{h}^{(1)}(w_e, w_h)\right) = 0.45$. 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.*, $\tilde{h}^{(1)}(w_d^\circ, w_h)$ 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 $\tilde{h}^{(0)}(w, w)$ 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 ℓ-th level (including non-attention nodes), if it has nonzero hitting probabilities $\tilde{h}^{(i)}(w', w_i)$ to any attention node w_i at the $(\ell+i)$-th level for $i = 0, ..., L - \ell$, each of the probabilities $\tilde{h}^{(i)}(w', w_i)$ 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 ℓ-th level of G_u. At Line 1, $\gamma^{(\ell)}(w)$ is initialized to 1. At Lines 2-4, when $i = 1$, all nonzero $\rho^{(1)}(w, w_1)$ are computed according to Equation (10), and are subtracted from $\gamma^{(\ell)}(w)$, based on Equation (9). Then all first-meeting probabilities $\rho^{(i)}$ for $2 \leq i \leq \Delta l$ are computed level by level from Lines 5 to 11, using Equation (11). Specifically, every $\rho^{(i)}(w, w_i)$ is initialized as $\tilde{h}^{(i)}(w, w_i)^2$ at Lines 6-7 and is subtracted by all non-first meeting probabilities from w to w_i via w_j at Lines 8-11. Whenever the first-meeting probabilities $\rho^{(i)}(w, w_i)$ for attention nodes $w_i \in A_u^{(\ell+i)}$ are ob-

Algorithm 4: Last-Meeting Probability

```
   Input: Source graph $G_u$; attention node $w \in A_u^{(\ell)}$;
   Output: Last-meeting probability $\gamma^{(\ell)}(w)$ in $G_u$
1  $\gamma^{(\ell)}(w) \leftarrow 1$; $\Delta l \leftarrow L - \ell$;
2  for each attention node $w_1$ with nonzero $\tilde{h}^{(1)}(w, w_1)$ do
3      $\rho^{(1)}(w, w_1) \leftarrow \tilde{h}^{(1)}(w, w_1)^2$;
4  $\gamma^{(\ell)}(w) \leftarrow \gamma^{(\ell)}(w) - \sum_{w_1 \in A_u^{(\ell+1)}} \rho^{(1)}(w, w_1)$;
5  for $i = 2$ to $\Delta l$ do
6      for each attention node $w_i$ with $\tilde{h}^{(i)}(w, w_i) > 0$ do
7          $\rho^{(i)}(w, w_i) \leftarrow \tilde{h}^{(i)}(w, w_i)^2$;
8      for $j = 1$ to $i - 1$ do
9          for each nonzero $\rho^{(j)}(w, w_j)$ of each attention
             node $w_j$ at the $(\ell + j)$-th level of $G_u$ do
10             for each nonzero $\tilde{h}^{(i-j)}(w_j, w_i)$ do
11                 $\rho^{(i)}(w, w_i) \leftarrow$
                     $\rho^{(i)}(w, w_i) - \rho^{(j)}(w, w_j) \cdot \tilde{h}^{(i-j)}(w_j, w_i)^2$;
12     $\gamma^{(\ell)}(w) \leftarrow \gamma^{(\ell)}(w) - \sum_{w_i \in A_u^{(\ell+i)}} \rho^{(i)}(w, w_i)$;
13 return $\gamma^{(\ell)}(w)$;
```

tained, they are subtracted from $\gamma^{(\ell)}(w)$ at Line 12, according to Equation (9). Finally, $\gamma^{(\ell)}(w)$ is returned. Lemma 6 presents the time complexity of Algorithm 3, Algorithm 4, and the second stage of SimPush as a whole.

LEMMA 6. *Algorithm 3 runs in $O(m \log \frac{1}{\epsilon} / \epsilon)$ time, and Algorithm 4 runs in $O(1/\epsilon^2)$ for a single $\gamma^{(\ell)}(w)$, and there are $O(1/\epsilon)$ attention nodes. Therefore, the overall time complexity for last-meeting computation is $O(m \log \frac{1}{\epsilon} / \epsilon + 1/\epsilon^3)$.*

4.3 Reverse-Push

Given w in $A_u^{(\ell)}$ with its $\gamma^{(\ell)}(w)$ obtained, we regard $r^{(\ell)}(w) = h^{(\ell)}(u, w) \cdot \gamma^{(\ell)}(w)$ as the residue of w. Aiming to estimate $h^{(\ell)}(u, w) \cdot \gamma^{(\ell)}(w) \cdot \hat{h}^{(\ell)}(v, w)$ as a whole, we propose Reverse-Push that propagates the residue over graph G, following the out-going edges of every encountered node.

In this section, we call the $(\ell - 1)$-th level as the next level of the ℓ-th level. At current ℓ-th level, by pushing initial residue $r^{(\ell)}(w)$ to the out-neighbors v of w in G, nodes v accumulate residue $r^{(\ell-1)}(v)$ at the $(\ell - 1)$-th level. Then, Reverse-Push goes to the next level to push. After ℓ iterations, we have many nonzero $r^{(0)}(v)$. Then $r^{(0)}(v)$ estimates $h^{(\ell)}(u, w) \cdot \gamma^{(\ell)}(w) \cdot \hat{h}^{(\ell)}(v, w)$ with respect to $r^{(\ell)}(w)$, and thus, $r^{(0)}(v)$ is added to $\tilde{s}(u, v)$. 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_u and w' with $r^{(2)}(w')$ at the 2-nd 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 it 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 $\tilde{s}(u, v)$ are initialized to zeros for $v \in V$. At Line 2, the initial residue of each attention node w is $r^{(\ell)}(w)$, and the residues of all other nodes at all levels are zeros by default. Starting from level $\ell' = L$ to 1, for every node v' with residue $r^{(\ell')}(v')$ that satisfies $\sqrt{c} \cdot r^{(\ell')}(v') \geq \epsilon_h$, we propagate its residue to its out-neighbors v (Lines 3-5). If $\ell' > 1$, residue $r^{(\ell'-1)}(v)$ is increased by $\frac{\sqrt{c} \cdot r^{(\ell')}(v')}{d_{\mathcal{I}}(v)}$, where $d_{\mathcal{I}}(v)$ is the indegree of v in G (Lines 6-7); if ℓ' is 1, which means v is at the 0-th level, $\tilde{s}(u, v)$ is increased by $\frac{\sqrt{c} \cdot r^{(\ell')}(v')}{d_{\mathcal{I}}(v)}$ at Line 9. Finally, $\tilde{s}(u, u)$ is set to 1 and all SimRank values $\tilde{s}(u, v)$ for all $v \in V$ are returned at Lines 10-11. The time complexity of Algorithm 5 is presented in Lemma 7.

Algorithm 5: Reverse-Push

```
   Input: Residues $r^{(\ell)}(w)$ of all attention nodes
   Output: $\tilde{s}(u, v)$ for $v \in V$
1  $\tilde{s}(u, v) \leftarrow 0$ for $v \in V$;
2  $r^{(\ell')}(v) \leftarrow 0$, for $\ell' = 1, ..., L$ and $v \in V$, except the initial
     residues $r^{(\ell)}(w)$ of all attention nodes $w$;
3  for $\ell' = L, ..., 1$ do
4      for each $v'$ with $\sqrt{c} \cdot r^{(\ell')}(v') \geq \epsilon_h$ do
5          for each $v \in \mathcal{O}(v')$ do
6              if $\ell' - 1 > 0$ then
7                  $r^{(\ell'-1)}(v) \leftarrow r^{(\ell'-1)}(v) + \frac{\sqrt{c} \cdot r^{(\ell')}(v')}{d_{\mathcal{I}}(v)}$;
8              else
9                  $\tilde{s}(u, v) \leftarrow \tilde{s}(u, v) + \frac{\sqrt{c} \cdot r^{(\ell')}(v')}{d_{\mathcal{I}}(v)}$
10 $\tilde{s}(u, u) \leftarrow 1$;
11 return $\tilde{s}(u, v)$ for $v \in V$;
```

LEMMA 7. *Algorithm 5 runs in $O(m \log \frac{1}{\epsilon})$ 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 $\tilde{s}(u, v)$ that satisfies $s(u, v) - \tilde{s}(u, v) \leq \epsilon$ for each node v in G, with at least $1 - \delta$ probability, where $s(u, v)$ is the exact SimRank value between u and v.*

THEOREM 2. *In expectation, Algorithm 1 runs in $O(m \log \frac{1}{\epsilon} / \epsilon + \log \frac{1}{\epsilon \delta} / \epsilon^2 + 1/\epsilon^3)$ time.*

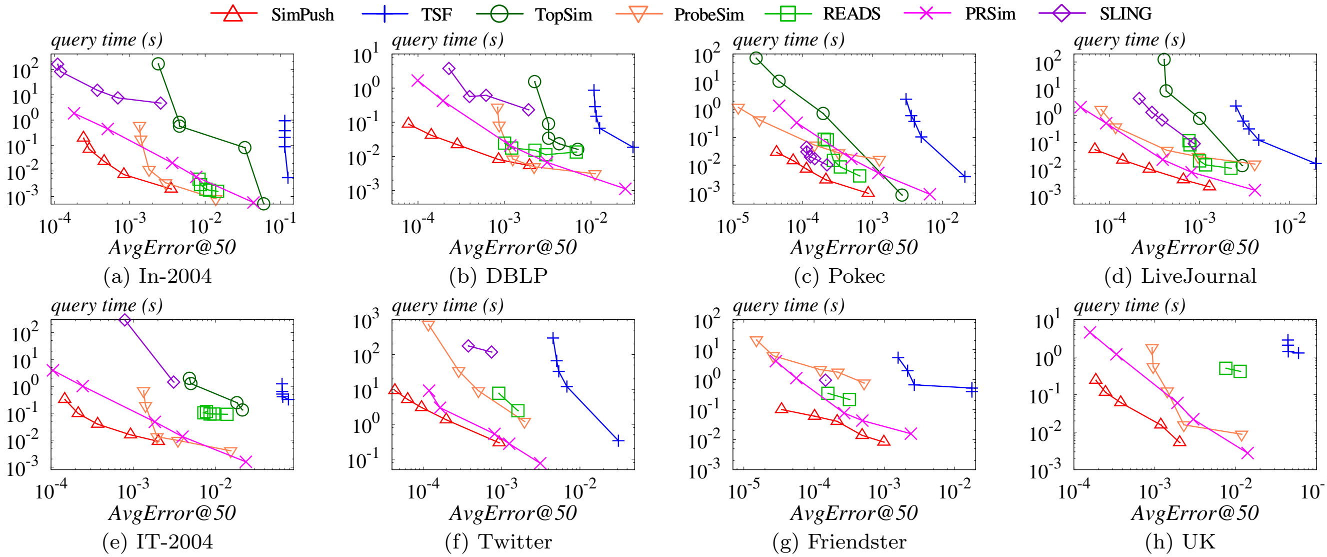
5. EXPERIMENTS

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 [32], READS [12], TopSim [15], SLING [30], 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



(a) In-2004 (b) DBLP (c) Pokec (d) LiveJournal

(e) IT-2004 (f) Twitter (g) Friendster (h) UK

Figure 4: Average error *vs.* Query time

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
IT-2004	41,291,594	1,135,718,909	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

million edges: In-2004, DBLP, Pokec, and LiveJournal. All datasets are available at [4,16,27]. For each dataset, we generate 100 queries by selecting nodes uniformly at random.

Parameters. Following [21, 30, 32], we set the decay factor c to 0.6, and fix the failure probability $\delta = 0.0001$. For SimPush, we vary ϵ in $\{0.05, 0.02, 0.01, 0.005, 0.002\}$. We set the parameters of all competitors following the settings in [32]. In particular, PRSim has two parameters: ϵ_a, the absolute error threshold, and j_0, the number of hub nodes. We vary ϵ_a in $\{0.5, 0.1, 0.05, 0.01, 0.005\}$, and set j_0 to $\sqrt{n}$ by default [32]. 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 $\sqrt{c}$-walks generated for each node in preprocessing stage, and t, the maximum depth of the $\sqrt{c}$-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; $1/h$, the minimal degree threshold to identify a high degree node; η, the similarity threshold for trimming a random walk; H, the number of random walks to be expanded at each level. We fix H and η to their default values, *i.e.*, 100 and 0.001, respectively. We vary $(T, 1/h)$ in $\{(1, 10), (3, 100), (3, 1000), (3, 10000), (4, 10000)\}$. SLING has a parameter ϵ_a, which denotes the upper bound on the absolute error. We vary ϵ_a in $\{0.5, 0.1, 0.05, 0.01, 0.005\}$. ProbeSim also has an absolute error threshold ϵ_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 on 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, 32]. 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, we adopt the *pooling* method [21, 32] to generate ground truth. 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 of $s(u, v)$ by Monte Carlo. The ground truth top-k node set V_k is then the set of k nodes with highest SimRank values from the pool.

Metrics. We adopt two metrics for accuracy evaluation, *i.e.*, *AvgError@k* and *Precision@k* [32]. We also evaluate the peak memory usage. *AvgError@k* is the average absolute error for approximating $s(u, v_i)$ for each node v_i in the ground truth top-k nodes V_k. For each node v_i in V_k, let $\hat{s}(u, v_i)$ be the estimation of $s(u, v_i)$, *AvgError@k* is:

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

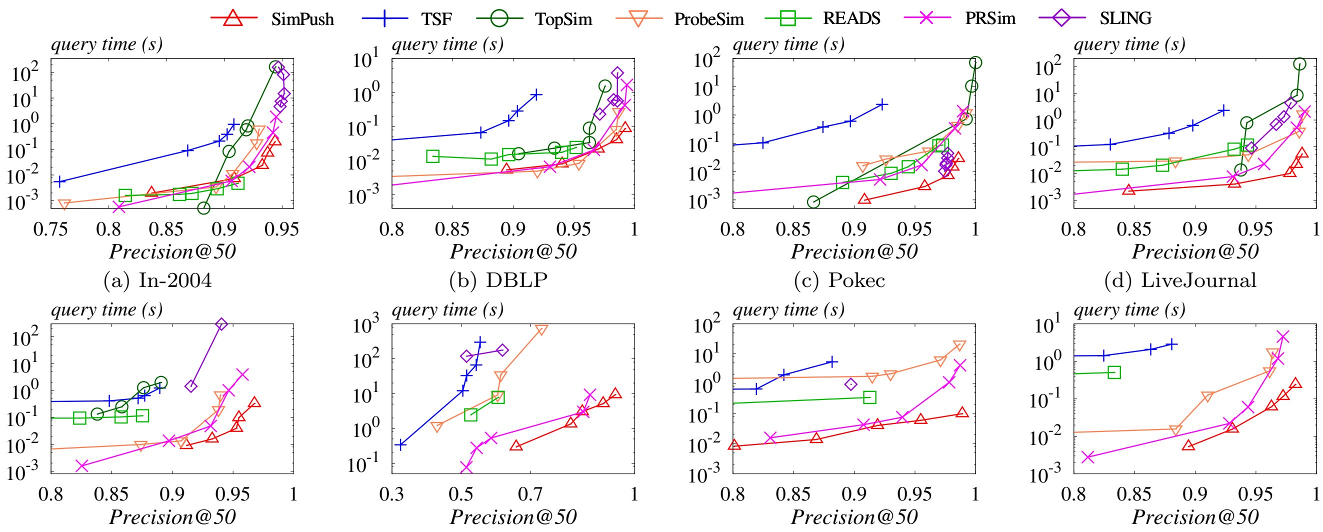
Precision@k evaluates the ability to return the top-k nodes for a query in terms of ground truth top-k node set V_k. Suppose that $V'_k = \{v'_1, \cdots, v'_k\}$ is the top-k nodes returned by the algorithm to be evaluated. *Precision@k* is defined as

$$Precision@k = |V_k \cap V'_k|/k.$$

Peak memory usage. We enquiry Linux system resource files for *rusage.ru_maxrss*, 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,



(a) In-2004 (b) DBLP (c) Pokec (d) LiveJournal

(e) IT-2004 (f) Twitter (g) Friendster (h) UK

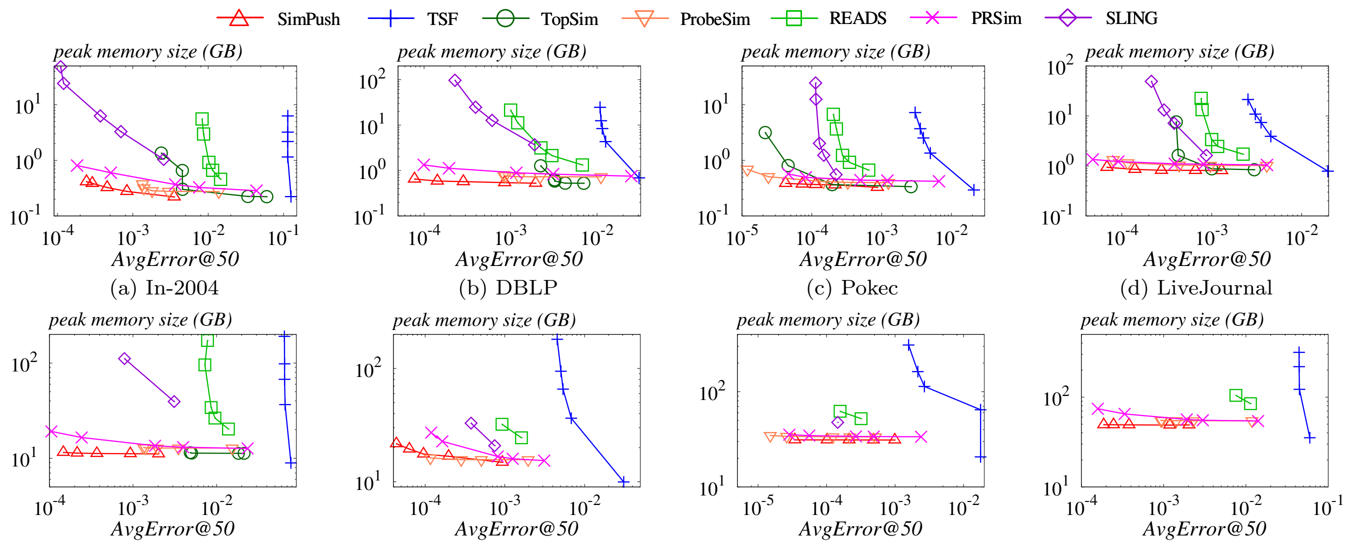
Figure 5: Precision *vs.* query time.

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 second(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 3.5×10^{-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 9×10^{-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 [32], SimPush also outperforms PRSim by a significant gap. To achieve 1.4×10^{-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 $\epsilon = 0.02$, 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 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 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, 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 corresponding to its 5 parameter settings, from right to left. We find that (i) the peak memory usage of SimPush is lower than all competitors over all datasets under almost all settings; (ii) the peak memory usage of SimPush is insensitive to ϵ. The reason is that when decreasing ϵ, the size of G_u and the number of attention nodes increase slowly, and thus,



(a) In-2004 (b) DBLP (c) Pokec (d) LiveJournal

(e) IT-2004 (f) Twitter (g) Friendster (h) UK

Figure 6: Average error *vs.* peak memory usage.

SimPush can maintain relatively stable peak memory usage. For instance, in Figure 6(h) for UK graph, SimPush requires 48 to 49 GB memory, while ProbeSim needs about 54 GB and PRSim needs 54 to 74 GB. Methods SLING, READS, TSF require much more memory and are sensitive to parameters.

Results on Billion-Node ClueWeb. Figure 7 reports the evaluation results on the ClubWeb dataset. 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 [30], ProbeSim [21], READS [12], TSF [28], TopSim [15] and PRSim [32], 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]:

$$\mathbf{S} = (c\mathbf{P}^\top \mathbf{S}\mathbf{P}) \vee \mathbf{I}, \tag{13}$$

where $\mathbf{S}$ is an $n \times n$ matrix such that $\mathbf{S}[i,j]$ is the SimRank value between the i-th and j-th nodes, $\vee$ is the element-wise maximum operator, $\mathbf{P}$ and $\mathbf{I}$ are the transition matrix and identity matrix of the input graph G. Power method starts with $\mathbf{S} = \mathbf{I}$, and then updates $\mathbf{S}$ iteratively based on Equation (13), until all elements in $\mathbf{S}$ converge. Subsequent studies [22, 31, 36, 38] improve the Power method in terms of efficiency or accuracy. However, all these methods incur $O(n^2)$ space overhead, which is prohibitively expensive for web-scale graphs. It is not straightforward to directly apply these methods for single-source SimRank queries. There are studies [7, 8, 14, 17, 33–35] that attempt to address the inefficiency issue caused by the operator $\vee$ in Equation (13), via an alternative formula for SimRank:

$$\mathbf{S} = c\mathbf{P}^\top \mathbf{S}\mathbf{P} + (1-c)\cdot \mathbf{I}. \tag{14}$$

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, 30]. 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, 32]. 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, 37, 39] on variants of SimRank, and SimRank similarity join [24, 29, 40]. However, these solutions are inapplicable for single-source SimRank queries.

7. CONCLUSION

We propose SimPush, an 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 evaluation on real-world web-scale graphs. In the future, we plan to study SimRank queries with relative error guarantees, batch processing, as well as computation on new hardware.

8. ACKNOWLEDGMENTS

This work is supported by the National University of Singapore under SUG grant R-252-000-686-133. This publication was made possible by NPRP grant #NPRP10-0208-170408 from the Qatar National Research Fund (a member of Qatar Foundation). The findings herein reflect the work, and are solely the responsibility, of the authors.

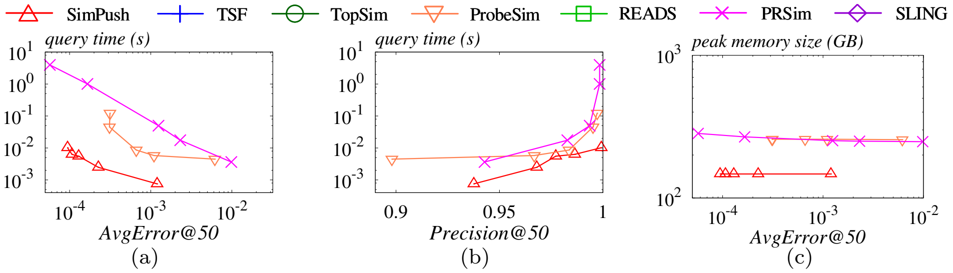


Figure 7: Evaluation on Billion-Node Clueweb

APPENDIX

LEMMA 8. *$\sum_i X_i = N$ and for all i, $X_i \in (0, \epsilon_h)$. $\sum_i Y_i = M$, $M \in (0,1)$ and for all i, $Y_i > 0$. Then $\sum_i X_i Y_i \leq \epsilon_h M$.*

PROOF: $\sum_i X_i Y_i \leq \max_i X_i \cdot \sum_i Y_i \leq \epsilon_h M$ □

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

Proof of Lemma 2. At level ℓ, $\sum_{w \in G_u^{(\ell)}} h^{(\ell)}(u,w) = \sqrt{c}^{\ell}$. Hence, at level ℓ, there exists at most $\lfloor \frac{\sqrt{c}^{\ell}}{\epsilon_h} \rfloor$ attention nodes, and for $\ell > L^*$, $w \in G_u^{\ell}$, $h^{(\ell)}(u,w) \leq \epsilon_h$. Therefore, the size of attention set A_u is at most $\sum_{\ell=1} \lfloor \frac{\sqrt{c}^{\ell}}{\epsilon_h} \rfloor \leq \lfloor \frac{\sqrt{c}}{(1-\sqrt{c})\cdot\epsilon_h} \rfloor$. □

Proof of Lemma 3. $f^{(\ell)}(u,v,w)$ is the *ℓ-th* step first meeting probability at w, and we can write $s(u,v)$ as $s(u,v) = \sum_{\ell=1}^{\infty} \sum_{w \in V} f^{(\ell)}(u,v,w)$. Given $A_u^{(\ell)}$, let $s_1(u,v) = \sum_{\ell=1}^{L^*} \sum_{w \in A_u^{(\ell)}} f^{(\ell)}(u,v,w)$, and $s_2(u,v) = \sum_{\ell=1}^{\infty} \sum_{w \notin A_u^{(\ell)}} f^{(\ell)}(u,v,w)$. Obviously, $s(u,v) = s_1(u,v) + s_2(u,v)$. We want to prove $s^+(u,v) \geq s_1(u,v) - s_2(u,v)$ and $s_2(u,v) \leq \frac{\epsilon_h \sqrt{c}}{1-\sqrt{c}}$. From Eq. (9),

$$\begin{aligned} s^+(u,v) = &\sum\nolimits_{\ell=1}^{L^*} \sum\nolimits_{w \in A_u^{(\ell)}} h^{(\ell)}(u,w) h^{(\ell)}(v,w) \\ &\times (1 - \sum\nolimits_{i=1}^{(L^*-\ell)} \sum\nolimits_{w_i \in A_u^{(\ell+i)}} \rho^{(i)}(w,w_i)) \\ = &\sum\nolimits_{\ell=1}^{L^*} \sum\nolimits_{w \in A_u^{(\ell)}} [h^{(\ell)}(u,w) h^{(\ell)}(v,w) \\ &- \sum_{\ell'=1}^{\ell-1} \sum_{w_a \in A_u^{(\ell')}} h^{(\ell')}(u,w_a) h^{(\ell')}(v,w_a) \rho^{(\ell-\ell')}(w_a,w)] \end{aligned} \tag{15}$$

$f^{(\ell)}(u,v,w) = h^{(\ell)}(u,w) h^{(\ell)}(v,w) - \sum_{\ell':\ell'<\ell} \sum_{w' \in G_u^{(\ell')}} f^{(\ell')}(u,v,w') h^{(\ell')}(w',w)^2$

Here we only consider $w' \in A_u^{(\ell')}$, i.e., $\hat{f}^{(\ell)}(u,v,w) =$

$$h^{(\ell)}(u,w) h^{(\ell)}(v,w) - \sum_{\ell'<\ell} \sum_{w' \in A_u^{(\ell')}} f^{(\ell')}(u,v,w') h^{(\ell')}(w',w)^2 \tag{16}$$

Consider the probability that two $\sqrt{c}$-walks from u and v, first meet at attention node w' then meet at attention node w. Given two events: (i) two $\sqrt{c}$-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 event corresponds to the last line Eq. (15). Let $\hat{s}_1(u,v) = \sum_{\ell=1}^{L^*} \sum_{w \in A_u^{(\ell)}} \hat{f}^{(\ell)}(u,v,w)$. Thus, $s^+(u,v) = \hat{s}_1(u,v)$. $s_1(u,v) - \hat{s}_1(u,v)$ is the probability that two $\sqrt{c}$ walks first meet at non-attention node, then meet at attention node. Thus, $s_1(u,v) - \hat{s}_1(u,v) \leq s_2(u,v)$.

Now we prove $s_2(u,v) \leq (\epsilon_h \sqrt{c})/(1-\sqrt{c})$. Based on Lemma 8, $s_2(u,v) = \sum_{\ell=1} \sum_{w \in G_u^{(\ell)} \setminus A_u^{(\ell)}} f^{(\ell)}(u,v,w) \leq \sum_{\ell=1} \sum_{w \in G_u^{(\ell)} \setminus A_u^{(\ell)}} h^{(\ell)}(v,w) h^{(\ell)}(u,w) \leq \frac{\sqrt{c}\cdot\epsilon_h}{1-\sqrt{c}}$, Thus, $s(u,v) \geq s^+(u,v) \geq \hat{s}_1(u,v) \geq s_1(u,v) - s_2(u,v) \geq s(u,v) - 2s_2(u,v) \geq s(u,v) - \frac{2\sqrt{c}\epsilon_h}{1-\sqrt{c}}$. □

Proof of Lemma 4. In Algorithm 5, consider the lose of Simrank at level ℓ. Similar to prove Lemma 1, the lose at level $\ell \leq \epsilon_h \cdot \sqrt{c}^{\ell}$. Summing up all levels, the total loss is $\leq \frac{\epsilon_h \cdot \sqrt{c}}{1-\sqrt{c}}$. Thus $s(u,v) - \tilde{s}(u,v) \leq \frac{3\epsilon_h \sqrt{c}}{1-\sqrt{c}} \leq \epsilon$. □

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 $O(m \log \frac{1}{\epsilon})$. Let $\hat{h}^{(\ell)}(u,w)$ be the Monte Carlo estimation of $h^{(\ell)}(u,w)$. From Hoeffding bound [9], $\Pr(\hat{h}^{(\ell)}(u,w) \geq h^{(\ell)}(u,w) - \epsilon_h/2) \geq 1 - \exp[-2(\epsilon_h/2)^2 \cdot 2\log \frac{1}{(1-\sqrt{c})\epsilon_h \delta} / \epsilon_h^2] \geq 1 - (1-\sqrt{c})\epsilon_h \delta$. Since attention nodes are at most $\lfloor \frac{\sqrt{c}}{(1-\sqrt{c})\cdot\epsilon_h} \rfloor$, applying union bound, with probability at least $1-\delta$, G_u contains all nodes u with $h^{(\ell)}(u,w) \geq \epsilon_h$, for all ℓ. The expected time of MC is $O(\log \frac{1}{\epsilon\delta} / \epsilon^2)$. □

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 $O(m \log \frac{1}{\epsilon} / \epsilon)$. Let Z_i be the number of nodes in G_u at level i. For all $w_1 \in G_u^{(\ell+1)}$, the cost of all $\rho^{(1)}(w,w_1)$ is $O(Z_{l+1})$. For all $w_2 \in G_u^{(\ell+2)}$, from Eq. (9), the cost of all $\rho^{(2)}(w,w_2)$ is $O(Z_{\ell+1} Z_{\ell+2})$. Similarly, we can compute all $w_i \in G_u^{(\ell+i)}$ for all $i > 0$ in $O(Z_{\ell+1} + \sum_{i=1} \sum_{j=1} Z_{\ell+i} Z_{\ell+i+j})$ time. $Z_{\ell+1} + \sum_{i=1} \sum_{j=1} Z_{\ell+i} Z_{\ell+i+j} \leq (\sum_{i=1} Z_i)^2$ and $\sum_{i=1} Z_i \leq O(1/\epsilon)$, then $O(Z_{\ell+1} + \sum_{i=1} \sum_{j=1} Z_{\ell+i} Z_{\ell+i+j}) \leq O(1/\epsilon^2)$. Thus the cost of computing $\gamma^{(\ell)}(w)$ for all attention nodes $O(\frac{1}{\epsilon^3})$. The total complexity is $\max\{\frac{m \log \epsilon}{\epsilon}, \frac{1}{\epsilon^3}\}$. □

Proof of Lemma 7. Algorithm 5 costs $O(m)$ per level and have $O(\log \frac{1}{\epsilon})$ levels. The total cost is $O(m \log \frac{1}{\epsilon})$. □

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

9. 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. Csalogány, and T. Sarlós. Link-based similarity search to fight web spam. In *AIRWEB*, pages 9–16, 2006.

[3] A. D. Broido 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. Rácz. Scaling link-based similarity search. In *WWW*, pages 641–650, 2005.

[6] 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.

[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. Tomeo, 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] W. Tao, M. Yu, and G. Li. Efficient top-k simrank based similarity join. *PVLDB*, 8(3):317–328, 2014.

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

[31] Y. Wang, X. Lian, and L. Chen. Efficient simrank tracking in dynamic graphs. In *ICDE*, page 545, 2018.

[32] 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.

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

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

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

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

[37] W. Yu and J. A. McCann. High quality graph-based similarity search. In *SIGIR*, pages 83–92, 2015.

[38] 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.

[39] P. Zhao, J. Han, and Y. Sun. P-rank: a comprehensive structural similarity measure over information networks. In *CIKM*, pages 553–562, 2009.

[40] W. Zheng, L. Zou, Y. Feng, L. Chen, and D. Zhao. Efficient simrank based similarity join over large graphs. *PVLDB*, 6(7):493–504, 2013.