Massively Parallel Single-Source SimRanks in $o(\log n)$ Rounds

Siqiang Luo[1*], **Zulun Zhu**[1*]
[1]Nanyang Technological University
{siqiang.luo, ZULUN001}@ntu.edu.sg

Abstract

SimRank is one of the most fundamental measures that evaluate the structural similarity between two nodes in a graph and has been applied in a plethora of data mining and machine learning tasks. These tasks often involve single-source SimRank computation that evaluates the SimRank values between a source node u and all other nodes. Due to its high computation complexity, single-source SimRank computation for large graphs is notoriously challenging, and hence recent studies resort to distributed processing. To our surprise, although SimRank has been widely adopted for two decades, theoretical aspects of distributed SimRanks with provable results have rarely been studied. In this paper, we conduct a theoretical study on single-source SimRank computation in the Massive Parallel Computation (MPC) model, which is the standard theoretical framework modeling distributed systems. Existing distributed SimRank algorithms enforce either $\Omega(\log n)$ communication round complexity or $\Omega(n)$ machine space for a graph of n nodes. We overcome this barrier. Particularly, given a graph of n nodes, for any query node v and constant error $\epsilon > \frac{3}{n}$, we show that using $O(\log^2 \log n)$ rounds of communication among machines is enough to compute single-source SimRank values with at most ϵ absolute errors, while each machine only needs a space sub-linear to n. To the best of our knowledge, this is the first single-source SimRank algorithm in MPC that can overcome the $\Theta(\log n)$ round complexity barrier with provable result accuracy.

1 Introduction

Evaluating the structural similarity between two nodes in a graph is fundamental in plenty of data mining and machine learning tasks. Examples include recommendation systems [Chen *et al.*, 2022; Mo and Luo, 2021], avoiding customer churn [Luo *et al.*, 2019], spam detection [Badola and Gupta, 2021], link prediction [Zhou *et al.*, 2019; Xu *et al.*, 2017; Hours *et al.*, 2016] and graph mining [Zhu *et al.*, 2024; Jin *et al.*, 2011; Liao *et al.*, 2022; Zhu *et al.*, 2022]. Among many similarity measures, SimRank [Jeh and Widom, 2002] is one of the most widely adopted measures over graphs. SimRank is defined based on the intuition that two nodes are similar only when their neighboring nodes are similar. Formally, it uses the following recursive equation to compute the SimRank between two nodes u and v, where $c \in (0, 1)$ and $\mathcal{I}(u)$ denotes the in-neighbor set of node u.

$$s(u,v) = \begin{cases} 1, & u = v \\ \frac{c}{|\mathcal{I}(u)||\mathcal{I}(v)|} \sum_{u' \in \mathcal{I}(u)} \sum_{v' \in \mathcal{I}(v)} s(u', v'), & u \neq v \end{cases} \quad (1)$$

Since it was proposed by Jeh and Widom [Jeh and Widom, 2002], SimRank has gained increasing popularity in various application domains, e.g., social analysis [Zheng *et al.*, 2013], nearest neighbor search [Lee *et al.*, 2012], as well as recommender systems [Antonellis *et al.*, 2008; Chuanyan and Xiaoguang, 2021; Symeonidis *et al.*, 2021].

Distributed SimRank Computation. We focus on *single-source SimRank computation*, whose importance has been uncovered in a plethora of recent studies [Wang *et al.*, 2020b; Li *et al.*, 2015; Wang *et al.*, 2020a; Shi *et al.*, 2020; Wang *et al.*, 2021; Maehara *et al.*, 2014; Tian and Xiao, 2016; Kusumoto *et al.*, 2014]. Given a graph G, a single-source SimRank for source node u evaluates the SimRank values between u and all other nodes in the graph. Single-source SimRank computation is widely used in applications where a ranking of the other objects with respect to an object is required. For example, it can be applied in search engines to locate the most similar web pages to a given one [Fogaras and Rácz, 2005], or in social network services to recommend new friends to a given user [He *et al.*, 2010; Nguyen *et al.*, 2015], or act as a ranking measurement to cluster objects [Cai *et al.*, 2008].

Given a graph of n nodes, computing single-source SimRank is challenging for large graphs because it inherently involves $O(n)$ times of pairwise SimRank evaluations, each of which can already be too costly. Particularly, following the recursive form in Equation 1, computing $s(u, v)$ requires accessing many pairs of nodes in the graph, leading to $O(n^2)$ complexity. To address the efficiency issue, recent works [Jiang *et al.*, 2017; Zhang *et al.*, 2017; Shao *et al.*, 2015] employ a random-walk-based approach to

[*]The authors are ordered alphabetically.

approximate the SimRank values. The main idea is to translate the SimRank computation into estimating the meeting probability of two decay-based random walks from the two source nodes (See Section 2.2 for more details). The computational complexity is then dependent on the number of random walks sampled, achieving a significant speed-up. However, even with the improved approach, it is still challenging to compute single-source SimRanks when n is large.

Therefore, it is increasingly popular to apply *distributed computation* [Wang *et al.*, 2020b; Li *et al.*, 2015; Song *et al.*, 2018] to SimRank computation, which involves multiple machines to compute SimRank values in a collaborative manner, ultimately scaling up the computation to large graphs. In modeling distributed computation, the local computation cost within each machine is typically dominated by the synchronization cost among the machines. Hence, by convention, the local computation cost is omitted in MPC analysis and the focus is on reducing communication rounds. Furthermore, there is an intrinsic trade-off between the number of communication rounds and space-per-machine.

Open Problem. Existing distributed SimRank algorithms (e.g., [Li *et al.*, 2015; Song *et al.*, 2018; Wang *et al.*, 2020b]) mostly focus on empirical evaluation, and non-trivial theoretical analysis with provable approximation result guarantees is rarely given. We analyze existing representative distributed SimRank algorithms and summarize their requirements regarding communication rounds and machine space in Table 1*. In a nutshell, they require either $\Omega(n)$ per-machine space (UniWalk [Song *et al.*, 2018]) or $\Omega(\log n)$ communication rounds (DISK [Wang *et al.*, 2020b] and CloudWalker [Li *et al.*, 2015]). We remark that overcoming the $\Theta(\log n)$ round complexity barrier in MPC for natural problems is usually challenging and it has attracted tremendous interest [Luo *et al.*, 2022; Lacki *et al.*, 2020; Luo, 2019; Behnezhad *et al.*, 2019; Czumaj *et al.*, 2021; Luo, 2020] to improve the distributed computation down to $O(\mathbf{poly}(\log\log n))$ rounds. Hence, a natural *open theoretical problem* is raised:

Given an error threshold $\epsilon \in (0,1)$, let an approximate single-source SimRank algorithm be an algorithm that outputs SimRank values with ϵ absolute errors. Is there a distributed approximate single-source SimRank algorithm over a graph of n nodes that can be finished in $o(\log n)$ rounds, while each machine only needs $o(n)$ space?

Our Main Results. In this paper, we give a positive answer to the aforementioned open problem and present a distributed single-source SimRank algorithm that suits the Massively Parallel Computing (MPC) model [Karloff *et al.*, 2010]. Particularly, for a graph of n nodes and m edges, we focus on the MPC model that involves a set of machines, each having a sub-linear space $S = n^\alpha$ for some $\alpha \in (0,1)$. Our main results can be stated by the following theorem. To the best of our knowledge, this is the first distributed single-source SimRank algorithm that achieves sub-$\log n$ communication rounds while only requiring strongly sub-linear per-machine space. We leave the proofs of lemmas and theorems in the Appendix of our technique report [Luo and Zhu, 2024].

Theorem 1. *Given a source node u in a graph of n nodes and m edges, there is an algorithm that computes ϵ-absolute-error guaranteed SimRank values between u and all the other nodes using M machines in $O\left(\log^2 \log n\right)$ communication rounds with high probability †. This algorithm only requires the space per machine is $S = n^\alpha$ for some $\alpha < 1$, and M is some value of $\tilde{O}((m+n)/S)$.*

2 Preliminaries

2.1 Distributed Computation Model

Distributed computation is one of the most important techniques to address various computation tasks [Luo *et al.*, 2023; Wang *et al.*, 2020b; Li *et al.*, 2015; Song *et al.*, 2018; Luo *et al.*, 2022; Luo *et al.*, 2014; Luo *et al.*, 2012; Klauck *et al.*, 2014]. In recent years, Massively Parallel Computation (MPC) [Karloff *et al.*, 2010] becomes a popular theoretical framework in modeling the complexity of a distributed algorithm, because it closely simulates the situation of general distributed computation. An MPC model has three important parameters: the input data size F, the number of involved machines M, and the space capacity (words) S on each machine. The frequently used notations are listed in Table 2 in the Appendix. It is required that when given F and S, the number of machines M should be of $\tilde{O}(\frac{F}{S})$, where $\tilde{O}(\cdot)$ hides a polylogarithmic factor. Our main focus on the MPC model derives from several perspectives as follows:

Space. Consider an input graph of n nodes. MPC for graph algorithms can be categorized into three types: strongly **super-linear** space ($S = n^{1+\omega}$) for some constant $\omega > 0$, near **linear** space ($S = \Theta(n)$), and strongly **sub-linear** space ($S = n^\alpha$) for some constant $\alpha \in (0,1)$. A super-linear model can typically be employed with a local algorithm, which loses the generality to be employed on a large scale of data. Hence, many studies focus on sub-linear settings that better capture the scalability of a distributed system. In this paper, we focus on the strongly sub-linear setting where each machine cannot even store the whole set of graph nodes.

Communication Rounds. The computation in the MPC model is based on *communication rounds*. Initially, each edge is randomly assigned to a machine. We assume each node has an integer ID from 1 to n, and each machine has an integer ID from 1 to M. The computation proceeds in synchronized rounds. At the beginning of a round, each machine may receive the messages sent from some other machines in the previous rounds. During a round, every machine conducts local computation based on its local data or messages received. Then each machine will send the computed results, packed as messages, to target machines. Each machine creates message packages to be routed onto the network and hence the size of messages sent/received per machine in one round shall not exceed the space capacity S. A new round starts only after the end of the previous round. The term *round complexity* [Ghaffari *et al.*, 2019] refers to the number of rounds required

*For the limitation of space, we leave the related work analysis in Appendix C. As the authors did not give an analysis based on the MPC model, we analyze them on our own.

†We say an event happens with high probability, if there exists a constant $\tau > 0$ such that the event happens with probability at least $1 - \frac{1}{n^\tau}$, where n is the number of graph nodes.

Algorithm	Rounds	Total space cost per round	Accuracy error	Space cost per-machine
CloudWalker[1] [Li *et al.*, 2015]	$O(\frac{nl(I+l)}{b})$	$O\left(\frac{bl^2 \log n}{\epsilon_p^2} + n + m\right)$	no guarantees	$O\left(\frac{bl^2 \log n}{\epsilon_p^2} + \frac{m+n}{M}\right)$
UniWalk[2] [Song *et al.*, 2018]	$O(l)$	$O\left(\frac{n^2 l \log n}{\epsilon^2}\right)$	$\epsilon + \epsilon_b$	$O\left(\frac{n^2 l \log n}{\epsilon^2}\right)$
DISK[3] [Wang *et al.*, 2020b]	$O\left(\frac{\log^2 n}{\epsilon_d^2} + K\right)$	$O\left(m \log n + Kn\right)$	$\frac{c\left(1-c^K\right)\epsilon_d}{1-c} + c^{K+1}$	$O\left(\frac{m \log n + Kn}{M}\right)$
Ours[4]	$O\left(\log^2 \log n\right)$	$O\left(m \log n + n^{1+o(1)} \log^{4.5} n + \frac{n \log^{3+o(1)} n}{2\left(\epsilon - \frac{3}{n}\right)^2}\right)$	ϵ	$O(n^\alpha)$

[1] ϵ_p is the error when estimating the random walk distribution, b is the parameter that controls the number of nodes to be handled in a single machine, and I is the number of iterations in the Jacobi method. l is a user-defined walk length or number of steps. [2] l is a user-defined walk length or number of steps, and ϵ_b is the absolute error of SimRank introduced by BiWalk [Fogaras and Rácz, 2005]. [3] K is the number of truncated terms of linearized SimRank and ϵ_d is an internal estimation error threshold. [4] $\alpha < 1$ and $n^\alpha > \log^5 n / (\log \frac{1}{\sqrt{c}})$.

Table 1: Communication rounds of distributed single-source SimRank Computation.

for program execution and serves as a measure of distributed computation cost, primarily driven by the expensive communication among machines. The main principle of designing algorithms in the MPC model is to achieve low round complexity subject to the space constraint on each machine.

Existence of an MPC Algorithm. With a per-machine space of $S = n^\alpha$, we acknowledge the existence of an MPC algorithm such that when $n \geq n_0$ (with n_0 being a constant), a distributed algorithm can operate within this space constraint using M machines, where $M = \tilde{O}(\frac{m+n}{S})$. For ease of analyzing the MPC algorithms, we first give a preparation lemma regarding machine space expansion, as follows.

Lemma 1. *If there exists an $O(f(n))$-round distributed graph algorithm that works for per-machine space $S = \Theta(n^\alpha)$ for any $0 < \alpha < 1$ using $M = \tilde{O}(\frac{m+n}{n^\alpha})$ machines, and $f(n)$ is a function not related to α (i.e., α only contributes a constant factor to the round complexity), then there exists an MPC algorithm that works for per-machine space $S = n^\alpha$ with the same round complexity.*

This lemma eliminates the obstacle of analyzing round complexity when there is a constant factor expansion of the machine space. For example, consider that we have an algorithm with a certain distributed algorithm such that (1) its round complexity hides α factors; (2) applies to any $\alpha \in (0, 1)$ with $M = \tilde{O}(\frac{m+n}{n^\alpha})$, and (3) the space per machine is Cn^α for some constant C. Then Lemma 1 guarantees the existence of an MPC algorithm with n^α machine space.

2.2 Approximate SimRank Computation

Given a directed graph $G = (V, E)$, and let $n = |V|$ and $m = |E|$. Following [Wang *et al.*, 2020a; Shi *et al.*, 2020], we aim to compute approximate SimRank values with constant errors. In particular, given any source node $u \in G$ and a constant error ϵ, we aim to compute the SimRank values between u and any other node $v \in G$, such that $|s(u, v) - \tilde{s}(u, v)| \leq \epsilon$, where $s(u, v)$ denotes the true SimRank value and $\tilde{s}(u, v)$ denotes the estimated value.

Calculating the SimRank value iteratively according to Equation 1 may occupy large memory space and incur a high computation cost. Therefore, state-of-the-art approaches employ the following $\sqrt{c}$-decay walk-based computation, first proposed in [Tian and Xiao, 2016]:

Lemma 2. *For any two nodes $u, v \in G$, the SimRank between u and v is equal to the meeting probability of two $\sqrt{c}$-decay walks starting at u and v on the reverse graph $\bar{G}$, where two walks meet if there exists an integer $i \geq 0$, such that the i-th step of the two walks visit the same node.*

Here, a $\sqrt{c}$-decay walk on $\bar{G}$ from a node u is a traversal on $\bar{G}$ such that at each step of the walk, it has $1 - \sqrt{c}$ probability to stop at the current node, and otherwise jumps to the next node that is a uniformly chosen out-neighbor of the current node. Here we define a length-l walk as a path that includes $l + 1$ nodes and l edges. We also say a length-l walk has l steps. Particularly, we have $(\sqrt{c})^i \cdot (1 - \sqrt{c})$ probability to generate a length-i walk from a given starting node.

Monte Carlo Method. By sampling N pairs of $\sqrt{c}$-walks from u and v, and if H pairs of the walks meet, then $\frac{H}{N}$ is an estimator of $s(u, v)$. The sampling number N controls the estimation accuracy since more random walks approximate the value of $s(u, v)$ more accurately.

3 $O(\log^2 \log n)$-Round Algorithm

It is not too challenging to design $O(\log n)$-round MPC algorithm (See Appendix E for details), but further improving the round complexity can be difficult. To explain, we outline challenges of direct adaptation of $\sqrt{c}$-walk based approach. First, the maximum length of a $\sqrt{c}$-decay walk can be infinite; a straightforward method to compute length-l in a distributed environment easily entails l communication rounds because one step may require one communication round when some neighbors are located in a different machine. Such design easily leads to $\Omega(\log n)$-round algorithms. Second, evaluating single-source SimRank values for node u requires running random walks from both u and all the other nodes, resulting in a large number of random walks being conducted. This can lead to a high space cost per machine because some hub nodes are prone to be passed by many random walks. Therefore, a careful design of the MPC algorithm is required to guarantee a small number of rounds and a low space cost for each machine. Third, the random walks, once computed in the MPC model, are stored in different machines. As such, detecting whether two walks meet may overload the machine regarding space cost.

Algorithm 1: Overall algorithm

Input : Graph $G = (V, E)$; source node u; decay factor c; Montel-Carlo failure probability δ; accuracy error ϵ
Output: SimRank scores $\tilde{s}(u, v)$ for each $v \in V$

1 *Store G across multiple machines*;
2 *Reverse the edges of G within each machine to form* $\bar{G}$;
3 $l \leftarrow \log_{\frac{1}{\sqrt{c}}} n$;
4 **for** $i = 0$ **to** l *in parallel* **do**
5 $\quad N_i = \left\lceil \frac{\log 2n}{2\left(\epsilon - \frac{3}{n}\right)^2} \cdot (\sqrt{c})^i \cdot (1 - \sqrt{c}) \right\rceil$;
6 $\quad$ **foreach** *node* $v \in V$ *in parallel* **do**
7 $\quad\quad$ Generate N_i random walks from node v in $O\left(\log^2 \log n\right)$ rounds;
8 *Shuffle and decompose random walks (See Sections 4.2 and 4.3);*
9 *Run* ***Algorithm 2*** *to calculate* $\tilde{s}(u, v)$ *for each* $v \in V$*;*

3.1 New Interpretation of Walk-based Approach

To address the aforementioned challenges, we reinterpret the SimRank computation between node u and node v in a batch manner using the following three operations, which are more MPC-friendly:

(a) Random walk generation. Instead of generating $\sqrt{c}$-decay random walks whose lengths are non-deterministic, we generate random walks with deterministic length distribution. Particularly, to generate N $\sqrt{c}$-walks from u, we will generate $N \cdot \sqrt{c}^i(1 - \sqrt{c})$ length-i walks for $i \geq 0$, following the corresponding geometric distribution of walk lengths. Similar operations are conducted for random walks starting from v. Fixing the random walk lengths is more MPC-friendly. As we will show in Section 4.1, we can generate sufficient such random walks both from the source node u and each node $v \in V$ in only $O\left(\log^2 \log n\right)$ MPC rounds.

(b) Random walk shuffling and matching. Since the random walks from u (or v) are generated in the order of increasing lengths (due to Step (a)), pairing up the random walks from u and v directly for meeting detection is not valid for estimating SimRanks as the i-th walks from u and v are correlated (they have the same length). In order to maintain the randomness, we shuffle the random walks above.

(c) SimRank computation. Last, we compute the SimRanks by detecting how many pairs of walks meet and estimating the SimRank value $s(u, v)$ by Lemma 2.

3.2 Overview of Main Algorithmic Steps

Based on the new interpretation, we give our main algorithmic steps in Algorithm 1, which consists of the following five main stages. For ease of presentation, we first outline the main idea in this section and defer the detailed MPC-related operations in Section 4.

Initial State. The SimRank algorithms are based on the topology information of the input graph. Initially, the whole graph should be partitioned across different machines, as assumed by the MPC model. We assume that each machine holds a random partition of edges of the graph (Line 1 in Algorithm 1), and hence each machine holds roughly $\frac{m}{M}$ edges.

Algorithm 2: Detect meeting and calculate SimRank

Input : Tuple set D_G from Section 4.3; source node u
Output: Estimated SimRank score $\tilde{s}(u, v)$ for each $v \in V$

1 *Sort all elements in* D_G;
2 *Calculate* $z = \frac{\log 2n}{2(\epsilon - \frac{3}{n})^2}$, $|Z_{u,l}| = z \cdot (\sqrt{c})^l \cdot (1 - \sqrt{c})$ *and* $|\hat{Z}_{u,l}| = \lceil |Z_{u,l}| \rceil$ *for* $l = 1, ..., \log_{\frac{1}{\sqrt{c}}} n$ and $u \in V$;
3 **for** *all tuple* $(i, j, v_j, l, v) \in D_G$ *in parallel* **do**
4 $\quad$ **if** $v = u$ **then**
5 $\quad\quad$ Label (i, j, v_j, l, v) as 1
6 $\quad$ **else**
7 $\quad\quad$ Label (i, j, v_j, l, v) as 0
8 **for** *all tuple* (i, j, v_j, l_2, v) *labeled by 0 in parallel* **do**
9 $\quad$ Link it to the closest tuple (i, k, v_k, l_1, u) labeled by 1 using PREDECESSOR;
10 $\quad$ **if** $(j = k) \wedge (v_j = v_k)$ **then**
11 $\quad\quad$ Send message (j, u, l_1, v, l_2) to $\left\lfloor \frac{vM}{n} \right\rfloor$-th machine.
12 **for** *each unique message* (j, u, l_1, v, l_2) *in each machine in parallel* **do**
13 $\quad \tilde{s}(u, v) \leftarrow \tilde{s}(u, v) + \frac{|Z_{u,l_1}| \cdot |Z_{v,l_2}|}{|\hat{Z}_{u,l_1}| \cdot |\hat{Z}_{v,l_2}|} \cdot \frac{1}{z}$

We also reverse the edges so that the graph represents $\bar{G}$ (Line 2 of Algorithm 1).

Parallel Random Walks Generation. This stage corresponds to Operation (a) mentioned earlier. As we need to compute single-source SimRank values from any source node u to all other nodes, we generate N $\sqrt{c}$-walks from each node. Further, we truncate them at $\log_{1/\sqrt{c}} n$ length (Line 3 of Algorithm 1). By setting $N = \frac{\log 2n}{2(\epsilon - 3/n)^2}$, we show that the truncated walks only cause negligible influence on the final accuracy (see Section 4), and we can guarantee the ϵ error bound of SimRank values. Another problem is when generating $N_i = N \cdot (\sqrt{c})^i (1 - \sqrt{c})$ length-i walks, N_i may not be an integer. In Section 5 we give a rounding technique and prove that the rounding still guarantees the error bound. We also prove that these random walks can be obtained in the MPC model using $O(\log^2 \log n)$ communication rounds (Lines 4-7 of Algorithm 1 and details in Section 4.1).

Shuffling Random Walks. It's important to note that in Lemma 2, the lengths of the random walks originating from u and v may differ, as they are independently and randomly generated. To guarantee the randomness of each pair of random walks in our Monte Carlo simulation, the generated random walks have to be shuffled (Line 8 of Algorithm 1). The shuffling operation in the MPC model is not as trivial as in a single machine because the generated walks from the same node can be located in different machines. Shuffling these walks incurs communication between machines, which may violate the space bound in each machine.

Decomposing Random Walks. After shuffling, we pair the j-th walk from the source node u with the j-th walks from other nodes respectively, and detect whether two walks in each pair meet. Detecting whether two walks meet is relatively simple in the single-machine setting because all information can be loaded locally. However, in a distributed

environment, we need to detect whether two paired-up walks meet while the walks are stored at different machines. This poses drastically different challenges. To address the challenges, we decompose the generated random walks into tuples, each containing one visited node in the walk (except the starting node). Each tuple contains five elements to convey all the information needed to detect whether the walk intersects another walk in the specified node. As shown in Figure 1, there is a walk $\{u, v_1, v_2, v_3\}$ decomposed into three tuples $(j, 1, v_1, 3, u)$, $(j, 2, v_2, 3, u)$, $(j, 3, v_3, 3, u)$, where j implies that the tuple is decomposed from the j-th walk of those starting from u. The remaining elements in the tuple are the *walk step*, *visited node at the particular walk step*, *the total steps in the walk*, and *source node*, respectively.

Computing SimRank Values. Evaluating the SimRank value between u and v is based on the probability that two walks from u and v meet. This can be detected by sorting the decomposed tuples (Line 9 of Algorithm 1→ Line 1 of Algorithm 2). Figure 1 shows the j-th walks from u and v respectively. The two walks meet because they share the same node v_2 in their third walk steps. Corresponding to v_2, there are two decomposed tuples from the two walks, denoted by $(j, 2, v_2, 3, u)$ and $(j, 2, v_2, 4, v)$. We note that they *share the first three elements*. In general, it is easy to verify that if two walks meet, there must be two decomposed tuples, each from one walk, that share the first three elements in the tuple. More formally, suppose there are two j-th walks $\{v_0, v_1, \ldots, v_{l_1}\}$ and $\{u_0, u_1, \ldots, u_{l_2}\}(0 \leq l_1, l_2 \leq \log_{1/\sqrt{c}} n)$ starting from v_0 and u_0. Walk $\{v_0, v_1, \ldots, v_{l_1}\}$ is decomposed into l_1 tuples, denoted by (j, i, v_i, l_1, v_0) $(1 \leq i \leq l_1)$; Walk $\{u_0, u_1, \ldots, u_{l_2}\}$ is decomposed into l_2 tuples, denoted by (j, k, u_k, l_2, u_0) $(1 \leq k \leq l_2)$. Then, if the two walks meet, or equivalently, they share the same node v_i in the same step i, there must be $v_i = u_i$ and the corresponding two tuples (for v_i and u_i) share the first three elements. Again, the challenges are in a proper rearrangement of the tuples across the machines, and the collection and aggregation of the walk-meeting cases within each machine.

With the above five stages, we highlight the major differences between our approach and the classic $\sqrt{c}$-walk-based algorithms: **(a)** the lengths of the walks are regularized to geometric distributions, and longer walks are truncated and we show that this would not violate the error tolerance in Section 5; **(b)** most steps are redesigned non-trivially so that all the steps can be efficiently implemented in the MPC model.

4 Detailed MPC Operations

4.1 Parallel Random Walks Generation

Performing random walks in the MPC model will incur communication rounds because the next sampled neighbor for each node can be located in different machines. Let us consider the situation of a node v_0 sampling its neighboring node v_1 in the walk. In the MPC model, as v_0 and v_1 can be in different machines, forming a walk of $v_0 \to v_1$ may need one communication round. In general, forming an l-step random walk in the MPC model typically incurs l communication rounds. For the purpose of reducing communication rounds, we first introduce a result provided by WRME [Lacki *et al.*, 2020], and then give our extension in Theorem 3:

Theorem 2. *Let G be a directed graph. Let N and l be positive integers such that $l = o(S)/\log^3 n$, where S is the available space per machine. For the task that samples N independent random walks of length l starting from each node v in G, there exists an MPC algorithm that runs in $O\left(\log^2 \log n + \log^2 l\right)$ rounds and uses $O\left(m + n^{1+o(1)} l^{3.5} + Nnl^{2+o(1)}\right)$ total space and strongly sub-linear space per machine $S = n^\alpha$ $(0 < \alpha < 1)$. The algorithm is an imperfect sampler that does not fail with probability $1 - O\left(n^{-1}\right)$.*

Essentially, Theorem 2 states that the generation of a length-l random walk from *every* node can be round-efficient in the MPC model. Particularly, it takes only $O\left(\log^2 \log n + \log^2 l\right)$ communication rounds to generate N length-l random walks, each with different source node. However, even with Theorem 2, we cannot directly give a reasonable round-complexity for SimRank evaluations if we use the original $\sqrt{c}$-decay walk-based method. The reason is that the length l of a $\sqrt{c}$-decay walk can be infinite. To address this issue, as we show in Algorithm 1 Line 3 and Line 5, we carefully design the truncated walk length (Line 3) and the number of random walks to be sampled for each walk length (Line 5). The truncated length guarantees a reasonable bound of $O\left(\log^2 \log n\right)$ communication rounds when $l = \log_{1/\sqrt{c}} n$, and the number of random-walk samples ensures the SimRank estimation accuracy (we will formally prove it in Section 5).

Parallel Random Walks in $O(\log^2 \log n)$ Rounds. To effectively apply Theorem 2, we let Task-i be the generation of N_i length-i walks from every node. We let all Task-i $(1 \leq i \leq \log_{1/\sqrt{c}} n)$ be conducted in MPC in parallel. We apply $\log_{1/\sqrt{c}} n$ WRME algorithm [Lacki *et al.*, 2020] instances *concurrently* in the MPC model, where each instance corresponds to one task.

Unfortunately, concurrently conducting multiple WRME algorithms will incur a space cost higher than n^α in each machine because by default each algorithm instance can incur a local cost up to n^α. To address this issue, we let β satisfy that $n^\beta = n^\alpha / \log_{1/\sqrt{c}} n$, and we apply the WRME algorithm concurrently for each Task-i with $S = n^\beta$. Particularly, each machine space n^α is divided into subspaces of size n^β to compute Task-i. By careful analysis of the round complexity and total space cost, we give the following theorem.

Theorem 3. *The random walk generation in Algorithm 1 can be done in $O\left(\log^2 \log n\right)$ rounds with high probability, with machine space $S = n^\alpha$ such that $n^\alpha > \log^5 n / (\log \frac{1}{\sqrt{c}})$ and a total space $O\left(m \log n + n^{1+o(1)} \log^{4.5} n + \frac{n \log^{3+o(1)} n}{2(\epsilon - \frac{3}{n})^2}\right)$.*

We note that the WRME algorithm is an imperfect sampler that fails with at most a probability $O(n^{-1})$. We only apply $O(\log n)$ times the sampler, which can still easily guarantee that our algorithm is successful with high probability (because $O(\log n) = o(n^\tau)$ for any constant $\tau > 0$).

4.2 Shuffling Random Walks

We need to shuffle the generated N random walks sourced at each node to fully mimic the behavior of generating a $\sqrt{c}$-

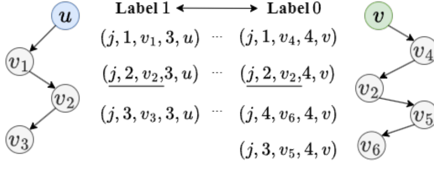

Figure 1: Detect meeting using tuple association.

walk. The reason for shuffling is that when we select two random walks that start from two nodes and calculate their meeting probability, the lengths of these two walks are not bound to be the same. This shuffling operation helps us guarantee the randomness in our Monte Carlo simulation of the generated $\sqrt{c}$-decay walks.

Shuffling in $O(1)$ Rounds. We note that sorting in MPC can be done in $O(1)$ rounds [Goodrich *et al.*, 2011] as long as the number of items is $O(n^\eta)$ for some constant $\eta > 0$. Observe that $N = O(\frac{\log n}{\epsilon^2})$, and hence there exists η such that $nN = O(n^\eta)$. Hence, sorting the nN walks all together can be finished in constant rounds in MPC, where the comparison of two walks is based on the comparison of the corresponding nodes in the walks. As such, all the walks sourced at the same node will be clustered.

For sufficiently large n we have $n^\alpha > 2N \log_{1/\sqrt{c}} n$, and the walks sourced at the same node, occupying a space of $N \log_{1/\sqrt{c}} n$, can be held in one machine. Since all the walks are sorted and $\frac{n^\alpha}{N \log_{1/\sqrt{c}} n}$ may not be an integer, each set of N walks sourced at the same node may go across two machines. Let $\boldsymbol{W}_u$ be the set of walks sourced at node u. We discuss two cases to shuffle the walks of the same source node.

Case 1: If the walks in $\boldsymbol{W}_u$ are fully located in a machine, shuffling is done locally. We shuffle all sets of walks, and for each set we assign numbers from 1 to N to each walk in $\boldsymbol{W}_u$ after shuffling.

Case 2: If the walks sourced at the same node are located at two consecutive machines. Without loss of generality, we denote the source node as u, and there are N_0 walks in the first machine and $N - N_0$ in the second. We then swap those $N - N_0$ walks in the second machine with the last $N - N_0$ walks sourced at node $u - 1$, as illustrated in Figure 2 in the Appendix. Since $n^\alpha > 2N \log_{1/\sqrt{c}} n$ for sufficiently large n, it is guaranteed that those $N - N_0$ walks sourced at $u - 1$ are located in the first machine and have been shuffled locally in Case 1. Once swapped, the shuffling within $\boldsymbol{W}_u$ can be done locally. We note that by guaranteeing $n^\alpha > 2N \log_{1/\sqrt{c}} n$ we can make sure the swaps are not conflicting and thus can be done in one round.

Any set $\boldsymbol{W}_u$ can be shuffled in Case 1 or Case 2. During shuffling, each walk is labeled by 1 to N sequentially, and the j-th walk refers to the walk assigned with a number j.

4.3 Decomposing Random Walks

Decomposition in $O(1)$ Rounds. Recall that each length-t walk $\{v_0, v_1, \ldots, v_t\}$ is to be decomposed into t tuples each of which has 5 elements: (j, i, v_i, t, v_0), for $1 \leq i \leq t$. Tuple (j, i, v_i, t, v_0) implies that in the j-th random walk of the walks starting from v_0, the i-th step of the walk visits node v_i and the walk has overall t steps. In the MPC model, the decomposition of a random walk can be done within the machine holding the walk. While each walk will generate multiple tuples, the total space is only amplified by a constant factor (i.e., 5 times) because each tuple corresponds to one visited node along the walk. Hence, this operation will not incur any communication among the machines, and only has a constant expansion of the space cost, which has a negligible effect due to Lemma 1. We denote the set of all the decomposed tuples by D_G.

4.4 Detecting Meeting-Walks

As the decomposed tuples are stored at different machines, challenges exist if we use these tuples to detect whether some pairs of random walks meet. Algorithm 2 shows the pseudocode of detecting walk-meetings and computing SimRank.

Sorting in $O(1)$ Rounds. Our first step is to sort all the tuples across the machines (Line 1). The sorting is based on the *multi-dimensional* sorting because the tuple contains five elements, i.e., sorting is first done based on the first element, and then for the elements that share the same first element, sorting is based on the second element, and so on. Elements are sorted based on their node IDs. We also intentionally let the node ID of source node u be **smaller** than other nodes to guarantee that walks sharing the same first three elements start with the walk sourced at u.

Recall in Section 3 we show that if two walks meet, then there must be two decomposed tuples, each from one walk, sharing the first three elements. After sorting the tuples, the tuples sharing the first three elements will be clustered together and the tuple whose fifth element is u, if exists, will be placed in the first position in the cluster. Particularly, suppose we have tuples (j, i, v, l_1, u), (j, i, v, l_2, v_1), (j, i, v, l_3, v_2) that share the first three elements, implying that the j-th walk started from u meets the j-th walks started from v_1 and v_2. These three tuples will be clustered in the order of (j, i, v, l_1, u), (j, i, v, l_2, v_1), (j, i, v, l_3, v_2) after sorting. The formal result is as follows.

Lemma 3. *If the j-th walk W started from the query source node u meets the j-th walk W' started from v, then there must be a decomposed tuple T of W and a decomposed tuple T' of W' that share the first three elements. Furthermore, after sorting the tuples, T is the closest tuple that is decomposed from W and comes before T'.*

Walk Paired-Up in $O(1)$ Rounds. To compute the SimRank between u and v_1, we need to pair up (j, i, v, l_1, u) and (j, i, v, l_2, v_1); similarly, we also need to pair up (j, i, v, l_1, u) and (j, i, v, l_3, v_2) to compute the SimRank between u and v_2. Pairing-up can be challenging because the tuples are stored at different machines, and hence communication between machines is unavoidable. In the MPC model, a machine that communicates with other machines would require the same space as the size of communication messages. To reasonably bound the communication cost between the machines, we employ the following *PREDECESSOR* procedure [Behnezhad *et al.*, 2019] to couple the tuples sharing the same first three elements, which can be finished in $O(1)$ communication rounds.

PREDECESSOR*: Considered an ordered list of tuples such that each tuple is labeled by 0 or by 1. Then, for each tuple T' labeled by 0, PREDECESSOR associates the closest*

tuple T labeled by 1 such that T comes before T' in the ordering. PREDECESSOR can be implemented in $O(1)$ MPC rounds with n^α space per machine, for any constant $\alpha > 0$.

To apply *PREDECESSOR* for pairing up the tuples, each machine can run a local algorithm to distinguish the tuples that are generated from the random walks starting at node u and others. As shown in Figure 1, all machines assign label 1 to the tuples whose fifth elements are u in parallel. Otherwise, the tuples are labeled by 0. By Lemma 3, we can apply the *PREDECESSOR* procedure to associate the closest Label-1 tuple (i.e., the tuple generated from the walk sourced at u) to each of its following Label-0 tuples. Here, the *association* means that the machine holding the Label-0 tuple is aware of its closest Label-1 tuple before it. For each such associated pair T and T', if their first three elements are the same, then the corresponding walks of T' and T meet.

4.5 Computing SimRanks

The paired-up tuples in the previous step will contribute to the corresponding SimRank value. For example, suppose two walks from u and v meet, then this pair of walks contributes to the SimRank value $s(u, v)$, because $s(u, v)$ is estimated by the meeting probability of the walks from node u and node v. The challenge is how to concurrently distribute and aggregate these *SimRank contributions* across machines. Note that it is crucial to let those *SimRank contributions* corresponding to the same pair of nodes (e.g., (u, v)) be processed in the same machine, to avoid repeat counting of walk-meets. For example, walks $\{u, u_1, u_2, w\}$ and $\{v, u_1, u_2, k\}$ meet at both node u_1 and node u_2. To ensure accuracy, the event of each walk-meet should be counted only once. For this purpose, we let each machine specifically handle SimRank evaluations of $\frac{n}{M}$ nodes with respect to u. Without loss of generality, we assume the node IDs are from 1 to n. Then, the i-th machine handles the node set with node IDs in $\left[\frac{n(i-1)}{M} + 1, \frac{ni}{M}\right]$.

Computing SimRanks in $O(1)$ Rounds. The detailed MPC operations of computing SimRanks are performed as follows (also illustrated in Algorithm 2 Lines 8 - 11). Using the PREDECESSOR procedure, for each tuple (i, j, v_j, l_2, v) with label 0, it can be linked to the closest tuple (i, k, v_k, l_1, u); if $j = k$ and $v_j = v_k$, then that indicates the meet of two walks represented by the two tuples. Then, a *walk-meet message* (j, u, l_1, v, l_2) will be sent (by the machine holding tuple (j, i, v_i, l_1, u)) to the target machine that is responsible to computing SimRank $s(u, v)$. Here, the walk-meet message (j, u, l_1, v, l_2) indicates that a length-l_1 walk sourced at node u and a length-l_2 walk sourced at node v meet. Also, assuming u and v are node IDs from 1 to n, the target machine that the message is sent to is the $\lfloor \frac{uM}{n} \rfloor$-th machine because this machine will compute $s(u, v)$. Each machine may then receive multiple copies of the walk-meet message (j, u, l_1, v, l_2), and only one of them will be kept because each walk-meet event should be counted only once. For each unique message (j, u, l_1, v, l_2) received, we add a value $\frac{|Z_{u,l_1}| \cdot |Z_{v,l_2}|}{|\hat{Z}_{u,l_1}| \cdot |\hat{Z}_{v,l_2}|} \cdot \frac{1}{z}$ to the SimRank value $\tilde{s}(u, v)$, where $z = \frac{\log 2n}{2(\epsilon - \frac{3}{n})^2}$ is the expected number of walk samples, $\hat{Z}_{u,l}$ denotes the set of actually generated length-l random walks sourced at u, and $Z_{u,l}$ is the expected set of length-l random walks to be generated sourced at u.

We distinguish actual and expected samples because $Z_{u,l}$ may not be an integer. Particularly, we note that $|\hat{Z}_{u,l}| = \lceil z \cdot (\sqrt{c})^l \cdot (1 - \sqrt{c}) \rceil$, and $|Z_{u,l}| = z \cdot (\sqrt{c})^l \cdot (1 - \sqrt{c})$. When $z \cdot (\sqrt{c})^l \cdot (1 - \sqrt{c})$ is not an integer, the expected walk set $Z_{u,l}$ is defined based on size-$\lceil |Z_{u,l}| \rceil$ set Z_1 and size-$\lfloor |Z_{u,l}| \rfloor$ set Z_2, such that there is probability $|Z_{u,l}| - \lfloor |Z_{u,l}| \rfloor$ that one walk of $Z_{u,l}$ is selected from set Z_1 and otherwise selected from set Z_2.

Space Analysis for Walk-Meet Messages. The walk-meet messages received by each machine are at most $\log_{1/\sqrt{c}} n$ times the number of the random walks started at the nodes handled by the machine. To see this, in the worst case, for each node of a walk W generated from v handled by the machine, there is a walk W' from u that meets W at the node, creating a walk-meet message. Since each machine is responsible to compute SimRank values for n/M nodes, the space per machine for receiving the message is $O\left(\frac{n}{M} \log_{\frac{1}{\sqrt{c}}} n \cdot \frac{\log 2n}{2(\epsilon - \frac{3}{n})^2}\right) = O\left(\frac{n \log^2 n}{M(\epsilon - \frac{3}{n})^2}\right)$. The total space cost for walk-meet messages is therefore $O\left(\frac{n \log^2 n}{M(\epsilon - \frac{3}{n})^2} \cdot M\right) = O\left(\frac{n \log^2 n}{(\epsilon - \frac{3}{n})^2}\right)$.

5 Summary of the Results

Round Complexity. The round complexity is dominated by the generation of parallel random walks, which is $O(\log^2 \log n)$ by Theorem 3.

Space Complexity. The total space cost for walk-meet messages is dominated by the random-walk generation space cost $O\left(m \log n + n^{1+o(1)} \log^{4.5} n + \frac{n \log^{3+o(1)} n}{2\left(\epsilon - \frac{3}{n}\right)^2}\right)$ (see Theorem 3). We note that the total space cost is $\tilde{O}(m + n)$, indicating the existence of $M = \tilde{\Theta}(\frac{m+n}{S})$ for $S = n^\alpha$.

All these results give us Theorem 4 as well as Theorem 1.

Theorem 4. *Given a graph of n nodes and m edges, and a constant error $\epsilon > 0$, for sufficiently large n, Algorithm 1 can be run in MPC using $O\left(\log^2 \log n\right)$ communication rounds, with a total space per-round as $O\left(m \log n + n^{1+o(1)} \log^{4.5} n + \frac{n \log^{3+o(1)} n}{2(\epsilon - \frac{3}{n})^2}\right)$. The space per-machine needed is $S = n^\alpha$ such that $n^\alpha > \log^5 n / (\log \frac{1}{\sqrt{c}})$. Algorithm 1 is an imperfect sampler that fails with probability at most $O\left(n^{-1}\right)$.*

Accuracy Analysis. The accuracy guarantee of our random-walk sampling techniques resembles that of a simple Monte Carlo method, but we also incorporate the techniques of length-truncation and the rounding of the number of walks. We only include the main result as follows. The proof and the methodology for further reducing the threshold $\frac{3}{n}$ are detailed in the Appendix.

Theorem 5. *Algorithm 1 outputs SimRank values $\tilde{s}(u, v)$ with error at most ϵ ($\epsilon \geq \frac{3}{n}$) and with probability at least $1 - \frac{1}{n}$.*

Acknowledgments

This research is supported by the Ministry of Education, Singapore, under its AcRF Tier-2 Grant (T2EP20122-0003).

References

[Antonellis *et al.*, 2008] Ioannis Antonellis, Hector Garcia-Molina, and Chi-Chao Chang. Simrank++: query rewriting through link analysis of the clickgraph (poster). In *WWW*, pages 1177–1178. ACM, 2008.

[Badola and Gupta, 2021] Kshitiz Badola and Mridul Gupta. Twitter spam detection using natural language processing by encoder decoder model. In *ICAIS*, pages 402–405. IEEE, 2021.

[Behnezhad *et al.*, 2019] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, and Vahab S. Mirrokni. Near-optimal massively parallel graph connectivity. In *FOCS*, pages 1615–1636. IEEE Computer Society, 2019.

[Cai *et al.*, 2008] Yuanzhe Cai, Pei Li, Hongyan Liu, Jun He, and Xiaoyong Du. S-simrank: Combining content and link information to cluster papers effectively and efficiently. In *ADMA*, volume 5139 of *Lecture Notes in Computer Science*, pages 317–329. Springer, 2008.

[Chen *et al.*, 2022] Rihan Chen, Bin Liu, Han Zhu, Yaoxuan Wang, Qi Li, Buting Ma, Qingbo Hua, Jun Jiang, Yunlong Xu, Hongbo Deng, et al. Approximate nearest neighbor search under neural similarity metric for large-scale recommendation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 3013–3022, 2022.

[Chuanyan and Xiaoguang, 2021] Zhang Chuanyan and Hong Xiaoguang. Neural graph filtering for context-aware recommendation. In *Asian Conference on Machine Learning*, pages 969–984. PMLR, 2021.

[Czumaj *et al.*, 2021] Artur Czumaj, Peter Davies, and Merav Parter. Graph sparsification for derandomizing massively parallel computation with low space. *ACM Trans. Algorithms*, 17(2):16:1–16:27, 2021.

[Fogaras and Rácz, 2005] Dániel Fogaras and Balázs Rácz. Scaling link-based similarity search. In *Proceedings of the 14th international conference on World Wide Web*, pages 641–650, 2005.

[Ghaffari *et al.*, 2019] Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional hardness results for massively parallel computation from distributed lower bounds. In *FOCS*, pages 1650–1663. IEEE Computer Society, 2019.

[Goodrich *et al.*, 2011] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *ISAAC*, volume 7074 of *Lecture Notes in Computer Science*, pages 374–383. Springer, 2011.

[He *et al.*, 2010] Guoming He, Haijun Feng, Cuiping Li, and Hong Chen. Parallel simrank computation on large graphs with iterative aggregation. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 543–552. ACM, 2010.

[Hoeffding, 1994] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The collected works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[Hours *et al.*, 2016] Hadrien Hours, Eric Fleury, and Márton Karsai. Link prediction in the twitter mention network: impacts of local structure and similarity of interest. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 454–461. IEEE, 2016.

[Jeh and Widom, 2002] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543, 2002.

[Jiang *et al.*, 2017] Minhao Jiang, Ada Wai-Chee Fu, and Raymond Chi-Wing Wong. Reads: a random walk approach for efficient and accurate dynamic simrank. *Proceedings of the VLDB Endowment*, 10(9):937–948, 2017.

[Jin *et al.*, 2011] Ruoming Jin, Victor E Lee, and Hui Hong. Axiomatic ranking of network role similarity. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 922–930, 2011.

[Karloff *et al.*, 2010] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 938–948. SIAM, 2010.

[Klauck *et al.*, 2014] Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed computation of large-scale graph problems. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 391–410. SIAM, 2014.

[Kusumoto *et al.*, 2014] Mitsuru Kusumoto, Takanori Maehara, and Ken-ichi Kawarabayashi. Scalable similarity search for simrank. In *SIGMOD Conference*, pages 325–336. ACM, 2014.

[Lacki *et al.*, 2020] Jakub Lacki, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. Walking randomly, massively, and efficiently. In *STOC*, pages 364–377. ACM, 2020.

[Lee *et al.*, 2012] Pei Lee, Laks V. S. Lakshmanan, and Jeffrey Xu Yu. On top-k structural similarity search. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, pages 774–785. IEEE Computer Society, 2012.

[Li *et al.*, 2015] Zhenguo Li, Yixiang Fang, Qin Liu, Jiefeng Cheng, Reynold Cheng, and John CS Lui. Walking in the cloud: Parallel simrank at scale. *Proceedings of the VLDB Endowment*, 9(1):24–35, 2015.

[Liao *et al.*, 2022] Ningyi Liao, Dingheng Mo, Siqiang Luo, Xiang Li, and Pengcheng Yin. Scara: scalable graph neural networks with feature-oriented optimization. *Proceedings of the VLDB Endowment*, 15(11):3240–3248, 2022.

[Luo and Zhu, 2024] Siqiang Luo and Zulun Zhu. https://sites.google.com/view/massive-technical-report/, 2024.

[Luo *et al.*, 2012] Siqiang Luo, Yifeng Luo, Shuigeng Zhou, Gao Cong, and Jihong Guan. Disks: a system for distributed spatial group keyword search on road networks. *Proceedings of the VLDB Endowment*, 5(12):1966–1969, 2012.

[Luo *et al.*, 2014] Siqiang Luo, Yifeng Luo, Shuigeng Zhou, Gao Cong, and Jihong Guan. Distributed spatial keyword querying on road networks. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014*, pages 235–246. OpenProceedings.org, 2014.

[Luo *et al.*, 2019] Siqiang Luo, Xiaokui Xiao, Wenqing Lin, and Ben Kao. Efficient batch one-hop personalized pageranks. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1562–1565, 2019.

[Luo *et al.*, 2022] Siqiang Luo, Xiaowei Wu, and Ben Kao. Distributed pagerank computation with improved round complexities. *Information Sciences*, 607:109–125, 2022.

[Luo *et al.*, 2023] Siqiang Luo, Zichen Zhu, Xiaokui Xiao, Yin Yang, Chunbo Li, and Ben Kao. Multi-task processing in vertex-centric graph systems: Evaluations and insights. In *Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023*, pages 247–259. OpenProceedings.org, 2023.

[Luo, 2019] Siqiang Luo. Distributed pagerank computation: An improved theoretical study. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, pages 4496–4503. AAAI Press, 2019.

[Luo, 2020] Siqiang Luo. Improved communication cost in distributed pagerank computation–a theoretical study. In *International Conference on Machine Learning*, pages 6459–6467. PMLR, 2020.

[Maehara *et al.*, 2014] Takanori Maehara, Mitsuru Kusumoto, and Ken-ichi Kawarabayashi. Efficient simrank computation via linearizationpublication of this article pending inquiry. In *KDD*, pages 1426–1435. ACM, 2014.

[Mo and Luo, 2021] Dingheng Mo and Siqiang Luo. Agenda: Robust personalized pageranks in evolving graphs. In *CIKM*, pages 1315–1324. ACM, 2021.

[Nguyen *et al.*, 2015] Phuong Nguyen, Paolo Tomeo, Tommaso Di Noia, and Eugenio Di Sciascio. An evaluation of simrank and personalized pagerank to build a recommender system for the web of data. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1477–1482, 2015.

[Shao *et al.*, 2015] Yingxia Shao, Bin Cui, Lei Chen, Mingming Liu, and Xing Xie. An efficient similarity search framework for simrank over large dynamic graphs. *Proceedings of the VLDB Endowment*, 8(8):838–849, 2015.

[Shi *et al.*, 2020] Jieming Shi, Tianyuan Jin, Renchi Yang, Xiaokui Xiao, and Yin Yang. Realtime index-free single source simrank processing on web-scale graphs. *arXiv preprint arXiv:2002.08082*, 2020.

[Song *et al.*, 2018] Junshuai Song, Xiongcai Luo, Jun Gao, Chang Zhou, Hu Wei, and Jeffrey Xu Yu. Uniwalk: Unidirectional random walk based scalable simrank computation over large graph. *IEEE Trans. Knowl. Data Eng.*, 30(5):992–1006, 2018.

[Symeonidis *et al.*, 2021] Panagiotis Symeonidis, Lidija Kirjackaja, and Markus Zanker. Session-based news recommendations using simrank on multi-modal graphs. *Expert Systems with Applications*, 180:115028, 2021.

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

[Wang *et al.*, 2020a] Hanzhi Wang, Zhewei Wei, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Exact single-source simrank computation on large graphs. In *SIGMOD Conference*, pages 653–663. ACM, 2020.

[Wang *et al.*, 2020b] Yue Wang, Ruiqi Xu, Zonghao Feng, Yulin Che, Lei Chen, Qiong Luo, and Rui Mao. DISK: A distributed framework for single-source simrank with accuracy guarantee. *Proc. VLDB Endow.*, 14(3):351–363, 2020.

[Wang *et al.*, 2021] Hanzhi Wang, Zhewei Wei, Yu Liu, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Exactsim: benchmarking single-source simrank algorithms with high-precision ground truths. *The VLDB Journal*, 30(6):989–1015, 2021.

[Xu *et al.*, 2017] Linchuan Xu, Xiaokai Wei, Jiannong Cao, and Philip S. Yu. On learning mixed community-specific similarity metrics for cold-start link prediction. In *WWW (Companion Volume)*, pages 861–862. ACM, 2017.

[Zhang *et al.*, 2017] Zhipeng Zhang, Yingxia Shao, Bin Cui, and Ce Zhang. An experimental evaluation of simrank-based similarity search algorithms. *Proceedings of the VLDB Endowment*, 10(5):601–612, 2017.

[Zheng *et al.*, 2013] Weiguo Zheng, Lei Zou, Yansong Feng, Lei Chen, and Dongyan Zhao. Efficient simrank-based similarity join over large graphs. *Proceedings of the VLDB Endowment*, 6(7):493–504, 2013.

[Zhou *et al.*, 2019] Kai Zhou, Tomasz P. Michalak, and Yevgeniy Vorobeychik. Adversarial robustness of similarity-based link prediction. In *ICDM*, pages 926–935. IEEE, 2019.

[Zhu *et al.*, 2022] Zulun Zhu, Jiaying Peng, Jintang Li, Liang Chen, Qi Yu, and Siqiang Luo. Spiking graph convolutional networks. In *IJCAI*, pages 2434–2440. ijcai.org, 2022.

[Zhu *et al.*, 2024] Zulun Zhu, Sibo Wang, Siqiang Luo, Dingheng Mo, Wenqing Lin, and Chunbo Li. Personalized pageranks over dynamic graphs–the case for optimizing quality of service. In *Proceedings of the 2024 IEEE 40th International Conference on Data Engineering*, 2024.