ClipSim: A GPU-friendly Parallel Framework for Single-Source SimRank with Accuracy Guarantee

TIANHAO WU, Tsinghua University, China
JI CHENG, Hong Kong University of Science and Technology, China
CHAORUI ZHANG[*], Theory Lab, 2012 Labs, Huawei Technologies, Co. Ltd, China
JIANFENG HOU, Theory Lab, 2012 Labs, Huawei Technologies, Co. Ltd, China
GENGJIAN CHEN, Wuhan University, China
ZHONGYI HUANG, Tsinghua University, China
WEIXI ZHANG, Theory Lab, 2012 Labs, Huawei Technologies, Co. Ltd, China
WEI HAN, Theory Lab, 2012 Labs, Huawei Technologies, Co. Ltd, China
BO BAI, Theory Lab, 2012 Labs, Huawei Technologies, Co. Ltd, China

SimRank is an important metric to measure the topological similarity between two nodes in a graph. In particular, single-source and top-k SimRank has numerous applications in recommendation systems, network analysis, and web mining, etc. Mathematically, given a vertex, the computation of single-machine and single-source SimRank mainly lies in matrix-matrix operations. However, it is almost impossible to directly compute on large graphs. Thus, existing works yield to two main operations: a series of random walks, and sparse matrix and dense vector multiplication operations. This brings about high computation cost for SimRank on large graphs. In real-world applications, there is always the query time and accuracy trade-off, which hinders the computation of high-precision SimRank on large-scale graphs.

To handle this problem, this paper proposes *ClipSim*, the first GPU-friendly parallel framework that accelerates the single-source SimRank on GPU with accuracy guarantee. We design a novel data structure and GPU-friendly parallel algorithms for efficient computation of all the operations of SimRank on GPU. Moreover, our theoretical derivation enables ClipSim to largely reduce the number of random walks required for each node, while maintaining the same theoretical accuracy as the state-of-the-art algorithm, ExactSim. We conduct extensive experiments on real-world and synthetic datasets to demonstrate the accuracy and efficiency of ClipSim. The results show that compared with ExactSim, ClipSim obtains single-source SimRank vectors with the same accuracy and up to 160× faster computation time.

CCS Concepts: • **Mathematics of computing → Graph algorithms**; • **Information systems → Data mining**.

Additional Key Words and Phrases: SimRank, exact computation, random walk, GPU acceleration

[*]Corresponding author.

Authors' addresses: Tianhao Wu, wuth20@mails.tsinghua.edu.cn, Tsinghua University, Beijing, China; Ji Cheng, jchengac@cse.ust.hk, Hong Kong University of Science and Technology, Hong Kong, China; Chaorui Zhang, chaorui.zhang@gmail.com, Theory Lab, 2012 Labs, Huawei Technologies, Co. Ltd, Hong Kong, China; Jianfeng Hou, jfhou@fzu.edu.cn, Theory Lab, 2012 Labs, Huawei Technologies, Co. Ltd, Hong Kong, China; Gengjian Chen, chengengjiancs@qq.com, Wuhan University, Wuhan, China; Zhongyi Huang, zhongyih@tsinghua.edu.cn, Tsinghua University, Beijing, China; Weixi Zhang, zhangweixi1@huawei.com, Theory Lab, 2012 Labs, Huawei Technologies, Co. Ltd, Hong Kong, China; Wei Han, harvey.hanwei@huawei.com, Theory Lab, 2012 Labs, Huawei Technologies, Co. Ltd, Hong Kong, China; Bo Bai, ee.bobbai@gmail.com, Theory Lab, 2012 Labs, Huawei Technologies, Co. Ltd, Hong Kong, China.

2836-6573/2023/5-ART27 $15.00
https://doi.org/10.1145/3588707

ACM Reference Format:
Tianhao Wu, Ji Cheng, Chaorui Zhang, Jianfeng Hou, Gengjian Chen, Zhongyi Huang, Weixi Zhang, Wei Han, and Bo Bai. 2023. ClipSim: A GPU-friendly Parallel Framework for Single-Source SimRank with Accuracy Guarantee. *Proc. ACM Manag. Data* 1, 1, Article 27 (May 2023), 26 pages. https://doi.org/10.1145/3588707

1 INTRODUCTION

Measuring similarities of nodes in a graph has a large variety of applications, such as recommendation systems [11], fraud detection [19] and community discovery [38]. Among the measurements for nodes similarities proposed in recent years [1, 8, 12, 13, 35, 36, 39], SimRank stands out as one of the most popular metrics for its quality performance. However, the computation of SimRank similarities on large graphs remains a daunting task. By the definition of SimRank, $O\left(n^2\right)$ space complexity is inevitable and intractable since there are n^2 node pairs in a graph, where n is the number of nodes in the graph. Due to the large space complexity of the all-pairs SimRank, current works [14, 16, 18, 26, 29, 30, 34, 37] mainly focus on the computation of single-source or top-k SimRank. Namely, given a source node v_i, single-source SimRank returns the similarities between v_i and every node, and top-k SimRank returns k nodes with the highest k similarities w.r.t. node v_i. In this paper, we focus on single-source SimRank. To compute single-source SimRank of a given vertex, there are two main steps: 1) estimation of diagonal correction matrix (matrix $\boldsymbol{D}$), which involves random walk based Monte Carlo sampling; 2) calculation of the SimRank vector, which is essentially a series of sparse matrix and dense vector multiplication operations.

Specifically, [14, 18, 37] proposed Linearization and ParSim, which is the first attempt to calculate single-source SimRank similarities. Then, in [26], SLING combined the iteration method and the Monte Carlo method, which accelerated the computation of single-source SimRank significantly. Afterward, [16] and [34] handled single-source SimRank in dynamic graphs and power-law graphs, respectively. Meanwhile, parallel algorithms for SimRank computation also appeared. In [31], they parallelized the local push based algorithm, which highly enhanced the scalability of their algorithms. However, these aforementioned algorithms did not provide a feasible theoretical precision bound on large graphs. Most recently, [29, 30] proposed ExactSim, which is the first work to provide a theoretical precision bound w.r.t. the number of random walks required for Monte Carlo sampling. ExactSim can obtain high-precision single-source SimRank similarities on large-scale graphs. Nevertheless, to achieve a higher accuracy, ExactSim has to suffer a large amount of computation cost. For instance, to obtain single-source SimRank similarities on the Twitter dataset [15] with error bounded by $\varepsilon = 10^{-6}$, on average takes about an hour on a high-performance server, which makes it impossible to support online query.

Recently, the development of modern heterogeneous hardware like GPU gives the single-source SimRank computation a rethink. [30] mentioned that the multiplication of sparse matrix and dense vector in the computation of single-source SimRank is "embarrassingly parallelizable" on GPU. However, it is not a trivial task, even on CPU [33], to parallelize the simulation of the variant of random walk called $\sqrt{c}$-*walk* required for the estimation of diagonal correction matrix. We may think random walk is easy to parallel because it does not involve any synchronization or atomic operations. As shown in our experiments of Section 5, we explored two different strategies to parallel the random walk involved in estimating the diagonal correction matrix on CPU, and found that for the Twitter dataset, on average the serial algorithm takes 306.35s, while the two parallel algorithms take 204.38s and 324.11s, respectively. As demonstrated, designing a high performance parallel random walk algorithm on CPU is indeed a non-trivial task.

It is even more challenging to design an efficient parallel algorithm on GPU due to the single instruction multiple threads (SIMT) architecture. Concretely, there are mainly three challenges in paralleling high-performance $\sqrt{c}$-walk based Monte Carlo sampling on GPU: 1) random walk facilitates non-contiguous memory access; 2) highly skewed number of random walks required by each vertex leads to severely imbalanced workload; 3) $\sqrt{c}$-walk introduces randomness into the length of each walk which leads to a warp divergence problem. To tackle these three challenges, we propose a GPU-friendly unified parallel framework called *ClipSim* to speed up single-source SimRank computation with accuracy guarantee. Technically, we propose 1) a CSR-based data structure for memory access and thread utilization; 2) a tailored task stealing strategy for load balance and thread reuse; 3) a double random seeds technique to avoid warp divergence. Besides, we further develop a tight theoretical bound to largely reduce the number of random walks for the computation of single-source SimRank without worsening accuracy.

Compare to ExactSim, the state-of-the-art single-source SimRank with precision guarantee on large graphs, with the same theoretical precision as ExactSim, ClipSim reduces the number of random walks required for each node, while estimating the diagonal correction matrix. Combining our new theoretical observation and our GPU-friendly parallel algorithm design, ClipSim achieved up to 160× speed up on synthetic datasets, and up to 80× speed up on real-world datasets. E.g. for the computation of single-source SimRank similarities on the Twitter dataset [15] with absolute error bounded by $\varepsilon = 10^{-6}$, on average ClipSim takes only around 43.31s with one Nvidia Tesla V100-PCIe while ExactSim takes around 3345.24s (almost an hour) with one Intel(R) Xeon(R) Gold 5117 CPU as illustrated in our experiments. ClipSim achieved 77× faster.

To sum up, our main contributions are summarized as follows.

- To the best of our knowledge, we are the first to propose a GPU-friendly unified parallel framework for the acceleration of single-source SimRank computation on large graphs with accuracy guarantee on a single machine.
- From the perspective of high-performance computing, ClipSim optimizes memory access, load balance and warp divergence on GPU. Specifically, we propose
  - a compressed sparse row (CSR)-based data structure for efficient memory access and better thread utilization in the random walk process.
  - a tailored task stealing strategy which improves load-balancing and thread reuse on power-law graphs.
  - a double random seeds technique that avoids warp divergence and improves the utilization efficacy of threads.
- We introduce truncation techniques to estimate single-source SimRank similarities. Besides, we theoretically reduce the number of random walks and provide a tighter accuracy guarantee for the computation of single-source SimRank.
- By fully utilizing the advantages of GPU parallelism and our tighter accuracy guarantee, our extensive experiments demonstrate that the proposed GPU-friendly ClipSim is magnitudes faster than ExactSim, the current state-of-the-art single-source SimRank for large graphs on a single machine. More specifically, our algorithm achieves up to 160× speedup with the same absolute error bounds on large-scale graphs. Furthermore, our algorithm can be used to support the high-precision online query, as the average query time for million-node graphs with error bound $\varepsilon = 10^{-6}$ is less than 4 seconds.

The rest of the paper is organized as follows. Section 2 gives the problem formulation and introduces recent works briefly. The proposed algorithm for single-source SimRank on large-scale graphs is described in Section 3. We parallelize our proposed algorithm on GPU and present the optimizations for GPU parallel in Section 4. The numerical results in Section 5 show the advantage of our GPU-based algorithm. Finally, Section 6 concludes the work.

Table 1. Key Notations and Definitions.

Notation	**Description**
$n,\ m$	the number of nodes and edges in G
$\mathcal{I}(v),\ \mathcal{O}(v)$	the sets of in-neighbors and out-neighbors of node v
$s(u,v)$	the SimRank similarity of nodes u and v
c	the decay factor of SimRank
ε	the absolute error bound for each estimation of SimRank similarity
$\boldsymbol{\pi}_i$	the personalized PageRank vector
$\boldsymbol{\pi}_i^\ell$	the ℓ-hop personalized PageRank vector

2 PRELIMINARIES AND RELATED WORK

2.1 Problem Definition

Given a graph $G = (V, E)$, the SimRank similarity [12] of nodes u and v, denoted as $s(u, v)$, is defined as

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

where $\mathcal{I}(v)$ denotes the set of in-neighbors of node v, and $c \in (0, 1)$ is a *decay factor* which is usually set to 0.6 or 0.8. According to the definition of the SimRank similarity, $s(u, v)$ aggregates the information of multi-hop neighbors of nodes u and v, and thus serves as an effective similarity measure.

There are several variations of SimRank computation problems summarized as follows:

- **All-Pairs**: Output the SimRank similarities of all node pairs in the graph.
- **Single-Source**: Given a query vertex i, return the SimRank scores between node i and each vertex in the graph.
- **Top-*k***: Given a query vertex i, return the top k SimRank scores between the node i and each vertex in the graph.

However, due to the recursive structure, for all of these variations, the computation of exact SimRank similarities is a laborious work on large-scale graphs. Thus, we devote to calculating approximate SimRank similarities. Here, for a graph $G = (V, E)$ with n nodes, the ε-approximate SimRank [30] is defined as

$$\Pr\left[\forall u, v \in V \ \ |\hat{s}(u,v) - s(u,v)| > \varepsilon\right] < \frac{1}{n},$$

where $s(u, v)$ and $\hat{s}(u, v)$ are the exact similarity and approximate similarity between u and v, respectively.

In this paper, we focus on the approximate single-source SimRank similarities computation. The SimRank similarities of all pairs and top-k can be obtained based on the method of single-source SimRank computation.

2.2 Terminologies

- **Similarity Matrix**: Given the set of nodes $V = \{v_1, v_2, \cdots, v_n\}$, similarity matrix $\boldsymbol{S} = [s_{i,j}]_{n\times n}$ is an $n \times n$ matrix, where element $s_{i,j}$ is the SimRank similarity of nodes v_i and v_j.
- **Adjacent Matrix**: Given a graph $G = (V, E)$, adjacent matrix $\boldsymbol{A} = \left[a_{i,j}\right]_{n\times n}$ is an $n \times n$ matrix, where element $a_{i,j} = 1$ if $(v_j, v_i) \in E$. Otherwise $a_{i,j} = 0$.
- **Transition Matrix**: Given a graph $G = (V, E)$, transition matrix $\boldsymbol{P} = \left[p_{i,j}\right]_{n\times n}$ is an $n \times n$ matrix, where element $p_{i,j} = 1/|\mathcal{I}(v_j)|$ for $v_i \in \mathcal{I}(v_j)$, and $p_{i,j} = 0$ otherwise. Another equivalent definition of transition matrix is the column-normalized adjacent matrix.
- **$\sqrt{c}$-Random Walk**: Given a graph $G = (V, E)$, source node $s \in V$ and a constant factor $c \in (0, 1)$, a $\sqrt{c}$-random walk (abbreviated "$\sqrt{c}$-*walk*") from s is defined as follows: A surfer stays at source node s initially. Then, at each step, the surfer stops at current node v (initially be s) with probability $1 - \sqrt{c}$ and moves forward with probability $\sqrt{c}$. When the surfer moves forward, there are two cases. If $\mathcal{I}(v) \neq \emptyset$, the surfer moves to any node $u \in \mathcal{I}(v)$ with equal probability; otherwise, the surfer jumps back to s directly.
- **Personalized PageRank (PPR)**: Given a source node $s \in V$ and a target node $t \in V$, the personalized PageRank (with parameter $\sqrt{c}$) of t with respect of s, denoted as $\boldsymbol{\pi}_s(t)$, is defined as the probability of a $\sqrt{c}$-walk from s stops at t.

2.3 Existing Methods

2.3.1 Monte Carlo (MC). Monte Carlo method for calculating SimRank similarities was first proposed in [6]. The main idea is to compute the SimRank similarities with a new equivalent definition, in which SimRank similarities can be considered as the meeting probabilities of random walks. More specifically, in [26], SimRank similarity between nodes u and v is formulated as

$$s(u, v) = \Pr\left[\text{ two } \sqrt{c}\text{-walks from } u \text{ and } v \text{ meet }\right]. \tag{2}$$

Note that the meeting probability of two $\sqrt{c}$-walks can be estimated by generating a huge number of $\sqrt{c}$-walks, and approximate SimRank similarities are thus obtained with the above equation (2). According to well-known concentration inequalities[1], to compute an ε-approximate similarity $\hat{s}(u, v)$ needs $O\left(\frac{\log n}{\varepsilon^2}\right)$ pairs of $\sqrt{c}$-walks. Therefore, the time complexity of single-source SimRank vector computation is $O\left(\frac{n \log n}{\varepsilon^2}\right)$, which is infeasible for computing high precision SimRank in large-scale graphs.

2.3.2 Linearization. According to the definition of SimRank, [14, 18] and [37] proposed that there exists a diagonal matrix $\boldsymbol{D}$, called *correction matrix*, such that

$$\boldsymbol{S} = c \cdot \boldsymbol{P}^T \boldsymbol{S} \boldsymbol{P} + \boldsymbol{D}, \tag{3}$$

where $\boldsymbol{S}$ and $\boldsymbol{P}$ are similarity matrix and transition matrix, respectively. Consequently, single-source similarity vector with respect to source node v_i can be calculated by

$$\boldsymbol{S} \cdot \boldsymbol{e}_i = \sum_{\ell=0}^{\infty} c^{\ell} \cdot \left(\boldsymbol{P}^{\ell}\right)^T \boldsymbol{D} \boldsymbol{P}^{\ell} \cdot \boldsymbol{e}_i, \tag{4}$$

where $\boldsymbol{e}_i$ is a one-hot vector with the i-th element $\boldsymbol{e}_i(i) = 1$ and other elements being 0. Practically, the approximate single-source similarity vector $\boldsymbol{S}_L \cdot \boldsymbol{e}_i$ is obtained by computing $(L + 1)$ terms of

[1]https://en.wikipedia.org/wiki/Concentration_inequality

the series in (4), i.e.,

$$S_L \cdot \boldsymbol{e}_i = \sum_{\ell=0}^{L} c^\ell \cdot \left(\boldsymbol{P}^\ell\right)^T \boldsymbol{D}\boldsymbol{P}^\ell \cdot \boldsymbol{e}_i, \tag{5}$$

where L is set to $\left\lceil \frac{\log(\varepsilon(1-c))}{\log c} \right\rceil$.

However, to compute the diagonal correction matrix $\boldsymbol{D}$ is difficult in practice. In [18], they adopted an iterative method, where the diagonal elements of $\boldsymbol{D}$ are regarded as variables in a system of linear equations. Nevertheless, the convergence of the iterative method has not yet been proven. A more efficient and guaranteed way was proposed in SLING [26]. Let $\boldsymbol{D}(k,k)$ be the k-th diagonal element of $\boldsymbol{D}$, $\boldsymbol{D}(k,k)$ can be expressed as

$$\boldsymbol{D}(k,k) = 1 - \Pr\left[\text{ two } \sqrt{c}\text{-walks from } v_k \text{ meet }\right]. \tag{6}$$

Then, similar to MC method in Subsubsection 2.3.1, the meeting probability can be estimated by generating a large number of pairs of $\sqrt{c}$-walks. More specifically, to obtain ε-approximate single-source SimRank vector, the estimation of each $\boldsymbol{D}(k,k)$ needs $O\left(\frac{\log n}{\varepsilon^2}\right)$ pairs of $\sqrt{c}$-walks. Thus, it also incurs $O\left(\frac{n \log n}{\varepsilon^2}\right)$ time complexity, which is the same order of magnitude as the complexity of MC method.

2.3.3 ExactSim. ExactSim [29, 30] is the state-of-the-art method in high precision single-source SimRank computation so far, thus being used as a benchmark [29] for single-source SimRank computation. The main idea of ExactSim is to reduce the number of $\sqrt{c}$-walks for estimation of correction matrix $\boldsymbol{D}$ in Subsubsection 2.3.2. Note that the importance of each node in a graph is different, on account of its connectivity with other nodes. Thus, the number of $\sqrt{c}$-walks for estimating each $\boldsymbol{D}(k,k)$ should also be different. In ExactSim method, with respect to a source node v_i, the number of $\sqrt{c}$-walks for each $\boldsymbol{D}(k,k)$ is set as $O\left(\frac{\boldsymbol{\pi}_i(k)\cdot\log n}{\varepsilon^2}\right)$, where $\boldsymbol{\pi}_i(k)$ is the personalized PageRank (PPR) of v_k with respect to v_i. Hence, the time complexity of this method is $O\left(\frac{\log n}{\varepsilon^2}\right)$ with the fact $\sum_{k=1}^{n} \boldsymbol{\pi}_i(k) = 1$.

2.4 GPU Architecture

GPU is the acronym for Graphics Processing Unit. In recent years, GPU has been adopted in general parallel computing tasks, e.g., graph algorithms [25, 32]. In this subsection, we introduce the basic architecture of GPU and the differences between GPU and CPU.

2.4.1 Basic Concepts in GPU.

- **Kernels and Thread**: Kernels are extended C++ functions, which are executed N times in parallel by N different CUDA threads. In CUDA, threads are managed in a *warp-block-grid* hierarchy. *Warp*, the basic unit of execution, consists of 32 threads. Then, for convenience of computation across vectors and matrices, warps can be organized into 1, 2, 3-dim *blocks*. Similarly, blocks can also be organized into 1, 2, 3-dim *grids*. Usually, each block contains at most 1, 024 threads (i.e., 32 warps) and each grid contains at most 65, 536 blocks.
- **Memory Hierarchy**: Graphics memory can be divided into categories according to the action scope. First of all, each thread has *private local memory*. Then, each thread in the same warp shares data within *registers*. Next, threads in the same block share data via *shared memory*, which has the same lifetime as the block. Finally, all threads have access to *global memory*, which is the largest but slowest among the memory mentioned above.

- **Atomic Operations**: The word "atomic" means "inseparable like an atom". Consequently, atomic operations means the operations execute absolutely with no interruption. It is crucial to utilize atomic operations when multiple threads update the value of the same variable simultaneously in the parallel computing. On the other hand, atomic operations can only be executed sequentially, which worsen the performance of parallel algorithms. Thus, the number of atomic operations should be as few as possible.
- **Single Instruction Multiple Threads (SIMT)**: There are several *streaming multiprocessors* (SMs) in a GPU. Each multiprocessor executes and schedules threads in groups of 32 parallel threads called *warps*. SIMT means that threads in the same warp execute a common instruction at a time. If there are several blocks of threads executing in parallel, they are divided into warps and handled by warp scheduler. Note that the synchronization of 32 threads in a warp is quite important. When threads in the same warp diverge due to a data-dependent conditional branch instruction, the GPU executes all the branches of the instruction sequentially. This is called *warp divergence*, which will be discussed in detail in Subsubsection 4.4.

2.4.2 Difference between GPU and CPU. We summarizes three major differences as follows.

- **Latency and Throughput**: Latency means the time taken to finish a given task, while throughput means the number of tasks finished within a given period. Compared with CPU, there are less cache and more cores in GPU. In memory hierarchy, DRAM is about an order of magnitude slower than cache. Larger cache means less addressing operations in DRAM and less latency. Thus, GPU has larger latency than CPU. To make up for this point, GPU has more computational cores and largely increases the throughput, which means it can operate on massive data at once.
- **Multi-core Processor**: There are much more cores in a GPU processor. As a result, there can be much more available threads in parallel computing. However, a single GPU core is significantly slower than a single CPU core, especially in the sophisticated tasks (e.g., recursion). Thus, parallelization is the major issue in GPU computing, where each GPU core is utilized for calculation efficiently.
- **Applications**: One thing needs to be emphasized, CPU can never be fully replaced by GPU, and vice versa. The two kinds of processors are designed for different applications. Generally, CPU is used for complicated tasks or algorithms which is difficult to parallelize. CPU can process the data sequentially with its powerful single core and large cache for accelerating. While GPU is more suitable for highly-parallel and repetitive computing tasks, such as matrix multiplication. Specifically, in our SimRank computation, based on the Linearization method introduced in Subsubsection 2.3.2, there are lots of matrix-vector multiplication operations, which is exactly the opportunity to harness the power of GPU.

3 CLIPSIM ALGORITHM

3.1 Motivation

We propose a new strategy for the construction of diagonal correction matrix D, called *cutting at both ends (CutEnds). CutEnds* strategy consists of two parts, *LargeCut* and *SmallCut.* Our idea can be summarized in two aspects:

- According to *law of large numbers* [5], with a large number of trials, the average of the results should be close to the expectation. For instance, tossing a coin for $10,000$ times and $100,000$ times, the ratios of times that getting heads are nearly the same, i.e., both nearly 0.5. Thus, it is possible to reduce the number of $\sqrt{c}$-walks starting from v_k, where corresponding PageRank value $\boldsymbol{\pi}_i(k)$ is sufficiently large.

- Note that the SimRank similarity between node v_i and v_j can be formulated as [34]

$$S(i,j) = \frac{1}{(1-\sqrt{c})^2} \sum_{\ell=0}^{\infty} \sum_{k=1}^{n} \boldsymbol{\pi}_i^{\ell}(k) \cdot \boldsymbol{\pi}_j^{\ell}(k) \cdot \boldsymbol{D}(k,k), \tag{7}$$

where $\boldsymbol{\pi}_i^{\ell}(k)$ is the k-th element of the *ℓ-hop personalized PageRank vector* $\boldsymbol{\pi}_i^{\ell}$ with respect to v_i. $\boldsymbol{\pi}_i^{\ell} = \left(1-\sqrt{c}\right)\left(\sqrt{c}\boldsymbol{P}\right)^{\ell} \cdot \boldsymbol{e}_i$. In fact, $\boldsymbol{\pi}_i^{\ell}(k)$ is the probability that a $\sqrt{c}$-walk from node v_i stops at node v_k in exactly ℓ steps and $\boldsymbol{\pi}_i(k) = \sum_{\ell=0}^{\infty} \boldsymbol{\pi}_i^{\ell}(k)$. The definition of $\boldsymbol{\pi}_j^{\ell}(k)$ is exactly similar. Thus, when PageRank value $\boldsymbol{\pi}_i(k)$ is sufficiently small, it is possible to obtain satisfactory SimRank similarity without operating $\sqrt{c}$-walks from node v_k.

3.2 ClipSim

Based on *CutEnds* strategy for the estimation of D, we propose *ClipSim*, a new algorithm for single-source SimRank computation. The algorithm details of our ClipSim are as follows.

- **Set the number of pairs of $\sqrt{c}$-walks.** In ExactSim algorithm, the number of pairs of $\sqrt{c}$-walks for the estimation of correction matrix $\boldsymbol{D}$, denoted as R, is $O\left(\frac{\log n}{\varepsilon^2}\right)$. According to our analysis in Subsection 3.3, the number of samples can be reduced by a factor "$\|\boldsymbol{\pi}_i\|_2$", which is the L_2-norm of the PPR vector of v_i. That is,

$$R = \frac{75\left(1+\frac{1}{\sqrt{n}}\right) \cdot \|\boldsymbol{\pi}_i\|_2 \cdot \log n}{8\left(1-\sqrt{c}\right)^4 \varepsilon^2}. \tag{8}$$

For each $v_k \in V$, the number of pairs of $\sqrt{c}$-walks from v_k, denoted as $R(k)$, is

$$R(k) = \lceil R \cdot \boldsymbol{\pi}_i(k) \rceil. \tag{9}$$

- **Truncation for Large PageRank (LargeCut).** For node v_k with PPR value $\boldsymbol{\pi}_i(k)$ larger than $\frac{(1-\sqrt{c})^2 \cdot \varepsilon}{10 \cdot c}$, the number of $\sqrt{c}$-walks from v_k after applying LargeCut, denoted as $r(k)$, can be expressed as

$$r(k) = \left\lfloor R\boldsymbol{\pi}_i(k) \cdot \frac{5\boldsymbol{\pi}_i(k)}{\|\boldsymbol{\pi}_i\|_2 + 5\boldsymbol{\pi}_i(k)} \right\rfloor. \tag{10}$$

- **Truncation for Small PageRank (SmallCut).** For node v_k with PPR value $\boldsymbol{\pi}_i(k)$ smaller than $\frac{(1-\sqrt{c})^2 \cdot \varepsilon}{10 \cdot c}$, the number of pairs of $\sqrt{c}$-walks from v_k is set to be zero.

The pseudo code of ClipSim is summarized in Algorithm 1.

3.3 Theoretical Analysis

3.3.1 Time and Space Complexity. The time complexity of Algorithm 1 is analyzed as follows.

- **Sparse Matrix and Dense Vector Multiplication.** According to the definition in (5), given the matrix $\boldsymbol{D}$, there are $O\left(\sum_{\ell=0}^{L}(2\ell+1)\right) = O\left(L^2\right)$ matrix-vector multiplication and each multiplication takes $O\left(m\right)$ time, where m is the number of edges in graph G (also is the number of nonzero elements of matrix $\boldsymbol{P}$). Thus, the time complexity of matrix-vector multiplication is $O\left(m \log^2 \frac{1}{\varepsilon}\right)$. A general technique used in [18, 37] is to precompute each vector $\boldsymbol{u}_{\ell} = \boldsymbol{P}^{\ell} \cdot \boldsymbol{e}_i$, $(\ell = 0, 1, \cdots, L)$, and then calculate the single-source SimRank vector with Horner's method [10]. The time complexity is reduced to $O\left(m \log \frac{1}{\varepsilon}\right)$ with $O\left(nL\right) = O\left(n \log \frac{1}{\varepsilon}\right)$ additional space usage.

Algorithm 1 ClipSim Algorithm.

```
Input: Graph $G$, Transition matrix $\boldsymbol{P}$, source node $v_i$, maximum error $\varepsilon$ and decay factor $c$.
Output: Single-source SimRank vector $\boldsymbol{s}^L = \boldsymbol{S}_L \cdot \boldsymbol{e}_i$.
 1: $L \leftarrow \left\lceil \log_{\frac{1}{c}} \frac{10}{\varepsilon} \right\rceil$.
 2: $\boldsymbol{\pi}_i^0 \leftarrow (1-\sqrt{c}) \cdot \boldsymbol{e}_i,\ \boldsymbol{\pi}_i \leftarrow (1-\sqrt{c}) \cdot \boldsymbol{e}_i$.
 3: for $\ell$ from 1 to $L$ do
 4:     $\boldsymbol{\pi}_i^\ell \leftarrow \sqrt{c}\boldsymbol{P} \cdot \boldsymbol{\pi}_i^{\ell-1}$.
 5:     $\boldsymbol{\pi}_i \leftarrow \boldsymbol{\pi}_i + \boldsymbol{\pi}_i^\ell$.
 6: end for
 7: $R \leftarrow \frac{75\left(1+\frac{1}{\sqrt{n}}\right)\cdot\|\boldsymbol{\pi}_i\|_2\cdot\log n}{8\left(1-\sqrt{c}\right)^4\varepsilon^2}$.
 8: Call Algorithm 2 with PPR vector $\boldsymbol{\pi}_i$ to obtain the number of samples $R(k)$ $(k = 1, \cdots, n)$.
 9: for $k$ from 1 to $n$ do
10:     Call Algorithm 3 with $R(k)$ to obtain the estimation $\hat{\boldsymbol{D}}(k,k)$ for $\boldsymbol{D}(k,k)$.
11: end for
12: $\boldsymbol{s}^0 \leftarrow \frac{1}{1-\sqrt{c}}\hat{\boldsymbol{D}} \cdot \boldsymbol{\pi}_i^L$.
13: for $\ell$ from 1 to $L$ do
14:     $\boldsymbol{s}^\ell \leftarrow \sqrt{c}P^T \cdot \boldsymbol{s}^{\ell-1} + \frac{1}{1-\sqrt{c}}\hat{\boldsymbol{D}} \cdot \boldsymbol{\pi}_i^{L-\ell}$.
15: end for
16: return $\boldsymbol{s}^L$.
```

Algorithm 2 Clipping Method for the Number of $\sqrt{c}$-walks.

```
Input: Personalized PageRank vector $\boldsymbol{\pi}_i$ with respect to source node $v_i$, total number of $\sqrt{c}$-walks
    $R$, maximum error $\varepsilon$ and decay factor $c$.
Output: The number of $\sqrt{c}$-walks $R(k)$ for each node $v_k$ $(k = 1, 2, \cdots, n)$.
 1: for $k$ from 1 to $n$ do
 2:     Original number of $\sqrt{c}$-walks $\hat{R}(k) \leftarrow \lceil R \cdot \boldsymbol{\pi}_i(k) \rceil$.
 3:     if $\boldsymbol{\pi}_i(k) > \frac{(1-\sqrt{c})^2\cdot\varepsilon}{10\cdot c}$ and $\frac{\|\boldsymbol{\pi}_i\|_2}{5\boldsymbol{\pi}_i(k)} \geq 5$ then
 4:         $R(k) \leftarrow \left\lfloor R \cdot \boldsymbol{\pi}_i(k) \cdot \frac{5\boldsymbol{\pi}_i(k)}{\|\boldsymbol{\pi}_i\|_2 + 5\boldsymbol{\pi}_i(k)} \right\rfloor$.
 5:     else if $\boldsymbol{\pi}_i(k) \leq \frac{(1-\sqrt{c})^2\cdot\varepsilon}{10\cdot c}$ then
 6:         $R(k) \leftarrow 0$.
 7:     end if
 8: end for
 9: return $R(k)$ $(k = 1, 2, \cdots, n)$.
```

- **Basic Sampling for $\boldsymbol{D}$ estimation.** When the number of $\sqrt{c}$-walks from each node v_k is determined according to the corresponding PPR value $\boldsymbol{\pi}_i(k)$, the total number of $\sqrt{c}$-walks R can be $O\left(\frac{\|\boldsymbol{\pi}_i\|_2\cdot\log n}{\varepsilon^2}\right)$. Thus, the complexity of $\boldsymbol{D}$ estimation is $O\left(\frac{\|\boldsymbol{\pi}_i\|_2\cdot\log n}{\varepsilon^2}\right)$. Note that $\|\boldsymbol{\pi}_i\|_2 \leq \sum_{k=1}^n \boldsymbol{\pi}(k) = 1$. Thus, compared with the state-of-the-art algorithm, namely ExactSim, our proposed algorithm provides at least the same time complexity when using the same samples allocation scheme. Moreover, the complexity of sampling can be much smaller than 1 on scale-free graphs. The theoretical details and further optimization will be discussed in Subsubsection 3.3.2.

Algorithm 3 Sampling Method for Estimating $\boldsymbol{D}(k,k)$.

```
Input: Graph $G$, node $v_k$, number of samples $R(k)$ and factor $c$.
Output: Estimation $\hat{\boldsymbol{D}}(k,k)$ for $\boldsymbol{D}(k,k)$.
1: if $|\mathcal{I}(v_k)|$ is zero then
2:     $\boldsymbol{D}(k,k) \leftarrow 1$.
3:     return $\boldsymbol{D}(k,k)$.
4: end if
5: if $R(k)$ is zero then
6:     $\boldsymbol{D}(k,k) \leftarrow 1 - \frac{c}{|\mathcal{I}(v_k)|}$.
7:     return $\boldsymbol{D}(k,k)$.
8: end if
9: Initialize $q \leftarrow 0$.
10: for $x$ from 1 to $R(k)$ do
11:     Select two nodes $v_k^{(1)}$, $v_k^{(2)}$ from $\mathcal{I}(v_k)$ uniformly.
12:     if $v_k^{(1)} \neq v_k^{(2)}$ then
13:         Sample two independent $\sqrt{c}$-walks from $v_k^{(1)}$ and $v_k^{(2)}$, respectively.
14:         if the two $\sqrt{c}$-walks meet then
15:             $q \leftarrow q + 1$.
16:         end if
17:     end if
18: end for
19: $\boldsymbol{D}(k,k) \leftarrow 1 - \frac{c}{|\mathcal{I}(v_k)|} - c \cdot \frac{q}{R(k)}$.
20: return $\boldsymbol{D}(k,k)$.
```

- **CutEnds strategy.** Given that we distribute $R \cdot \boldsymbol{\pi}_i(k)$ samples to estimate each $\boldsymbol{D}(k,k)$, both *LargeCut* and *SmallCut* in our *CutEnds* strategy can further reduce the number of samples significantly. *LargeCut* can reduce the number of sampling walks by a factor "$\|\boldsymbol{\pi}_i\|_2$". Thus, the time complexity of estimating $\boldsymbol{D}$ can be $O\left(\frac{\|\boldsymbol{\pi}_i\|_2^2 \cdot \log n}{\varepsilon^2}\right)$. Experimentally, according to *SmallCut* strategy, we can estimate correction matrix $\boldsymbol{D}$ with $\sqrt{c}$-walks starting from nearly 45% nodes truncated on large scale-free graphs, which is shown in Fig.1. Fig.1 illustrates the distribution of the number of random walks for graph500-scale23. The random walks in the green part can be truncated by SmallCut. We can see there are 7 purple points in the green part. For instance, if the horizontal and vertical coordinates of a point are 3 and 10^6, it means that there are 10^6 vertices from each of which the number of random walks is 3.

We store the transition matrix $\boldsymbol{P}$ of graph G with compressed sparse row (CSR) format, which needs $(2m+n)$ space. Moreover, the precomputed vectors $\boldsymbol{u}_i$, $(\ell = 0, 1, \cdots, L)$ mentioned above needs $O(nL)$ space. Thus, the space complexity of Algorithm 1 is $O\left(m + n \log \frac{1}{\varepsilon}\right)$. To sum up, the time and space complexity of ClipSim algorithm are $O\left(\frac{\|\boldsymbol{\pi}_i\|_2^2 \cdot \log n}{\varepsilon^2} + m \log \frac{1}{\varepsilon}\right)$ and $O\left(m + n \log \frac{1}{\varepsilon}\right)$, respectively.

3.3.2 *Precision Guarantee.*

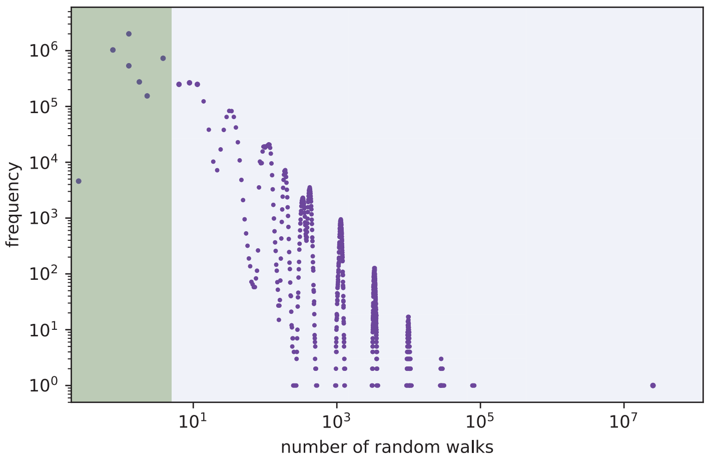


Fig. 1. The number of random walks distribution in graph500-scale23. Random walks in the green part are truncated by SmallCut.

Theorem. *[Hoeffding's Inequality] [9] Let $X_1, X_2, \cdots, X_n$ be independent random variables with $a_i \le X_i \le b_i$ almost surely. Let $S_n = X_1 + X_2 + \cdots + X_n$, we have*

$$\Pr\left(|S_n - \mathrm{E}\left[S_n\right]| \ge t\right) \le 2\exp\left(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right).$$

Lemma 1. *The truncation of series in* (5) *introduces an extra additive error of* $\varepsilon/10$.

Proof. Note that we set $L = \left\lceil \log_{\frac{1}{c}} \frac{10}{\varepsilon} \right\rceil$. According to the definition of SimRank (4) formulated in Linearzation, the extra error incurred by L truncation is at most

$$c^L \le c^{\log_{\frac{1}{c}} \frac{10}{\varepsilon}} = \frac{\varepsilon}{10}.$$

□

Theorem 1. *Despite the error of truncated series, for any source node $v_i \in V$, the basic ClipSim without CutEnds strategy provides single-source SimRank vector $\hat{\mathbf{s}} = \hat{\mathbf{S}} \cdot \boldsymbol{e}_i$, such that for any node $v_j \in V$,*

$$\left|\hat{s}(j) - s(v_i, v_j)\right| > \frac{2\varepsilon}{5}$$

with probability at most $1/n$.

Proof. Let $\hat{\boldsymbol{D}}_r(k,k)$ be a random variable indicating whether r-th pair of $\sqrt{c}$-walks meets or not. If r-th pair of $\sqrt{c}$-walks meets, $\hat{\boldsymbol{D}}_r(k,k) = 0$. Otherwise, $\hat{\boldsymbol{D}}_r(k,k) = 1$. The total number of $\sqrt{c}$-walks pairs and the number of pairs from node v_k are denoted as R and $R(k)$, respectively. Here, $R(k) = \lceil R \cdot \boldsymbol{\pi}_i(k) \rceil$. Then,

$$\hat{\boldsymbol{D}}(k,k) = \frac{\sum_{r=1}^{R(k)} \hat{\boldsymbol{D}}_r(k,k)}{R(k)},$$

which is an estimator of the k-th value $\boldsymbol{D}(k,k)$ in the correction matrix $\boldsymbol{D}$ and $\mathbb{E}\left[\hat{\boldsymbol{D}}(k,k)\right] = \boldsymbol{D}(k,k)$. Thus, the approximate SimRank value between v_i and v_j is

$$\hat{\mathbf{s}}_i(j) = \frac{1}{(1-\sqrt{c})^2} \sum_{\ell=0}^{\infty} \sum_{k=1}^{n} \boldsymbol{\pi}_i^{\ell}(k) \cdot \boldsymbol{\pi}_j^{\ell}(k) \cdot \hat{\boldsymbol{D}}(k,k), \tag{11}$$

and $\mathbb{E}\left[\hat{\mathbf{s}}_i(j)\right] = s(v_i, v_j)$. To bound the failure probability of the approximate SimRank value for any $v_i, v_j \in V$, we express $\hat{\mathbf{s}}_i(j)$ as a summation of several independent random variables. Note that $R(k) = \lceil R \cdot \boldsymbol{\pi}_i(k) \rceil \geq R \cdot \boldsymbol{\pi}_i(k)$, there exists constant $0 \leq \delta_k < 1$ such that

$$\begin{aligned} \hat{\mathbf{s}}_i(j) &= \frac{1}{(1-\sqrt{c})^2} \sum_{k=1}^{n} \sum_{r=1}^{R(k)} \frac{\sum_{\ell=0}^{\infty} \boldsymbol{\pi}_i^{\ell}(k) \cdot \boldsymbol{\pi}_j^{\ell}(k)}{R(k)} \cdot \hat{\boldsymbol{D}}_r(k,k) \\ &= \frac{1}{(1-\sqrt{c})^2} \sum_{k=1}^{n} \sum_{r=1}^{R(k)} \frac{(1-\delta_k) \sum_{\ell=0}^{\infty} \boldsymbol{\pi}_i^{\ell}(k) \cdot \boldsymbol{\pi}_j^{\ell}(k)}{R \cdot \boldsymbol{\pi}_i(k)} \cdot \hat{\boldsymbol{D}}_r(k,k). \end{aligned} \tag{12}$$

Let

$$\hat{\mathcal{D}}_r(k) = \frac{(1-\delta_k) \sum_{\ell=0}^{\infty} \boldsymbol{\pi}_i^{\ell}(k) \cdot \boldsymbol{\pi}_j^{\ell}(k)}{\boldsymbol{\pi}_i(k)} \cdot \hat{\boldsymbol{D}}_r(k,k), \tag{13}$$

$$\hat{\mathbf{s}}_i(j) = \sum_{k=1}^{n} \sum_{r=1}^{R(k)} \frac{\hat{\mathcal{D}}_r(k)}{(1-\sqrt{c})^2 R}.$$

Then, we calculate the bound of random variable $\hat{\mathcal{D}}_r(k)$. By the Cauchy-Schwarz inequality, and the facts $\boldsymbol{\pi}_i(k) = \sum_{\ell=0}^{\infty} \boldsymbol{\pi}_i^{\ell}(k)$ and $\hat{\boldsymbol{D}}_r(k,k) \leq 1$, we have

$$\begin{aligned} \hat{\mathcal{D}}_r(k) &\leq \frac{\sum_{\ell=0}^{\infty} \boldsymbol{\pi}_i^{\ell}(k) \cdot \boldsymbol{\pi}_j^{\ell}(k)}{\boldsymbol{\pi}_i(k)} \\ &\leq \frac{\sum_{\ell=0}^{\infty} \boldsymbol{\pi}_i^{\ell}(k) \cdot \sum_{\ell=0}^{\infty} \boldsymbol{\pi}_j^{\ell}(k)}{\boldsymbol{\pi}_i(k)} \leq \boldsymbol{\pi}_j(k). \end{aligned} \tag{14}$$

The summation of the squared bound of each random variable $\frac{\hat{\mathcal{D}}_r(k)}{(1-\sqrt{c})^2 R}$ is

$$\begin{aligned} &\frac{1}{(1-\sqrt{c})^4 R^2} \sum_{k=1}^{n} \sum_{r=1}^{R(k)} \boldsymbol{\pi}_j(k)^2 \\ \leq &\frac{1}{(1-\sqrt{c})^4 R^2} \sum_{k=1}^{n} \left(R \cdot \boldsymbol{\pi}_i(k) + 1\right) \boldsymbol{\pi}_j(k)^2 \\ \leq &\frac{1}{(1-\sqrt{c})^4 R^2} \left[R \left(\sum_{k=1}^{n} \boldsymbol{\pi}_i(k)^2 \right)^{\frac{1}{2}} \left(\sum_{k=1}^{n} \boldsymbol{\pi}_j(k)^4 \right)^{\frac{1}{2}} + \sum_{k=1}^{n} \boldsymbol{\pi}_j(k)^2 \right] \\ \leq &\frac{R \left\| \boldsymbol{\pi}_i \right\|_2 + 1}{(1-\sqrt{c})^4 R^2} \end{aligned}$$

According to Hoeffding's inequality [9],

$$\Pr\left[\left| \hat{\mathbf{s}}_i(j) - s(v_i, v_j) \right| > \frac{2\varepsilon}{5} \right] = \Pr\left[\left| \hat{\mathbf{s}}_i(j) - \mathbb{E}\left[\hat{\mathbf{s}}_i(j)\right] \right| > \frac{2\varepsilon}{5} \right] < \frac{1}{n^3}.$$

when

$$\frac{R}{\|\boldsymbol{\pi}_i\|_2 + \frac{1}{R}} \geq \frac{75 \log n}{8\left(1-\sqrt{c}\right)^4 \varepsilon^2}.$$

We discuss two cases: 1) When $1/R > \|\boldsymbol{\pi}_i\|_2/\sqrt{n}$, with the fact that the lower bound of $\|\boldsymbol{\pi}_i\|_2$ is $\sqrt{1/n}$, R is at most n under this condition. 2) When $1/R \leq \|\boldsymbol{\pi}_i\|_2/\sqrt{n}$, R is at most

$$\frac{75\left(1+\frac{1}{\sqrt{n}}\right)\|\boldsymbol{\pi}_i\|_2 \cdot \log n}{8(1-\sqrt{c})^4\varepsilon^2}.$$

Note that the time complexity of matrix-vector multiplication in ClipSim is $O(m\log(1/\varepsilon))$. Thus, n can be negligible compared to $O(m\log(1/\varepsilon))$. To ensure the error bound for all the nodes $v_i, v_j \in V$, we have

$$\Pr\left[\forall v_i, v_j \in V,\ \left|\hat{s}_i(j) - s(v_i, v_j)\right| > \frac{2\varepsilon}{5}\right] < \frac{1}{n}. \tag{15}$$

□

Lemma 2. *LargeCut introduces an extra additive error of* $2\varepsilon/5$ *and cuts off a fraction* $\frac{5\boldsymbol{\pi}_i(k)}{\|\boldsymbol{\pi}_i\|_2+5\boldsymbol{\pi}_i(k)}$ *of* $\sqrt{c}$*-walks samples from node* v_k *with* $\boldsymbol{\pi}_i(k) \geq \frac{(1-\sqrt{c})^2\cdot\varepsilon}{10\cdot c}$ *and* $\frac{\|\boldsymbol{\pi}_i\|_2}{5\boldsymbol{\pi}_i(k)} \geq 5$.

Proof. For each k with $\boldsymbol{\pi}_i(k) < \frac{(1-\sqrt{c})^2\cdot\varepsilon}{10\cdot c}$, the number of random walk can be set to 0 according to Lemma 3. Thus, here we only consider the condition $\boldsymbol{\pi}_i(k) \geq \frac{(1-\sqrt{c})^2\cdot\varepsilon}{10\cdot c}$. Let the numbers of pairs of $\sqrt{c}$-walks from v_k before and after LargeCut are $R(k)$ and $r(k)$, respectively. Suppose

$$r(k) = \left\lfloor \frac{1}{p(k)+1} \cdot R\boldsymbol{\pi}_i(k) \right\rfloor, \tag{16}$$

where $p(k)$ is an undetermined function with respect to k. For the sake of simplicity, we assume that $r(k) \geq 1$. This assumption adds at most n random walks, which is negligible compared to the time complexity $O\left(m\log\left(1/\varepsilon\right)\right)$ of matrix-vector multiplication. Note that $R(k) = \lceil R\cdot\boldsymbol{\pi}_i(k)\rceil \geq R\cdot\boldsymbol{\pi}_i(k)$ and $\lfloor x \rfloor \geq x/2$ $(x \geq 1)$, there exist constants $0 \leq \mu_k < 1$ and $0 \leq \lambda_k < 2p(k)+1$, such that

$$\begin{aligned}
&\frac{\sum_{t=1}^{R(k)} \hat{\boldsymbol{D}}_t(k,k)}{R(k)} - \frac{\sum_{t=1}^{r(k)} \hat{\boldsymbol{D}}_t(k,k)}{r(k)} \\
=&\frac{(1-\mu_k)\sum_{t=1}^{R(k)} \hat{\boldsymbol{D}}_t(k,k)}{R\boldsymbol{\pi}_i(k)} - \frac{(2p(k)+2-\lambda_k)\sum_{t=1}^{r(k)} \hat{\boldsymbol{D}}_t(k,k)}{R\boldsymbol{\pi}_i(k)} \\
=&\frac{(1-\mu_k)\sum_{t=r(k)+1}^{R(k)} \hat{\boldsymbol{D}}_t(k,k) - (2p(k)+1+\mu_k-\lambda_k)\sum_{t=1}^{r(k)} \hat{\boldsymbol{D}}_t(k,k)}{R\boldsymbol{\pi}_i(k)}.
\end{aligned}$$

The SimRank values calculated by $R(k)$ and $r(k)$ samples for each k are denoted as $\hat{\mathbf{s}}_i^R(j)$ and $\hat{\mathbf{s}}_i^r(j)$, respectively. The difference between $\hat{\mathbf{s}}_i^R(j)$ and $\hat{\mathbf{s}}_i^r(j)$ is

$$\frac{1}{(1-\sqrt{c})^2}\sum_{k=1}^{n}\frac{\sum_{\ell=0}^{L}\boldsymbol{\pi}_i^\ell(k)\cdot\boldsymbol{\pi}_j^\ell(k)}{R\boldsymbol{\pi}_i(k)}\cdot\left((1-\mu_k)\sum_{t=r(k)+1}^{R(k)}\hat{\boldsymbol{D}}_t(k,k) - (2p(k)+1+\mu_k-\lambda_k)\sum_{t=1}^{r(k)}\hat{\boldsymbol{D}}_t(k,k)\right)$$

Similar to the proof of Theorem 1, the summation of the squared bound of each random variable is

$$\begin{aligned}&\frac{1}{\left(1-\sqrt{c}\right)^4 R^2}\sum_{k=1}^{n}\boldsymbol{\pi}_j(k)^2\cdot\left(R(k)-r(k)+(2p(k)+2)^2\cdot r(k)\right)\\ \leq&\frac{1}{\left(1-\sqrt{c}\right)^4 R^2}\sum_{k=1}^{n}\boldsymbol{\pi}_j(k)^2\cdot\left(\left(4p(k)+5\right)R\boldsymbol{\pi}_i(k)+1\right)\end{aligned}$$

Here, we assume that $p(k)\geq 5$. For $p(k)<5$, LargeCut is not applied for the estimation of matrix $\boldsymbol{D}$. Then, let

$$p(k)=\frac{\|\boldsymbol{\pi}_i\|_2}{5\boldsymbol{\pi}_i(k)}. \tag{17}$$

According to Hoeffding's inequality [9], we have

$$\Pr\left[\left|\hat{\mathbf{s}}_i^R(j)-\hat{\mathbf{s}}_i^r(j)\right|>\frac{2\varepsilon}{5}\right]<\frac{1}{n^3}. \tag{18}$$

when

$$r(k)=\left\lfloor\frac{5\boldsymbol{\pi}_i(k)}{\|\boldsymbol{\pi}_i\|_2+5\boldsymbol{\pi}_i(k)}\cdot R\boldsymbol{\pi}_i(k)\right\rfloor. \tag{19}$$

Finally, to ensure the error bound for all the nodes $v_i, v_j\in V$, we have

$$\Pr\left[\forall v_i, v_j\in V,\ \left|\hat{\mathbf{s}}_i(j)-s(v_i,v_j)\right|>\frac{2\varepsilon}{5}\right]<\frac{1}{n}. \tag{20}$$

□

LEMMA 3. *SmallCut introduces an extra additive error of* $\varepsilon/10$ *and obtains SimRank without* $\sqrt{c}$*-walks sampling from node* v_k *whose corresponding PPR value* $\boldsymbol{\pi}_i(k)<\frac{(1-\sqrt{c})^2\cdot\varepsilon}{10\cdot c}$.

PROOF. Given that the number of $\sqrt{c}$-walks pairs from v_k is $R(k)$, suppose the number of pairs that meet sometime after the first step is m_k. Thus, we have

$$\hat{\boldsymbol{D}}(k,k)=1-c\cdot\frac{1}{d_{in}(v_k)}-c\cdot\frac{m_k}{R(k)},$$

where $d_{in}(v_k)$ is the in-degree of node v_k. If we cancel sampling $\sqrt{c}$-walks, the incurred error for the estimator $\hat{\boldsymbol{D}}(k,k)$ can be at most c. For the node v_k whose corresponding PPR value

$$\boldsymbol{\pi}_i(k)<\frac{(1-\sqrt{c})^2\cdot\varepsilon}{10\cdot c},$$

with the fact $m_k\leq R(k)$, the error of the approximate SimRank value between v_i and v_j incurred by SmallCut is at most

$$\begin{aligned}err^L(j)&=\frac{1}{(1-\sqrt{c})^2}\sum_{\ell=0}^{L}\sum_{k\in S}\boldsymbol{\pi}_i^\ell(k)\cdot\boldsymbol{\pi}_j^\ell(k)\cdot c\cdot\frac{m_k}{R(k)}\\ &\leq\frac{c}{(1-\sqrt{c})^2}\cdot\frac{(1-\sqrt{c})^2\cdot\varepsilon}{10\cdot c}\sum_{k\in S}\boldsymbol{\pi}_j(k)\\ &\leq\frac{\varepsilon}{10}\cdot\sum_{k=1}^{n}\boldsymbol{\pi}_j(k)=\frac{\varepsilon}{10},\end{aligned} \tag{21}$$

where S is the set of nodes without sampling $\sqrt{c}$-walks starting from them.

□

Based on theorem and lemmas above, we obtain Theorem 2.

Theorem 2. *For any source node $v_i \in V$, the basic ClipSim algorithm provides single-source SimRank vector $\hat{\mathbf{s}}^L = \hat{\mathbf{S}}_L \cdot \boldsymbol{e}_i$, such that for any node $v_j \in V$,*

$$\left|\hat{\mathbf{s}}^L(j) - s(v_i, v_j)\right| > \varepsilon$$

with probability less than $1/n$.

4 PARALLELIZATION FOR CLIPSIM ON GPU

4.1 Parallel Algorithm Overview

The parallelization of our proposed ClipSim algorithm mainly consists of two parts: 1) matrix and vector operations for the calculation of PPR and SimRank vectors; 2) random walks simulation for the estimation of correction matrix $\boldsymbol{D}$. To this, we design the parallel version of ClipSim algorithm on GPU, called *ClipSim-GPU.*

Here, we introduce two functions for the calculation of PPR vector $\boldsymbol{\pi}_i$ and SimRank vector $\mathbf{s}^L$.

- $\boldsymbol{y} = \textsc{SpMxV}(\alpha, \boldsymbol{A}, \boldsymbol{x})$. The function is for the multiplication of sparse matrix $\boldsymbol{A}$ and dense vector $\boldsymbol{x}$ with scalar α
$$\boldsymbol{y} = \alpha \cdot \boldsymbol{A}\boldsymbol{x}.$$
- $\boldsymbol{z} = \textsc{VecAdd}(\boldsymbol{x}, \boldsymbol{y})$. This function adds two dense vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and the result is a dense vector $\boldsymbol{z}$, i.e.,
$$\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y}.$$

The two functions can be easily and efficiently operated in parallel with Nvidia cuSPARSE [20] and cuBLAS [28] libraries. The remaining part is the parallelization of Algorithm 3, sampling method for estimating $\boldsymbol{D}(k, k)$. Thus, in the remainder of this section, we emphasize on the optimization for the GPU-based estimation of the correction matrix $\boldsymbol{D}$. The pseudo code of ClipSim-GPU is summarized in Algorithm 4. Specifically, line 9-21 is mainly for DCSR data structure introduced in Subsection 4.2, line 22-39 is mainly for double-seed technique introduced in Subsection 4.4.

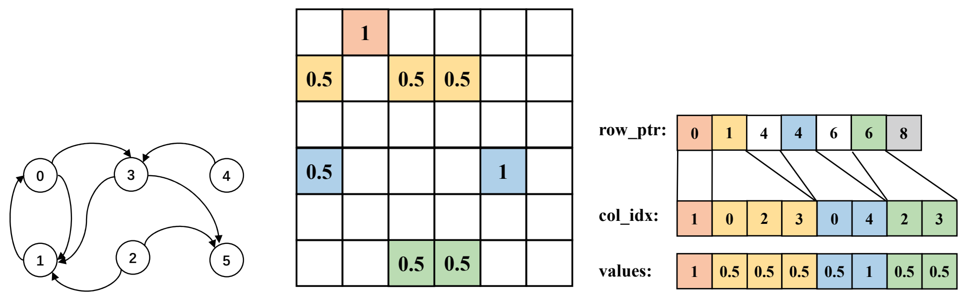
Algorithm 4 ClipSim-GPU Algorithm.

```
Input: thread id $tid$, number of thread $T$, transition matrix $P$, source node $v_i$, , maximum error $\varepsilon$
1: $L \leftarrow \left\lceil \log_{\frac{1}{c}} \frac{10}{\varepsilon} \right\rceil$.
2: $\boldsymbol{\pi}_i^0 \leftarrow (1-\sqrt{c}) \cdot \boldsymbol{e}_i,\ \boldsymbol{\pi}_i \leftarrow (1-\sqrt{c}) \cdot \boldsymbol{e}_i$.
3: for $\ell$ from 1 to $L$ do
4:     $\boldsymbol{\pi}_i^\ell \leftarrow \textsc{SpMxV}\left(\sqrt{c}, \boldsymbol{P}, \boldsymbol{\pi}_i^{\ell-1}\right)$.
5:     $\boldsymbol{\pi}_i \leftarrow \textsc{VecAdd}\left(\boldsymbol{\pi}_i, \boldsymbol{\pi}_i^\ell\right)$.
6: end for
7: $R \leftarrow \frac{75 \cdot \|\boldsymbol{\pi}_i\|_2 \cdot \log n}{8(1-\sqrt{c})^4 \varepsilon^2}$.
8: Call Algorithm 2 with PPR vector $\boldsymbol{\pi}_i$ in parallel to obtain each $R_k$ $(k = 1, \cdots, n)$.
9: $rowPtr, colIdx, values \leftarrow \boldsymbol{P}$.
10: $numLoop \leftarrow \lceil n/T \rceil$.
11: $rowIdx, emptyIdx, validMask \leftarrow n$-length arrays filled with zeroes.
12: for $p$ from 1 to $numLoop$ do
13:     $k \leftarrow p \cdot T + tid$.
14:     if then$R(k) > 0$
15:         $validMask[k] = 1$
16:     end if
17: end for
18: $rowIdx \leftarrow \textsc{PrefixScan}(validMask)$.
```

```
19: $lenRowIdx \leftarrow$ Reduce($validMask$).
20: $validMask \leftarrow \sim validMask$.
21: $emptyIdx \leftarrow$ PrefixScan($validMask$).
22: $numRWLoop \leftarrow \lceil lenRowIdx/T \rceil$.
23: $numEmptyLoop \leftarrow \lceil (n - lenRowIdx)/T \rceil$.
24: for $p$ from 1 to $numRWLoop$ do
25:     $k \leftarrow p \cdot T + rowIdx[tid]$.
26:     $wid \leftarrow tid/32$.
27:     $wSeed \leftarrow$ seedInit($wid$).
28:     $tSeed \leftarrow$ seedInit($tid$).
29:     for $x$ from 1 to $R(k)$ do
30:         while rand($wSeed$) $< c$ do
31:             $v_k^{(1)} \leftarrow rowPtr\,[colIdx[k] + \textbf{randint}(tSeed)\%|\mathcal{I}(v_k)|]$.
32:             $v_k^{(2)} \leftarrow rowPtr\,[colIdx[k] + \textbf{randint}(tSeed)\%|\mathcal{I}(v_k)|]$.
33:             if $v_k^{(1)} == v_k^{(2)}$ then
34:                 $q(k) \leftarrow q(k) + 1$.
35:             end if
36:         end while
37:     end for
38:     $D(k,k) \leftarrow 1 - \frac{q(k)}{R(k)}$.
39: end for
40: for $p$ from 1 to $numEmptyLoop$ do
41:     $k \leftarrow p \cdot T + emptyIdx[tid]$.
42:     $D(k,k) \leftarrow 1 - \frac{c}{|\mathcal{I}(v_k)|}$.
43: end for
44: $s^0 \leftarrow$ SpMxV$\left(\frac{1}{1-\sqrt{c}}, \hat{D}, \pi_i^L\right)$.
45: for $\ell$ from 1 to $L$ do
46:     $u^1 \leftarrow$ SpMxV$\left(\sqrt{c}, P^T, s^{\ell-1}\right)$.
47:     $u^2 \leftarrow$ SpMxV$\left(\frac{1}{1-\sqrt{c}}, \hat{D}, s^{L-\ell}\right)$.
48:     $s^\ell \leftarrow$ VecAdd$(u^1, u^2)$.
49: end for
Output: Single-source SimRank vector $s^L$.
```

4.2 DCSR-based Random Walk

Due to the large scale of real-world graphs (e.g., 10^7 nodes in a graph), the transition matrix $\boldsymbol{P}$ can only be stored as a compressed sparse matrix. Usually, compressed sparse row (CSR) format is used to store $\boldsymbol{P}$, which is widely applied in sparse matrix-vector multiplication [7]. In Fig.2, 2(a) illustrates a toy graph, the transition matrix $\boldsymbol{P}$ and the corresponding CSR format are presented in 2(b) and 2(c), respectively. Considering that the GPU storage (namely, graphics memory) is limited, we tailor the CSR format of $\boldsymbol{P}$ to simulate random walks. We convert the CSR format of $\boldsymbol{P}$ to DCSR (Doubly Compressed Sparse Row) format [3], which can provide each neighboring node with accessing array elements three times. Specifically, DCSR format is a variant of CSR. We "compress" the nonzero elements in "row_ptr" array in CSR (see Fig.2) and construct two smaller arrays named "row_idx" and "row_ptr" in DCSR, which are the nonzero indices and values, respectively. With this data structure, the nodes with zero indegree are excluded in advance, which avoids the waste of threads initially allocated for these nodes and non-contiguous memory access in the first steps of parallel random walks. The DCSR format of matrix $\boldsymbol{P}$ is illustrated in Fig.3.



(a) toy graph. (b) transition matrix $\boldsymbol{P}$. (c) CSR format of $\boldsymbol{P}$.

Fig. 2. CSR format and adjacency list.

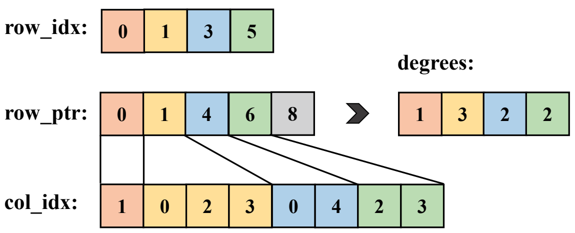


Fig. 3. The DCSR format of matrix $\boldsymbol{P}$.

4.3 Load Balancing and Task Stealing

4.3.1 Initial Threads Allocation. Considering the SIMT model in GPU parallel computing, we propose *thread-node mapping* strategy. In this strategy, the $\sqrt{c}$-walks from the same node are simulated with the same thread. Initially, for node v_k, the corresponding thread

$$tid = k \bmod T,$$

where T is the number of threads. However, the number of samples starting from each node v_k is calculated according to the PPR vector $\boldsymbol{\pi}_i$. In large scale-free graphs, the PageRank values follow power-law distribution. Thus, our thread-node mapping strategy may cause severe load imbalance, which means a few threads take a huge number of $\sqrt{c}$-walks. As a result, we introduce a popular method called *task stealing* in Subsubsection 4.3.2.

4.3.2 Task Stealing. Fig. 4 is an overview of our load balancing strategy. Thread 0 and thread 1 simulate $\sqrt{c}$-walks starting from node 0,2,4 and 1,3,5, respectively. When thread 0 has already simulated all the $\sqrt{c}$-walks from node 0,2,4, it simulates $\sqrt{c}$-walks from other nodes.

To simplify the explanation of task stealing, we introduce two concepts in advance.

- **Idle Thread.** If a thread has already simulated all the random walks allocated according to the thread-node mapping strategy, then we say this thread is *idle*.
- **Busy Thread.** Conversely, if a thread is not idle, then we say the thread is *busy*. According to the thread-node mapping strategy, the more random walks corresponding to the thread that are left unfinished, the busier the thread is.

The basic idea of task stealing is to let idle threads "help" busy threads simulate random walks. The underlying cause of load imbalance is the imbalance of the personalized PageRank values in scale-free graphs. Thus, we need more threads to simulate the $\sqrt{c}$-walks from v_k with large PPR

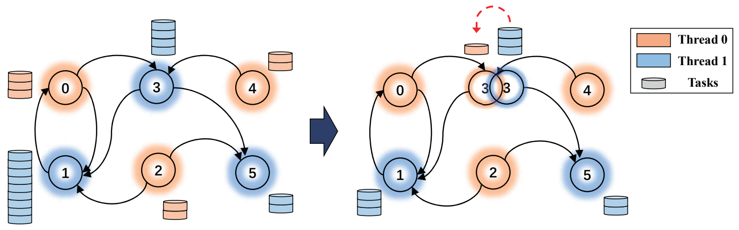


Fig. 4. Illustration of thread-node mapping and task stealing.

value $\boldsymbol{\pi}_i(k)$. The major issue is, for an idle thread, how to determine which threads urgently need help. In other words, which node does the idle thread simulate the random walks from? Here, we provide two different types of task stealing.

- **Random Jumping.** Given that the number of threads is T, for an idle thread, it generate a random integer r with range from 0 to $T-1$ uniformly. Then, this idle thread helps the thread t_r and simulates random walks from v_k where $t_r = k \ mod \, T$.
- **Fixed Stride Jumping.** Suppose the stride step is s. The idle thread t_i helps the thread t_b and simulates random walks from v_k, where $t_b = (t_i + s) \ mod \, T$ and $t_b = k \ mod \, T$.

4.4 Warp-centric Parallelism

Although thread-node mapping and task stealing seem to be sufficient for load balancing, *warp divergence* still affects efficiency. As is described in Subsection 2.4, SIMT model can efficiently handle huge amount of data with a single instruction and different threads. However, when executing an algorithm in parallel on GPU (e.g., random walk [24], graph sampling [21]), the instructions for the threads in the same warp must be the same. Specifically, when executing an if-else clause in the same warp, if some threads execute *if-clause* and other threads execute *else-clause*, if-clause and else-clause cannot be executed simultaneously, which costs more time. Moreover, if we use a *switch-case* clause, the situation can be even worse. To avoid warp divergence, we propose a warp-centric parallel algorithm for the estimation of matrix $\boldsymbol{D}$. The algorithm mainly consists of three parts.

- **Warp-node mapping.** Based on thread-node mapping, we propose *warp-node mapping* strategy, which means $\sqrt{c}$-walks from the same node are simulated with the same warp. Suppose wid is the warp index of the thread tid, T and W are the number of threads and warps, respectively. For node v_k, we have $W = \lceil T/32 \rceil$, $wid = \lfloor tid/32 \rfloor$, $wid = k \bmod W$.
- **Double random seeds.** When simulating a $\sqrt{c}$-walk, there are two places using randomness. One is to determine whether the $\sqrt{c}$-walk stops or not, which also determine the number of while-loop iterations, the other is to determine which node the $\sqrt{c}$-walk arrives at the next step. Note that for the threads in the same warp, if the numbers of while-loop iterations are different, execution time largely increases due to warp divergence. Experimentally, the execution time increases linearly with the increase of divergent threads [2]. Thus, we use the same seed r_{wid} for the threads in the same warp to maintain the same number of while-loop iterations, and use different seeds r_{tid} for each thread tid to determine the nodes arrived at the next step. The double-random-seed technique is depicted in Fig.5.
- **Warp-centric jumping.** Similar to the task stealing we mentioned above, let idle warps "help" busy warps simulate random walks, which also derives two types: random jumping and fixed stride jumping.

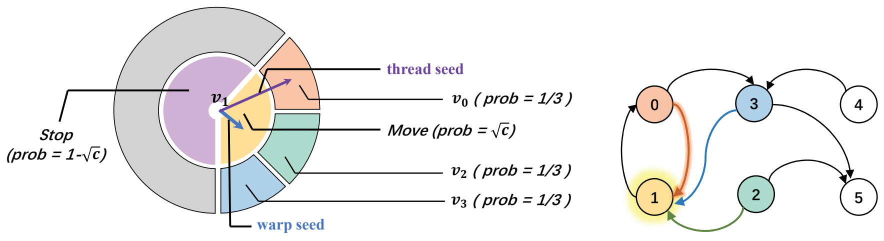


Fig. 5. The illustration of double-random-seed technique.

The hardware architecture of GPU (e.g., SIMD parallelization) may bring about dependency factors. We handle this potential challenge both theoretically and experimentally. Let h be the number of threads in a warp, R be the number of $\sqrt{c}$-walks we obtained in Theorem 1. For the double-seed technique, we choose one thread from each warp, and the $\sqrt{c}$-walks simulated with these threads are independent. Thus, the accuracy still has a theoretical guarantee when the number of $\sqrt{c}$-walks is set to $h \cdot R$. The time complexity for simulating random walks becomes $h \cdot R$, where h is a constant that depends on the hardware architecture of GPU. Moreover, based on the Monte Carlo method, the $\sqrt{c}$-walks simulated with the other $h-1$ threads in each warp refine our results and further reduce the error. Experimentally, we found that setting the number of $\sqrt{c}$-walks to R is sufficient to achieve the same accuracy as using $h \cdot R$. Note that in the experiments, using R random walks is a practical heuristic that is weaker than the theoretical guarantee. The effect of the double-seed technique on the precision of the algorithm will be discussed in the ablation study in Subsection 5.3.

4.5 Atomic Operations Reduction

When more than one threads simulate random walks from the same node, to count the number of meeting pairs of $\sqrt{c}$-walks, atomic operations are indispensable. However, atomic operations from different threads are essentially executed in series. To boost the efficiency of our algorithm, we use atomic operations as less as possible. Our idea is simple and explicit. We partition the $\sqrt{c}$-walks from the same node v_k into several portions, each thread takes away a portion and simulates $\sqrt{c}$-walks. Suppose the number of samples from v_k is $R(k)$ and there are $T(k)$ threads simulating these samples. Obviously, each thread needs at most $R(k)$ atomic operations. Let $P(k) = \lceil R(k)/T(k) \rceil$. Assume that the execution times of each atomic operation and each pair of $\sqrt{c}$-walks are a and b, respectively. If we partition the $R(k)$ samples into $P(k)$ portions, and every time each thread takes away a portion, then the computational time is

$$a \cdot T(k) + b \cdot P(k),$$

which reaches the minimum with $T(k) = \left\lceil \sqrt{\frac{b \cdot R(k)}{a}} \right\rceil$.

5 EXPERIMENTS SETTING

In this section, we evaluate ClipSim in comparison with existing single-machine and single-source SimRank algorithms on large graphs, including ExactSim, the current state-of-the-art algorithm with accuracy guarantee. In addition, we conduct an ablation study to show the effectiveness of our optimizations.

The implementations of our proposed ClipSim on GPU and CPU are denoted as *ClipSim-GPU* and *ClipSim-CPU*, respectively. We evaluate our proposed ClipSim by comparing it with four single-source and single-machine algorithms, i.e., ExactSim [30] (state-of-the-art), SLING [26], ProbeSim [16] and PRSim [34].

Table 2. Datasets.

Dataset	n	m	Type
graph500-scale22	2,393,285	128,194,008	directed
graph500-scale23	4,606,314	258,501,410	directed
graph500-scale24	8,860,450	520,523,686	directed
graph500-scale25	17,043,780	1,046,934,896	directed
DBLP-Author (DB)	5,425,963	17,298,032	undirected
IndoChina (IC)	7,414,768	191,606,827	directed
IT-2004 (IT)	41,291,594	1,135,718,909	directed
Twitter (TW)	41,652,230	1,468,364,884	directed

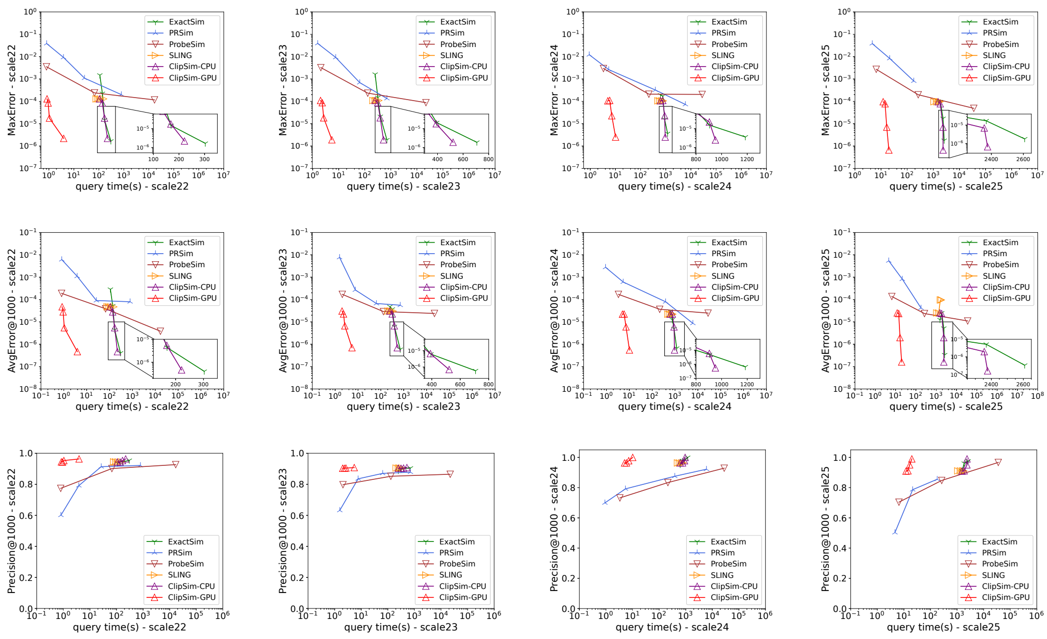


Fig. 6. Comparison on synthetic graphs.

5.1 Dataset and Parameters

Dataset and Platform. Referring to [23, 30], we use the same four real-world datasets. Moreover, we also use four synthetic datasets [22] as summarized in Table 2.

ClipSim-GPU is implemented with CUDA framework and compiled with CUDA Toolkit 11.2 and g++ 7.5.0 with optimization flag -O3. All the experiments are conducted on a platform with 128G RAM, one Nvidia Tesla V100-PCIe and one Intel(R) Xeon(R) Gold 5117 CPU @2.00GHz with 14 cores. For the random number generation, we use the cuRAND library [27].

Baselines. In all experiments, we set the decay factor c as 0.8. The parameter settings of the five algorithms are given below.

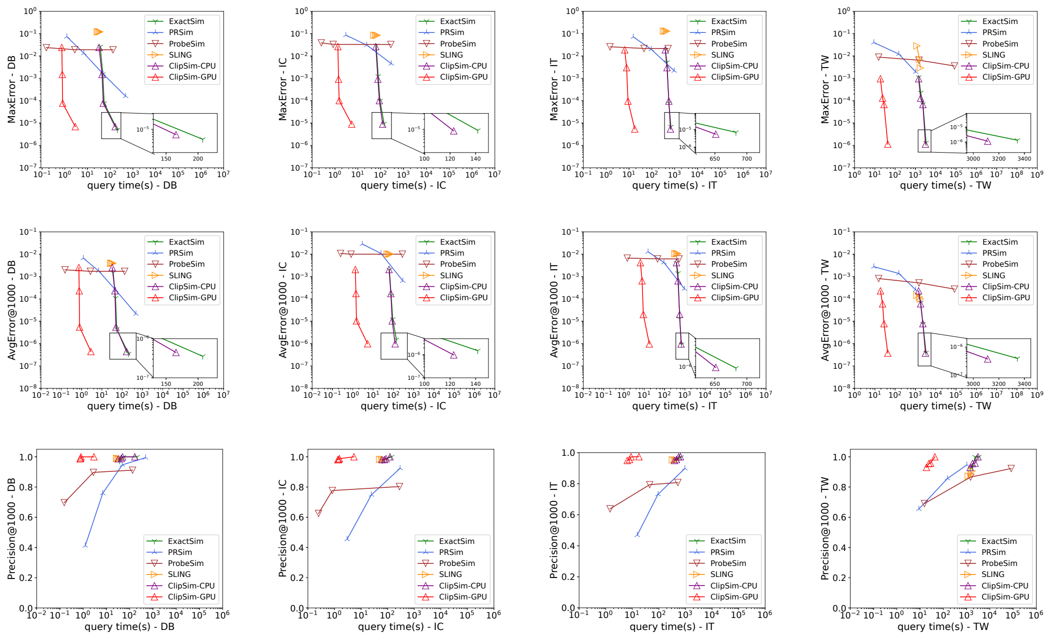


Fig. 7. Comparison on real-world graphs.

- **ExactSim [30]** is the state-of-the-art single-machine and single-source SimRank algorithm with accuracy guarantee on large graphs, which can obtain the exact single-source SimRank vectors. To compare with other algorithms, we vary ε from 10^{-3} to 10^{-6}.
- **SLING [26].** SLING has the same error parameter ε as ExactSim, which varies from 10^{-3} to 10^{-6} .
- **PRSim [34] and ProbeSim [16].** PRSim and ProbeSim have the same error parameter ε. Due to the complexity of the two algorithms, when the parameter is set to 10^{-4}, the preprocessing time exceeds 24 hours. Thus, we only vary parameter ε from 10^{-1} to 10^{-3}.
- **ClipSim-GPU and ClipSim-CPU.** We set the error parameter ε ranging from 10^{-3} to 10^{-6} for our proposed ClipSim-GPU and ClipSim-CPU. For parallel computing on GPU, we use 512 blocks with 512 threads per block.

Ground Truth. Since that the float type in IEEE 754 usually supports precision of up to 6 or 7 decimal places, and ExactSim provides accuracy bound, we set the absolute error parameter $\varepsilon = 10^{-7}$ for ExactSim and regard the obtained single-source SimRank vector as the ground truth.

Evaluation Metrics. In consistency of the previous works [30, 34], we use the same three metrics to evaluate the accuracy of single-source SimRank vectors. The introduction of the three metrics is presented below.

- **MaxError.** Given a source node v_i and an approximate single-source SimRank similarities vector $\hat{\mathbf{s}}_i$, *MaxError* is the maximum error over n similarities, i.e.,

$$MaxError = \max_{j=1,\cdots,n} \left|\hat{\mathbf{s}}_i(j) - s\left(v_i, v_j\right)\right|.$$

- **AvgError@1000.** The definitions of v_i and vector $\hat{\mathbf{s}}_i$ are the same as above. *AvgError@1000* is the average error over top-1000 similarities, i.e.,

$$AvgError@1000 = \frac{1}{1000}\sum_{j=1}^{1000}\left|\hat{\mathbf{s}}_i\left(\Pi(j)\right) - s\left(v_i, v_{\Pi(j)}\right)\right|,$$

where $\Pi(\cdot)$ is a permutation obtained by sorting single-source SimRank similarities w.r.t. source node v_i in a descending order.
- **Precision@1000.** Given a source node v_i and an approximate top-1000 similarities node set $V = \{v_1, \cdots, v_{1000}\}$, *Precision@1000* is the percentage of nodes in V that coincides with the exact top-1000 results.

On each dataset, *MaxError*, *AvgError@1000* and *Precision@1000* results are averaged over 50 queries with different source nodes.

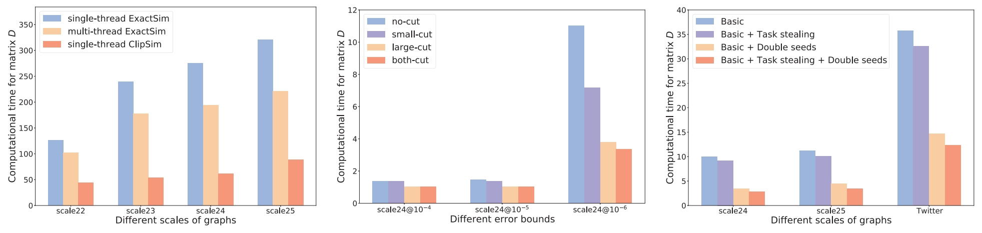
5.2 Performance Results

5.2.1 Comparison with baselines. In this subsection, we evaluate our proposed ClipSim on eight graphs by comparing with baselines mentioned above. Fig.6 and Fig.7 show the query time and accuracy trade-off on graph500 networks and real-world graphs.

- The horizontal axis in each subfigure is the query time, which contains the computational time of SimRank vector without taking preprocessing time [2] into account.
- The vertical axis in the three rows of both Fig.6 and Fig.7 are the three metrics mentioned above. For MaxError and AvgError@1000, the lower the values are, the better the algorithms perform. While for Precision@1000, the higher values are better.

Since all the baselines only have serial CPU implementation, for the fairness of comparison, we also implement a serial ClipSim on CPU (i.e., ClipSim-CPU in each subfigure). To better visualize the result, the performances of ClipSim-CPU and ExactSim are enlarged at the bottom right of each subfigure. In Fig.6 and Fig.7, it may seem that the performance improvement of ClipSim is not very pronounced. This is because although our ClipSim sharply reduces the computation time of matrix $\boldsymbol{D}$, as shown in Fig.8(a), when the accuracy parameter $\varepsilon = 10^{-6}$ or larger, the main overhead of SimRank computation with serial CPU algorithms lies in the sparse matrix and vector multiplication, which overshadows the computation time of matrix $\boldsymbol{D}$. For instance, with $\varepsilon = 10^{-6}$ and graph500-scale25, the query time of ClipSim-CPU and ExactSim is 2376.65 s and 2608.55 s, while corresponding computational time of $\boldsymbol{D}$ is 88.55 s and 320.45 s, respectively. The comparison of computational time of matrix $\boldsymbol{D}$ is presented in detail in the next subsection (see Fig.8(a)). Here, we add more discussion on the bottleneck of the SimRank computation. If we parallelize matrix-vector multiplication on GPU and calculate matrix $\boldsymbol{D}$ with serial CPU algorithm, for graph500-scale25, $\varepsilon = 10^{-6}$, the first part on GPU takes 15.65 s and the second part on CPU takes 320.45 s. We can clearly see that in this case the bottleneck lies in the computation of matrix $\boldsymbol{D}$. Moreover, when ε is set to 10^{-7}, simulating random walks with ClipSim on GPU becomes the major part of the overhead of SimRank computation. With $\varepsilon = 10^{-7}$, the total computational time of ExactSim is 26696.4 s, and the time for estimating matrix $\boldsymbol{D}$ and matrix-vector multiplication is 23848.9 s and 2847.5 s on graph500-scale25 dataset, respectively. With the same experiment settings, using ClipSim-GPU, computational time for matrix $\boldsymbol{D}$ estimation and matrix-vector multiplication is 273.15 s and 21.45 s, respectively. Therefore, a more accurate calculation of single-source SimRank must tackle the overhead bottleneck of estimating matrix $\boldsymbol{D}$.

[2] Note that PRSim and ProbeSim need preprocessing to construct indices, while other algorithms can be operated directly without preprocessing.



(a) Computation time of ExactSim and ClipSim on CPU.

(b) Four types of ClipSim.

(c) Basic ClipSim and optimized ClipSim.

Fig. 8. Scalability and ablation study.

- **Accuracy.** According to Fig.6 and Fig.7, we can see that both MaxError and AvgError@1000 of the single-source SimRank vectors obtained by the proposed ClipSim and ExactSim are approximately the same, and are much smaller than PRSim, ProbeSim and SLING. Besides, according to subfigures in the third rows of Fig.6 and Fig.7, ClipSim can achieve more than 90% precision even with parameter $\varepsilon = 10^{-3}$, and nearly 100% with $\varepsilon = 10^{-6}$.
- **Efficiency.** In terms of efficiency, the query time of the proposed ClipSim-GPU is up to $120 - 160\times$, $30 - 80\times$ faster than ExactSim with parameter ε ranging from 10^{-3} to 10^{-6} on graph500 scale22-25 networks and four real-world graphs, respectively. Especially, for large-scale graphs, such as Twitter, ExactSim needs more than $3,000$ seconds to obtain a SimRank vector with MaxError smaller than 10^{-6}, while ClipSim only needs about 40 seconds to obtain the solution with the same accuracy.

5.2.2 Scalability of ClipSim. To validate the scalability of ClipSim, we compare the single-thread ExactSim, ExactSim parallelized with OpenMP [4] (8 threads) and single-thread ClipSim on CPU. In terms of computational time, we can see single-thread ClipSim is 4× and 3× faster than single-thread ExactSim and OpenMP parallelized ExactSim, respectively.

5.2.3 Performance-price ratio. ClipSim-GPU shows a high performance price ratio. The price of Nvidia Tesla V100-PCIe GPU is 9× than Gold 5117 CPU @2.00GHz. Referring to [17], we define the performance-price ratio as: $1/time/price$. With up to 160× speedup, the performance-price ratio of ClipSim-GPU shows up to 18× better cost-efficiency compared with ExactSim on CPU.

5.2.4 Performance-power ratio. ClipSim-GPU also shows a high performance power ratio compared to CPU. The thermal design powers (TDP) of the V100 and Gold 5117 CPU are $250W$ and $105W$, respectively. With up to 160× speedup on GPU, ClipSim-GPU shows up to 67× better energy efficiency than the compared CPU-based algorithms.

5.3 Ablation Study

5.3.1 Large cut and Small Cut. We first evaluate the computational time of matrix $\boldsymbol{D}$ under four different conditions. As is described in Section 3, ClipSim consists of two parts, LargeCut and SmallCut. Recall that LargeCut means truncating a portion of $\sqrt{c}$-walks starting from nodes with large personalized PageRank (PPR), and SmallCut means truncating all the $\sqrt{c}$-walks from nodes with small PPR values. We denote the algorithm with both LargeCut and SmallCut as both-cut and the algorithm without CutEnds strategy as no-cut. Fig.8(b) shows the computational time of $\boldsymbol{D}$ with parameter $\varepsilon = 10^{-4} - 10^{-6}$. We can see that with $\varepsilon = 10^{-6}$, both-cut is 3×, 2×, 1.2× faster than no-cut, small-cut and large-cut, respectively.

5.3.2 GPU-based matrix $\boldsymbol{D}$ estimation. We evaluate the efficiency of the two GPU-based optimizations, i.e., the task stealing and warp-centric double random seeds, which are denoted as *Task stealing* and *Double seeds* in Fig.8(c). As is presented in Fig.8(c), the two GPU-based optimizations brings a $3\times-3.5\times$ speedup on all the three graphs. In addition, in terms of accuracy, on the graph500-scale22 dataset, with $\varepsilon = 10^{-6}$, the MaxError accuracy of ClipSim with and without the double-seed technique is 2.678×10^{-6} and 2.518×10^{-6}, while the time for estimating matrix $\boldsymbol{D}$ is 1.87 s and 7.90 s, respectively. On the Twitter dataset, with the same settings, the MaxErrors are 1.332×10^{-6} and 1.143×10^{-6}, while time for estimating $\boldsymbol{D}$ is 12.31 s and 35.75 s. The experimental results demonstrate that setting the number of $\sqrt{c}$-walks to R with the double-seed technique is sufficient to achieve the same accuracy as the basic ClipSim without the double-seed technique.

6 CONCLUSION

In this paper, we proposed ClipSim, a novel algorithm for single-source SimRank computation. In the traditional linearization method, there is a diagonal matrix $\boldsymbol{D}$ needing to be estimated with a huge number of random walks. Inspired by the observation in experiments and law of large numbers, we truncate a large number of random walks and hence accelerate the computation of single-source SimRank without loss of accuracy. Furthermore, we give a theoretical analysis on our pruning rules, which guarantees the error bound of the single-source SimRank. Moreover, with a careful modification, we parallelize our proposed ClipSim on the GPU platform and achieve a significant efficiency enhancement. Extensive simulations show that compared with the state-of-the-art algorithm ExactSim, GPU-based ClipSim achieves a comparable accuracy with two orders of magnitude lower computational time on both the common benchmark graphs and real-world graphs.

ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China Grant Nos. 12025104, 81930119.

REFERENCES

[1] Ioannis Antonellis, Hector Garcia-Molina, and Chi-Chao Chang. 2008. SimRank++: Query rewriting through link analysis of the clickgraph (poster). In *Proceedings of the 17th International Conference on World Wide Web*. Beijing, China, 1177–1178.

[2] Piotr Bialas and Adam Strzelecki. 2015. Benchmarking the cost of thread divergence in CUDA. In *International Conference on Parallel Processing and Applied Mathematics*. Springer, Krakow, Poland, 570–579.

[3] Aydin Buluc and John R Gilbert. 2008. On the representation and multiplication of hypersparse matrices. In *Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, Miami, Florida, USA, 1–11.

[4] Rohit Chandra, Leo Dagum, David Kohr, Ramesh Menon, Dror Maydan, and Jeff McDonald. 2001. *Parallel programming in OpenMP.* Morgan kaufmann.

[5] Nasrollah Etemadi. 1981. An elementary proof of the strong law of large numbers. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* 55, 1 (1981), 119–122.

[6] Dániel Fogaras and Balázs Rácz. 2005. Scaling link-based similarity search. In *Proceedings of the 14th International Conference on World Wide Web*. Chiba, Japan, 641–650.

[7] Joseph L Greathouse and Mayank Daga. 2014. Efficient sparse matrix-vector multiplication on GPUs using the CSR storage format. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, New Orleans, Louisiana, USA, 769–780.

[8] Taher H Haveliwala. 2003. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering* 15, 4 (2003), 784–796.

[9] W. Hoeffding. 1963. Probability inequalities for sums of bounded random variables. In *J. Amer. Statist. Assoc.* 13–30.

[10] William George Horner. 1819. XXI. A new method of solving numerical equations of all orders, by continuous approximation. *Philosophical Transactions of the Royal Society of London* 109 (1819), 308–335.

[11] Ankush Jain, Surendra Nagar, Pramod Kumar Singh, and Joydip Dhar. 2020. EMUCF: Enhanced multistage user-based collaborative filtering through non-linear similarity for recommendation systems. *Expert Systems with Applications* 161 (2020), 113724.

[12] Glen Jeh and Jennifer Widom. 2002. SimRank: A measure of structural-context similarity. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Edmonton Alberta, Canada, 538–543.

[13] Ruoming Jin, Victor E Lee, and Hui Hong. 2011. Axiomatic ranking of network role similarity. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Diego, California, USA, 922–930.

[14] Mitsuru Kusumoto, Takanori Maehara, and Ken-ichi Kawarabayashi. 2014. Scalable similarity search for SimRank. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. Snowbird, Utah, USA, 325–336.

[15] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a social network or a news media?. In *Proceedings of the 19th International Conference on World Wide Web*. Raleigh, North Carolina, USA, 591–600.

[16] Yu Liu, Bolong Zheng, Xiaodong He, Zhewei Wei, Xiaokui Xiao, Kai Zheng, and Jiaheng Lu. 2017. ProbeSim: scalable single-source and top-k simrank computations on dynamic graphs. *Proceedings of the VLDB Endowment* 11, 1 (2017), 14–26.

[17] Shengliang Lu, Bingsheng He, Yuchen Li, and Hao Fu. 2020. Accelerating exact constrained shortest paths on GPUs. *Proceedings of the VLDB Endowment* 14, 4 (2020), 547–559.

[18] Takanori Maehara, Mitsuru Kusumoto, and Ken-ichi Kawarabayashi. 2014. Efficient SimRank computation via linearization. *arXiv preprint arXiv:1411.7228* (2014).

[19] Sara Makki, Rafiqul Haque, Yehia Taher, Zainab Assaghir, Mohand-Said Hacid, and Hassan Zeineddine. 2018. A cost-sensitive cosine similarity K-nearest neighbor for credit card fraud detection. In *Big Data and Cyber-Security Intelligence*. Hadath, Lebanon, 1–6.

[20] Maxim Naumov, L Chien, Philippe Vandermersch, and Ujval Kapasi. 2010. Cusparse library. In *GPU Technology Conference*. San Jose, California, USA, 1–96.

[21] Santosh Pandey, Lingda Li, Adolfy Hoisie, Xiaoye S Li, and Hang Liu. 2020. C-SAW: A framework for graph sampling and random walk on GPUs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Atlanta, Georgia, USA, 1–15.

[22] Ryan A Rossi and Nesreen K Ahmed. 2015. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. Austin Texas, USA, 4292–4293.

[23] Jieming Shi, Tianyuan Jin, Renchi Yang, Xiaokui Xiao, and Yin Yang. 2020. Realtime index-free single source SimRank processing on web-scale graphs. *Proceedings of the VLDB Endowment* 13, 7 (2020), 966–980.

[24] Jieming Shi, Renchi Yang, Tianyuan Jin, Xiaokui Xiao, and Yin Yang. 2019. Realtime top-k personalized pagerank over large graphs on GPUs. *Proceedings of the VLDB Endowment* 13, 1 (2019), 15–28.

[25] Julian Shun and Guy E Blelloch. 2013. Ligra: a lightweight graph processing framework for shared memory. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. Shenzhen, China, 135–146.

[26] Boyu Tian and Xiaokui Xiao. 2016. SLING: A near-optimal index structure for SimRank. In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*. San Francisco, California, USA, 1859–1874.

[27] Xiang Tian and Khaled Benkrid. 2009. Mersenne twister random number generation on FPGA, CPU and GPU. In *NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE, San Francisco, California, USA, 460–464.

[28] Darya Gennadievna Vorotnikova and Dmitriy L Golovashkin. 2014. CUBLAS-aided long vector algorithms. *Journal of Mathematical Modelling and Algorithms in Operations Research* 13, 4 (2014), 425–431.

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

[30] Hanzhi Wang, Zhewei Wei, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Exact single-source SimRank computation on large graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Portland, Oregon, USA, 653–663.

[31] Yue Wang, Yulin Che, Xiang Lian, Lei Chen, and Qiong Luo. 2020. Fast and accurate simrank computation via forward local push and its parallelization. *IEEE Transactions on Knowledge and Data Engineering* 33, 12 (2020), 3686–3700.

[32] Yangzihao Wang, Andrew Davidson, Yuechao Pan, Yuduo Wu, Andy Riffel, and John D Owens. 2016. Gunrock: A high-performance graph processing library on the GPU. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 1–12.

[33] Yue Wang, Ruiqi Xu, Zonghao Feng, Yulin Che, Lei Chen, Qiong Luo, and Rui Mao. 2020. DISK: a distributed framework for single-source simrank with accuracy guarantee. *Proceedings of the VLDB Endowment* 14, 3 (2020), 351–363.

[34] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Yu Liu, Xiaoyong Du, and Ji-Rong Wen. 2019. PRSim: Sublinear time simrank computation on large power-law graphs. In *Proceedings of the 2019 ACM SIGMOD International Conference*

on Management of Data. Amsterdam, Netherlands, 1042–1059.

[35] Weiren Yu, Sima Iranmanesh, Aparajita Haldar, Maoyin Zhang, and Hakan Ferhatosmanoglu. 2021. RoleSim*: Scaling axiomatic role-based similarity ranking on large graphs. *World Wide Web* (2021), 1–45.

[36] Weiren Yu, Xuemin Lin, Wenjie Zhang, Jian Pei, and Julie A McCann. 2019. SimRank*: Effective and scalable pairwise similarity search based on graph topology. *The VLDB Journal* 28, 3 (2019), 401–426.

[37] Weiren Yu and Julie A McCann. 2015. Efficient partial-pairs SimRank search on large networks. *Proceedings of the VLDB Endowment* 8, 5 (2015), 569–580.

[38] Fan Zhang, Xuemin Lin, Ying Zhang, Lu Qin, and Wenjie Zhang. 2019. Efficient community discovery with user engagement and similarity. *The VLDB Journal* 28, 6 (2019), 987–1012.

[39] Jing Zhang, Jie Tang, Cong Ma, Hanghang Tong, Yu Jing, and Juanzi Li. 2015. Panther: Fast top-k similarity search on large networks. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. Sydney, Australia, 1445–1454.

Received April 2022; revised July 2022; accepted August 2022