The VLDB Journal (2021) 30:989–1015
https://doi.org/10.1007/s00778-021-00672-7

REGULAR PAPER

ExactSim: benchmarking single-source SimRank algorithms with high-precision ground truths

Hanzhi Wang[1] · Zhewei Wei[2] · Yu Liu[3] · Ye Yuan[4] · Xiaoyong Du[5] · Ji-Rong Wen[6]

Received: 4 August 2020 / Revised: 23 February 2021 / Accepted: 17 April 2021 / Published online: 5 June 2021

Abstract

SimRank is a popular measurement for evaluating the node-to-node similarities based on the graph topology. In recent years, single-source and top-*k* SimRank queries have received increasing attention due to their applications in web mining, social network analysis, and spam detection. However, a fundamental obstacle in studying SimRank has been the lack of ground truths. The only exact algorithm, Power Method, is computationally infeasible on graphs with more than 10^6 nodes. Consequently, no existing work has evaluated the actual accuracy of various single-source and top-*k* SimRank algorithms on large real-world graphs. In this paper, we present ExactSim, the first algorithm that computes the exact single-source and top-*k* SimRank results on large graphs. This algorithm produces ground truths with precision up to 7 decimal places with high probability. With the ground truths computed by ExactSim, we present the first experimental study of the accuracy/cost trade-offs of existing approximate SimRank algorithms on large real-world graphs and synthetic graphs. Finally, we use the ground truths to exploit various properties of SimRank distributions on large graphs.

Keywords SimRank · Single-source · Exact computation · Ground truths · Power-law · Benchmarks

✉ Zhewei Wei
zhewei@ruc.edu.cn

Hanzhi Wang
hanzhi_wang@ruc.edu.cn

Yu Liu
dokiliu@pku.edu.cn

Ye Yuan
yuan-ye@bit.edu.cn

Xiaoyong Du
duyong@ruc.edu.cn

Ji-Rong Wen
jrwen@ruc.edu.cn

[1] School of Information, Renmin University of China, Beijing, China

[2] Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China

[3] Wangxuan Institute of Computer Technology, Peking University, Beijing, China

[4] School of Computer Science and technology, Beijing Institute of Technology, Beijing, China

[5] MOE Key Lab DEKE, Renmin University of China, Beijing, China

[6] Beijing Key Lab of Big Data Management and Analysis Method, Renmin University of China, Beijing, China

1 Introduction

Computing link-based similarity is an overarching problem in graph analysis and mining. Amid the existing similarity measures [28,37,45,46], SimRank has emerged as a popular metric for assessing structural similarities between nodes in a graph. SimRank was introduced by Jeh and Widom [10] to formalize the intuition that "two pages are similar if they are referenced by similar pages." Given a directed graph $G = (V, E)$ with n nodes $\{v_1, \ldots, v_n\}$ and m edges, the SimRank matrix S defines the similarity between any two nodes v_i and v_j as follows:

$$S(i,j)=\begin{cases}1, & \text{for } i=j;\\ \displaystyle\sum_{v_{i'}\in\mathcal{I}(v_i)}\sum_{v_{j'}\in\mathcal{I}(v_j)}\frac{c\cdot S(i',j')}{d_{in}(v_i)\cdot d_{in}(v_j)}, & \text{for } i\neq j.\end{cases} \tag{1}$$

Here, c is a decay factor typically set to 0.6 or 0.8 [10,23]. $\mathcal{I}(v_i)$ denotes the set of in-neighbors of v_i, and $d_{in}(v_i)$ denotes the in-degree of v_i. SimRank aggregates similarities of multi-hop neighbors of v_i and v_j to produce high-quality similarity measure and has been adopted in various applications such as recommendation systems [17], link prediction [24], and graph embeddings [32].

A fundamental obstacle for studying SimRank is the lack of ground truths on large graphs. Currently, the only methods that compute the SimRank matrix is Power Method and its variations [10,22], which inherently takes $\Omega(n^2)$ space and at least $\Omega(n^2)$ time as there are $\Omega(n^2)$ node pairs in the graphs. This complexity is infeasible on large graphs ($n \geq 10^6$). Consequently, the majority of recent works [7,11,13,14,18,21,26,29,31,36,41] focus on *single-source and top-k queries*. Given a source node v_i, a single-source query asks for the SimRank similarity between every node and v_i, and a top-k query asks for the k nodes with the highest SimRank similarities to v_i. Unfortunately, computing ground truths for the single-source and top-k queries on large graphs still remains an open problem. To the best of our knowledge, Power Method is still the only way to obtain exact single-source and top-k results, which is not feasible on large graphs. Due to the hardness of exact computation, existing works on single-source and top-k queries focus on approximate computations with efficiency and accuracy guarantees.

The lack of ground truths has severely limited our understanding towards SimRank and SimRank algorithms. First of all, designing approximate algorithms without the ground truths is like shooting in the dark. Most existing works take the following approach: they evaluate the accuracy on small graphs where the ground truths can be obtained by the Power Method with $\Omega(n^2)$ space complexity. Then, they report the efficiency/scalability results on large graphs with consistent parameters. This approach is flawed for the reason that consistent parameters may still lead to unfair comparisons. For example, some of the existing methods generate a fixed number of random walks from each node, while others fix the maximum error ε and generate $\frac{\log n}{\varepsilon^2}$ random walks from each node. If we increase the graph size n, the comparison becomes unfair as the latter methods require more random walks from each node. Secondly, it is known that the structure of large real-world graphs can be very different from that of small graphs. Consequently, the accuracy results on small graphs can only serve as a rough guideline for accessing the actual error of the algorithms in real-world applications. We believe that the only right way to evaluate the effectiveness of a SimRank algorithm is to evaluate its results against the ground truths on large real-world graphs.

Second, the lack of ground truths has also prevented us from exploiting the distribution of SimRank on real-world graphs. For example, it is known [4] that the PageRank of most real-world graphs follows the power-law distribution. The natural question is that, does SimRank also follow the power-law distribution on real-world graphs? Furthermore, the performances of some existing methods [35] depend on the *density* of the SimRank, which is defined as the percentage of node pairs with SimRank similarities larger than some threshold ε. Analyzing the distribution or density of SimRank is clearly infeasible without the ground truths.

Finally, the lack of ground truths restricts us to conduct scientific benchmarking experiments towards these existing approximation algorithms. Without insightful experimental observations, we are hard to explore the connections between algorithms' characteristics and performances. For example, what kinds of algorithms tend to show better scalabilities? Algorithms belonging to which categories can perform better trade-off lines? A comprehensive benchmarking survey is fundamentally based on the ground truths.

Exact Single-Source SimRank Computation. In this paper, we study the problem of computing the exact single-source SimRank results on large graphs. A key insight is that exactness does not imply absolutely zero error. This is because SimRank values may be infinite decimals, and we can only store these values with finite precision. Moreover, we note that the ground truths computed by Power Method also incur an error of at most c^L, where L is the number of iterations in Power Method. In most applications, L is set to be large enough such that c^L is smaller than the numerical error and thus can be ignored. In this paper, we aim to develop an algorithm that answers single-source SimRank queries with an additive error of at most $\varepsilon_{min} = 10^{-7}$. Note that the float type in various programming languages usually supports precision of up to 6 or 7 decimal places. So by setting $\varepsilon_{min} = 10^{-7}$, we guarantee the algorithm returns the same answers as the ground truths in the float type. As we shall see, this precision is extremely challenging for existing methods. To make the exact computation possible, we are also going to allow a small probability to fail. We define the probabilistic exact single-source SimRank algorithm as follows.

Definition 1 With probability at least $1 - 1/n$, for *every* source node $v_i \in V$, a probabilistic exact single-source SimRank algorithm answers the single-source SimRank query of v_i with additive error of at most $\varepsilon_{min} = 10^{-7}$.

Our Contributions In this paper, we propose ExactSim, the first algorithm that enables probabilistic exact single-source SimRank queries on large graphs. We show that existing single-source methods share a common complexity term $O\left(\frac{n \log n}{\varepsilon^2_{min}}\right)$ and thus are unable to achieve exactness on large graphs. However, ExactSim runs in $O\left(\frac{\log n}{\varepsilon^2_{min}} + m \log \frac{1}{\varepsilon_{min}}\right)$ time, which is feasible for both large graph size m and small error guarantee ε_{min}. We also apply several non-trivial optimization techniques to reduce the query cost and space overhead of ExactSim. In our empirical study, we show that ExactSim is able to compute the ground truth with a precision of up to 7 decimal places within one hour on graphs with billions of edges. Hence, we believe ExactSim is an effective tool for producing the ground truths for single-source SimRank queries on large graphs.

Comparison with the conference version [33] We make the following new contributions over the conference version.

- We conduct a comprehensive survey on all single-source SimRank algorithms which can support large graphs. We summarize the complexity of each method and analyze the reasons why these methods cannot achieve exactness on large graphs.
- Based on the ground truths provided by ExactSim, we conduct the first empirical study on the accuracy/cost trade-offs of existing approximate single-source algorithms on large real-world graphs and synthetic graphs.
- We use ExactSim to exploit various properties of SimRank on large real-world graphs. In particular, we show that the single-source SimRank values follow the power-law distribution on real-world graphs. We also study the density of SimRank values on large graphs.

2 Preliminaries and related work

In this section, we review the state-of-the-art single-source SimRank algorithms which can support large graphs. We introduce a taxonomy to classify these algorithms into three categories: Monte Carlo methods, iterative methods, and local push/sampling methods. Note that our ExactSim algorithm is largely inspired by three prior works: Linearization [26], PRSim [36], and pooling [21], and we will describe them in details. In Sect. 5, we will also use the ground truths provided by ExactSim to evaluate the algorithms mentioned in this section. Table 1 summarizes the notations used in this paper.

Table 1 Table of notations

Notation	Description
n, m	The numbers of nodes and edges in G
$\mathcal{I}(v_i), \mathcal{O}(v_i)$	the in/out-neighbor set of node v_i
$S, S(i,j)$	The SimRank matrix and the SimRank similarity of v_i and v_j
c	The decay factor in the definition of SimRank
$\varepsilon, \varepsilon_{min}$	Additive error parameter and error required for exactness ($\varepsilon_{min} = 10^{-7}$)
P, D	The transition matrix and the diagonal correction matrix
$\boldsymbol{\pi}_i, \boldsymbol{\pi}_i^\ell,$	The Personalized PageRank and ℓ-hop Personalized PageRank vectors of node v_i
$\mathbf{h}_i^\ell$	the ℓ-hop hitting probability vector of v_i

2.1 Monte Carlo methods

A popular interpretation of SimRank is the *meeting probability* of random walks. In particular, we consider a random walk from node u that, at each step, moves to a random *in-neighbor* with probability $\sqrt{c}$, and stops at the current node with probability $1-\sqrt{c}$. Such a random walk is called a *$\sqrt{c}$-walk*. Suppose we start a $\sqrt{c}$-walk from node v_i and a $\sqrt{c}$-walk from node v_j, we call the two $\sqrt{c}$-walks *meet* if they visit the same node at the same step. It is known [31] that

$$S(i,j) = \Pr[\text{two } \sqrt{c}\text{-walks from } v_i \text{ and } v_j \text{ meet}]. \tag{2}$$

According to Eq. (2), we can employ Monte Carlo sampling to estimate $S(i,j)$. That is, by simulating adequate pairs of $\sqrt{c}$-walks from nodes v_i, v_j, the percentage of the walks that meet in the walking process serves as the estimator of $S(i,j)$. Hence, we classify the approximation algorithms as Monte Carlo methods if they use the fraction of target random walks to estimate the meeting probability based on Eq. (2) or its variants. We will introduce some representative Monte Carlo methods as below, and the complexities of these methods are listed in Table 2.

MC [6] makes use of Eq. (2) to derive a Monte Carlo algorithm for computing single-source SimRank. In the preprocessing phase, we simulate R $\sqrt{c}$-walks from each node in V. Given a source node v_i, we compare the $\sqrt{c}$-walks from v_i and from each node $v_j \in V$ and use the fraction of $\sqrt{c}$-walks that meet as an estimator for $S(i,j)$. By standard concentration inequalities, the maximum error of estimated $S(i,j)$ is bounded by ε with high probability if we set $R = O\left(\frac{\log n}{\varepsilon^2}\right)$, leading to a preprocessing time of $O\left(\frac{n \log n}{\varepsilon^2}\right)$.

READS [11] is an optimized version of the MC-based algorithm. The key idea is to build an index of nR compressed $\sqrt{c}$-walks such that the algorithm only needs to generate a few more $\sqrt{c}$-walks in the query phase. An appealing feature of READS is that its index supports efficient insertions and deletions of edges. Consequently, READS is able to support approximate single-source queries on large dynamic graphs. The theoretical query cost of READS remains $O\left(\frac{n \log n}{\varepsilon^2}\right)$.

TSF [29] is a MC-based algorithm for single-source and top-k SimRank queries on both static and dynamic graphs. TSF builds an index that consists of R_g *one-way graphs*, each of which contains the coupling of random walks of length T from each node. In the query phase, TSF samples R_q more random walks for each one-way graph to provide the final estimators. TSF allows two random walks to meet multiple times and assumes that there is no cycle with a length shorter than T, leading to a lower precision in practice. The

Table 2 Comparison of MC-based SimRank algorithms

Algorithm	Query time	Preprocessing time	Index size	Dynamic update time
MC [6]	$O\left(n \log n/\varepsilon^2\right)$	$O\left(n \log n/\varepsilon^2\right)$	$O\left(n \log n/\varepsilon^2\right)$	–
READS [11]	$O\left(n \log n/\varepsilon^2\right)$	$O\left(n \log n/\varepsilon^2\right)$	$O\left(n \log n/\varepsilon^2\right)$	$O\left(\log n/\varepsilon^2\right)$
TSF [29]	$O\left(n \log n/\varepsilon^2\right)$	$O\left(n \log n/\varepsilon^2\right)$	$O\left(n \log n/\varepsilon^2\right)$	$O\left(\log n/\varepsilon^2\right)$
Uniwalk [25]	$O\left(n^2 \mathrm{l\dot{o}g} n/\varepsilon^2\right)$	0	0	$O\left(n^2 \mathrm{l\dot{o}g} n/\varepsilon^2\right)$

query time of TSF is bounded by $O(nR_gR_q)$, which is in turn bounded by $O\left(\frac{n \log n}{\varepsilon^2}\right)$ for ε additive error.

Uniwalk [25] is a MC-based method for single-source and top-k SimRank computation on undirected graphs. It randomly generates R unidirectional random walks from the given source node s. With the help of a rectified factor, Uniwalk regards the probability of the node s walking along the unidirectional path to the terminal node t as the SimRank value $S(s,t)$, that is, two random walks starting from s and t meet at the midpoint of the original unidirectional path. The query time of Uniwalk is bounded by $O(RL)$, where L denotes the expected length of the unidirectional path. However, the rectified factor can influence the error bound. On the graph with a hub node, R can reach $O\left(\frac{n^2 \mathrm{l\dot{o}g} n}{\varepsilon^2}\right)$ for ε additive error. Hence, the query time of Uniwalk can be bounded by $O\left(\frac{n^2 \mathrm{l\dot{o}g} n}{\varepsilon^2}\right)$.

2.2 Iterative methods

Given a graph $G = (V, E)$, let P denote the (reverse) *transition matrix*, that is, $P(i, j) = 1/d_{in}(v_j)$ for $v_i \in \mathcal{I}(v_j)$, and $P(i, j) = 0$ otherwise. S denotes the SimRank matrix. Yu et al. [44] proved that the definition formula of SimRank can be expressed as

$$S = \left(cP^\top SP\right) \vee I, \tag{3}$$

where I denotes an $n \times n$ identity matrix and $\vee$ is an element-wise maximum operator that for any matrices $A, B \in \mathcal{R}^{n\times n}$ and $\forall i, j \in \{0, 1, ..., n-1\}$, $(A \vee B)(i, j) = \max\{A(i, j), B(i, j)\}$. Equation (3) provides an iterative calculation method to derive SimRank results. That is, we can initialize $S = I$ and repeat the iteration to update matrix S. We classify all the SimRank algorithms as iterative methods if they calculate SimRank values via iterative updating based on Eq. (3) or its variants. We list all the iterative methods which can support single-source SimRank queries on large graphs in the following. Table 3 summarizes the complexities of these iterative methods.

Linearization and ParSim It is shown in two independent works, Linearization [26] and ParSim [42], that the iterative definition Eq. (3) can be expressed as the following linear summation:

$$S = cP^\top SP + D = \sum_{\ell=0}^{+\infty} c^\ell \left(P^\ell\right)^\top DP^\ell, \tag{4}$$

where D is the *diagonal correction matrix* with each diagonal element $D(k, k)$ taking value from $1 - c$ to 1. Consequently, a single-source query for node v_i can be computed by

$$S \cdot \mathbf{e_i} = \sum_{\ell=0}^{+\infty} c^\ell \left(P^\ell\right)^\top DP^\ell \cdot \mathbf{e_i}, \tag{5}$$

where $\mathbf{e}_i$ denotes the one-hot vector with the i-th element being 1 and all other elements being 0. Assuming the diagonal matrix D is correctly given, the single-source query for node v_i can be approximated by

$$S_L \cdot \mathbf{e_i} = \sum_{\ell=0}^{L} c^\ell \left(P^\ell\right)^\top DP^\ell \cdot \mathbf{e_i}, \tag{6}$$

where L is the number of iterations. After L iterations, the additive error reduces to c^L. So setting $L = O\left(\log \frac{1}{\varepsilon}\right)$ is sufficient to guarantee a maximum error of ε. At the ℓ-th iterations, the algorithm performs $2\ell + 1$ matrix–vector multiplications to calculate $c^\ell \left(P^\ell\right)^\top DP^\ell \cdot \mathbf{e_i}$, and each matrix-vector multiplication takes $O(m)$ time. Consequently, the total query time is bounded by $O\left(\sum_{\ell=1}^{L} m\ell\right) =$

Table 3 Comparison of iterative SimRank algorithms

Algorithm	Query time	Preprocessing time	Index size	Dynamic update time
Linearization [26]	$O\left(m \log^2 \frac{1}{\varepsilon}\right)$	$O(n \log \frac{1}{\varepsilon} \log \frac{n}{\varepsilon} \log n/\varepsilon^2)$	$O(n)$	–
ParSim [42]	$O\{\min\{m \log \frac{1}{\varepsilon}, d^{2 \log \frac{1}{\varepsilon}}\}\}$	0	0	–

$O(mL^2) = O\left(m \log^2 \frac{1}{\varepsilon}\right)$. Maehara et al. and Yu et al. also show in [26] and [42] that if we first compute and store the transition probability vectors $\mathbf{u}_\ell = P^\ell \cdot \mathbf{e_i}$ for $\ell = 0, \ldots, L$, then we can use the following equation to compute

$$S_L \cdot \mathbf{e_i} = D \cdot \mathbf{u}_0 + cP^\top (D \cdot \mathbf{u}_1 + \cdots + cP^\top (D \cdot \mathbf{u}_{T-1} + cP^\top \cdot D \cdot \mathbf{u}_T) \cdots). \quad (7)$$

This optimization reduces the query time to $O\left(m \log \frac{1}{\varepsilon}\right)$, while it requires a memory size of $O(nL) = O\left(n \log \frac{1}{\varepsilon}\right)$, which is usually several times larger than the graph size m. Therefore, [26] only uses the $O\left(m \log^2 \frac{1}{\varepsilon}\right)$ algorithm in the experiments.

Besides the large space overhead, another problem with Linearization and ParSim is that the diagonal correction matrix D is hard to compute. Linearization [26] formulates D as the solution to a linear system and proposes a Monte Carlo solution that takes $O\left(\frac{n \log n}{\varepsilon^2}\right)$ to derive an ε-approximation of D. On the other hand, ParSim directly sets $D = (1 - c)I$, where I is the identity matrix. This approximation basically ignores the first meeting constraint and has been adopted in many other SimRank works [8,9,13,16,38,39,41]. It is shown that the similarities calculated by this approximation are different from the actual SimRank [13]. However, the quality of this approximation is still a myth due to the lack of ground truths on large graphs.

2.3 Local push/sampling methods

Compared with Monte Carlo and iterative methods, local push/sampling methods locally restrict each SimRank update operation and omit to touch a large fraction of nodes on the graphs in each update. Hence, the time cost of each update operation is smaller than $O(n)$. This allows local push/sampling methods to outperform other methods in terms of scalability. In the following, we will present a brief introduction to the local push/sampling single-source SimRank methods which can support large graphs. The complexities of these methods are listed in Table 4.

SLING [31] is an index-based SimRank algorithm that supports fast single-source and top-k queries on static graphs. Let $\mathbf{h}_i^\ell = \left(\sqrt{c}P\right)^\ell \cdot \mathbf{e_i}$ denote the *ℓ-hop hitting probability vector* of v_i. Note that $\mathbf{h}_i^\ell$ describes the probability of an $\sqrt{c}$-walk from node v_i *visiting* each node at its ℓ-th step. [31] suggests that Eq. (5) can be rewritten as

$$S(i,j) = \sum_{\ell=0}^{\infty} \sum_{k=1}^{n} \mathbf{h}_i^\ell(k) \cdot \mathbf{h}_j^\ell(k) \cdot D(k,k). \quad (8)$$

where $D(k,k)$ denotes the k-th entry in the diagonal correction matrix D. It is shown [31] that $D(k,k)$ can be characterized by the meeting probability of two $\sqrt{c}$-walks from the same node v_k:

$$D(k,k) = \Pr[\text{two } \sqrt{c}\text{-walks from } v_k \text{ never meet}]. \quad (9)$$

This interpretation implies a simple Monte Carlo algorithm for estimating $D(k,k)$: we simulate R pairs of $\sqrt{c}$-walks from v_k and use the fraction of pairs that do not meet as the estimator for $D(k,k)$. By setting $R = O\left(\frac{\log n}{\varepsilon^2}\right)$, we can approximate each $D(k,k)$ with additive error ε. SLING precomputes each $D(k,k)$ in the preprocessing phase using $O\left(\frac{n \log n}{\varepsilon^2}\right)$ time. SLING also precomputes $\mathbf{h}_i^\ell(k)$ with additive error ε for each ℓ and $v_i, v_k \in V$, using a *local push* algorithm [2]. Given a single-source query for node v_i, SLING retrieves $\mathbf{h}_i^\ell(k)\,\mathbf{h}_j^\ell(k)$ and $D(k,k)$ for each $v_j, v_k \in V$ from the index and uses Eq. (8) to estimate $S(i,j)$ for each $v_j \in V$. SLING answers a single-source query with time $O(\min\{n/\varepsilon, m\})$, and the index size is bounded by $O\left(\frac{n}{\varepsilon}\right)$.

ProbeSim [21] is an index-free solution based on reverse local sampling and local push. ProbeSim starts by sampling a $\sqrt{c}$-walk from the source node v_i. For the ℓ-th node v_k on the $\sqrt{c}$-walk, ProbeSim uses a *Probe algorithm* to reversely sample each node v_j at level ℓ with probability $\mathbf{h}_j^\ell(k)$, the hitting probability that any other node $v_j \in V$ can reach v_k at the ℓ-th step. It is shown in [21] that each sample takes $O(n)$ time, and we need $O\left(\frac{\log n}{\varepsilon^2}\right)$ samples to ensure an maximum error of ε with high probability. Consequently, the query time of ProbeSim is bounded by $O\left(\frac{n \log n}{\varepsilon^2}\right)$. ProbeSim naturally works on dynamic graphs due to its index-free nature.

Table 4 Comparison of local push/sampling SimRank algorithms

Algorithm	Query time	Preprocessing time	Index size	Dynamic update time
SLING [31]	$O\left(n/\varepsilon\right)$	$O\left(\frac{m}{\varepsilon} + \frac{n \log \frac{n}{\delta}}{\varepsilon^2}\right)$	$O\left(n/\varepsilon\right)$	-
	$O\left(m \log^2 \frac{1}{\varepsilon}\right)$			
ProbeSim [21]	$O\left(n \log n/\varepsilon^2\right)$	0	0	$O\left(n \log n/\varepsilon^2\right)$
PRSim [36]	$O\left(n \log n \cdot \lVert\boldsymbol{\pi}_i\rVert^2/\varepsilon^2\right)$.	$O\left(m/\varepsilon\right)$	$O\left(\min\{\frac{n}{\varepsilon}, m\}\right)$	-
TopSim [14]	$O\left(m^{2n}/n^{2n}\right)$	0	0	$O\left(m^{2n}/n^{2n}\right)$

PRSim [36] introduces a partial indexing and a probe algorithm. Let $\boldsymbol{\pi}_i^\ell = (1-\sqrt{c})\mathbf{h}_i^\ell = (1-\sqrt{c})\left(\sqrt{c}P\right)^\ell \cdot \mathbf{e}_i$ denote the *ℓ-hop Personalize PageRank vector* of v_i. In particular, $\boldsymbol{\pi}_i^\ell(k)$ is the probability that a $\sqrt{c}$-walk from node v_i *stops* at node v_k in exactly ℓ steps. PRSim suggests that Eq. (5) can be rewritten as

$$S(i,j) = \frac{1}{(1-\sqrt{c})^2}\sum_{\ell=0}^{\infty}\sum_{k=1}^{n}\boldsymbol{\pi}_i^\ell(k)\cdot\boldsymbol{\pi}_j^\ell(k)\cdot D(k,k). \quad (10)$$

PRSim precomputes $\boldsymbol{\pi}_j^\ell(k)$ with additive error ε for each ℓ and $v_j, v_k \in V$, using a *local push* algorithm [2]. To avoid overwhelming index size, PRSim only precomputes $\boldsymbol{\pi}_j^\ell(k)$ for a small subset of v_k. Furthermore, PRSim computes D by estimating the product $\boldsymbol{\pi}_i^\ell(k)\cdot D(k,k)$ together with an $O\left(\frac{\log n}{\varepsilon^2}\right)$ time Monte Carlo algorithm. Finally, PRSim proposes a new Probe algorithm that samples each node v_j according to $\boldsymbol{\pi}_j^\ell(k)$. The average query time of PRSim is bounded by
$O\left(\frac{n\log n}{\varepsilon^2}\cdot\sum_{k=1}^{n}\boldsymbol{\pi}(k)^2\right)$, where $\boldsymbol{\pi}(k)$ denotes the PageRank of v_k. It is well known that on scale-free networks, the PageRank vector $\boldsymbol{\pi}$ follows the power-law distribution, and thus, $\|\boldsymbol{\pi}\|^2 = \sum_{k=1}^{n}\boldsymbol{\pi}(k)^2$ is a value much smaller than 1. However, for worst-case graphs or even some "bad" source nodes on scale-free networks, the running time of PRSim remains $O\left(\frac{n\log n}{\varepsilon^2}\right)$.
TopSim [14] is an index-free algorithm based on local exploitation. Given source node v_i, TopSim firstly finds all nodes v_k reachable from v_i within $\ell = 1,\ldots,L$ steps. For each such v_k on the ℓ-th level, TopSim deterministically computes $\mathbf{h}_j^\ell(k)$, the probability that each v_j reaches v_k in exactly ℓ steps. [14] also proposes various optimizations to reduce the query cost. Due to the dense structures of real-world networks, TopSim is only able to exploit a few levels on large graphs, which leads to a low precision.

2.4 Other related work

Besides the state-of-the-art methods that we discuss above, there are several other techniques for SimRank computation, which we review in the following. *Power method* [10] is the classic algorithm that computes all-pair SimRank similarities for a given graph. Power method recursively computes the SimRank Matrix S based on Eq. (3). Several follow-up works [23,40,44] improve the efficiency or effectiveness of the power method in terms of either efficiency or accuracy. However, these methods still incur $O(n^2)$ space overheads, as there are $O(n^2)$ pairs of nodes in the graph. Finally, there are existing works on *SimRank similarity join* [27,30,48] and the variants of SimRank [3,6,19,43,47], but the proposed solutions are inapplicable for top-k and single-source SimRank queries.
Pooling Finally, pooling [21] is an experimental method for evaluating the accuracy of top-k SimRank algorithms without the ground truths. Suppose the goal is to compare the accuracy of top-k queries for z algorithms $A_1,\ldots,A_z$. Given a query node v_i, we retrieve the top-k nodes returned by each algorithm, remove the duplicates, and merge them into a pool. Note that there are at most ℓk nodes in the pool. Then, we estimate $S(i,j)$ for each node v_j in the pool using the Monte Carlo algorithm. We set the number of random walks to be $O\left(\frac{\log n}{\varepsilon_{min}^2}\right)$ so that we can obtain the ground truth of $S(i,j)$ with high probability. After that, we take the k nodes with the highest SimRank similarity to v_i from the pool as the ground truth of the top-k query and use this "ground truth" to evaluate the precision of each of the ℓ algorithms. Note that the set of these k nodes is not the actual ground truth. However, it represents the best possible k nodes that can be found by the ℓ algorithms that participate in the pool and thus can be used to compare the quality of these algorithms.

Although pooling is proved to be effective in our scenario where ground truths are hard to obtain, it has some drawbacks. First of all, the precision results obtained by pooling are *relative* and thus cannot be used outside the pool. This is because the top-k nodes from the pool is not the actual ground truth. Consequently, an algorithm that achieves 100% precision in the pool may have a precision of 0% when compared to the actual top-k result. Secondly, the complexity of pooling z algorithms is $O\left(\frac{kz\log n}{\varepsilon_{min}^2}\right)$, so pooling is only feasible for evaluating top-k queries with small k. In particular, we cannot use pooling to evaluate the single-sources queries on large graphs.

2.5 Limitations of existing methods

We now analyze the reasons why existing methods are unable to achieve exactness (a.k.a an error of at most $\varepsilon_{min} = 10^{-7}$). First of all, ParSim and TSF ignore the first meeting constraint and thus incur large errors. For other methods that enforce the first meeting constraint, they all incur a complexity term of $O\left(\frac{n\log n}{\varepsilon^2}\right)$, either in the preprocessing phase or in the query phase. In particular, SLING and Linearization simulate $O\left(\frac{n\log n}{\varepsilon^2}\right)$ random walks to estimate the diagonal correction matrix D. For ProbeSim, MC, READS, and PRSim, this complexity is caused by simulating random walks in the query phase or the preprocessing phase. The $O\left(\frac{n\log n}{\varepsilon^2}\right)$ complexity is infeasible for exact SimRank computation on large graphs, since it combines two expensive terms n and $\frac{1}{\varepsilon_{min}^2}$. As an example, we consider the IT dataset used in our experiment, with $4*10^7$ nodes and over 1 billion edges. In order to achieve a maximum error of $\varepsilon_{min} = 10^{-7}$, we need to simulate $\frac{n\log n}{\varepsilon^2} \approx 10^{23}$ random walks. This may take years, even with parallelization on a cluster of thousands of machines.

Besides, there are many works focusing on all-pairs SimRank queries [9,16,23,34,38]. As we shall show in Sect. 5.3, the number of node pairs whose SimRank values are more than 10^{-4} can nearly achieve n^2. For large graphs with million nodes, like Twitter(TW) dataset with 4×10^7 nodes, this can cost 10^4 TB for storage, not to mention the exact SimRank computation for each node pair. Hence, it's may infeasible for exact all-pairs SimRank computation within reasonable time.

3 Basic ExactSim algorithm

In this section, we present ExactSim, a probabilistic algorithm that computes the exact single-source SimRank results within reasonable running time. We first present a basic version of ExactSim. In Sect. 4, we will introduce some more advanced techniques to optimize the query and space cost.

Our ExactSim algorithm is largely inspired by three prior works: pooling [21], Linearization [26], and PRSim [36]. We now discuss how ExactSim extends from these existing methods in details. These discussions will also reveal the high level ideas of the ExactSim algorithm.

1. Despite its limitations, pooling [21] provides a key insight for achieving exactness: while an $O\left(\frac{n\log n}{\varepsilon^2}\right)$ algorithm is not feasible for exact SimRank computation on large graphs, we can actually afford an $O\left(\frac{\log n}{\varepsilon^2}\right)$ algorithm. The $\frac{1}{\varepsilon^2}$ term is still expensive for $\varepsilon = \varepsilon_{min} = 10^{-7}$; however, the new complexity reduces the dependence on the graph size n to logarithmic and thus achieves very high scalability.
2. Linearization [26] and ParSim [42] show that if the diagonal correction matrix D is correctly given, then we can compute the exact single-source SimRank results in $O\left(m\log_{\frac{1}{c}}\frac{1}{\varepsilon_{min}}\right)$ time and $O\left(n\log_{\frac{1}{c}}\frac{1}{\varepsilon_{min}}\right)$ extra space. For typical setting of c (0.6 to 0.8), the number of iterations $\log_{\frac{1}{c}}\frac{1}{\varepsilon_{min}} = \log 10^7 \le 73$ is a constant, so this complexity is essentially the same as that of performing BFS multiple times on the graphs. The scalability of the algorithm is confirmed in the experiments of [42], where D is set to be $(1-c)I$. Moreover, the exact algorithms [28] for Personalized PageRank and PageRank also incur a running time of $O\left(m\log\frac{1}{\varepsilon_{min}}\right)$ and have been widely used for computing ground truths on large graphs.
3. While the $O\left(\frac{n\log n}{\varepsilon^2}\right)$ complexity seems unavoidable as we need to estimate each entry in the diagonal correction matrix D with additive error ε, PRSim [36] shows that it only takes $O\left(\frac{\log n}{\varepsilon^2}\right)$ time to estimate the product $\boldsymbol{\pi}_i^\ell(k)\cdot D(k,k)$ with additive error ε for each $k = 1,\dots,n$ and $\ell = 0,\dots,\infty$, where $\boldsymbol{\pi}_i^\ell$ is the ℓ-hop Personalized PageRank vector of v_i. This result provides two crucial observations: 1) It is possible to answer an single-source query without an ε-approximation of each $D(k,k)$; 2) The accuracy of each $D(k,k)$ should depend on $\boldsymbol{\pi}_i(k)$, the Personalized PageRank of v_k with respect to the source node v_i.

Algorithm 1: Basic ExactSim Algorithm

```
Input: Graph $G$ with transition matrix $P$, source node $v_i$,
       maximum error $\varepsilon$
Output: Estimated single-source SimRank vector $S \cdot \mathbf{e_i}$
$L = \left\lceil \log_{\frac{1}{c}} \frac{2}{\varepsilon} \right\rceil$;
$\boldsymbol{\pi}_i^0, \boldsymbol{\pi}_i = (1-\sqrt{c})\mathbf{e}_i$;
for $\ell$ from 1 to $L$ do
    $\boldsymbol{\pi}_i^\ell = \sqrt{c}P \cdot \boldsymbol{\pi}_i^{\ell-1}$;
    $\boldsymbol{\pi}_i = \boldsymbol{\pi}_i + \boldsymbol{\pi}^\ell$;
$R = \frac{6\log n}{(1-\sqrt{c})^4\varepsilon^2}$;
for $k$ from 1 to $n$ do
    Invoke Algorithm 2 with $R(k) = \lceil R \cdot \boldsymbol{\pi}_i(k) \rceil$ to obtain an
    estimator $\hat{D}(k,k)$ for $D(k,k)$;
$\mathbf{s}^0 = \frac{1}{1-\sqrt{c}}\hat{D} \cdot \boldsymbol{\pi}_i^L$;
for $\ell$ from 1 to $L$ do
    $\mathbf{s}^\ell = \sqrt{c}P^\top \cdot \mathbf{s}^{\ell-1} + \frac{1}{1-\sqrt{c}}\hat{D} \cdot \boldsymbol{\pi}_i^{L-\ell}$ ;
    Clear $\mathbf{s}^{\ell-1}$;
return $\mathbf{s}^L$;
```

We combine the ideas of PRSim and Linearization/ ParSim to derive the basic ExactSim algorithm. Given an error parameter ε, ExactSim fixes the total number of $\sqrt{c}$-walk samples to be $R = O\left(\frac{\log n}{\varepsilon^2}\right)$ and distributes a fraction of $R \cdot \boldsymbol{\pi}_i(k)$ samples (note that $\sum_{k=1}^{n} \boldsymbol{\pi}_i(k) = 1$) to estimate $D(k,k)$. Then, it performs Linearization/ ParSim with the estimated D to obtain the single-source result. The algorithm runs in $O\left(\frac{\log n}{\varepsilon^2} + m\log\frac{1}{\varepsilon}\right)$ time and uses $O\left(n\log\frac{1}{\varepsilon}\right)$ extra space. Since both complexity terms $O\left(\frac{\log n}{\varepsilon^2}\right)$ and $O\left(m\log\frac{1}{\varepsilon}\right)$ are feasible for $\varepsilon_{min} = 10^{-7}$ and large graph size m, we have a working algorithm for exact single-source SimRank queries on large graphs.

Algorithm 1 illustrates the pseudocode of the basic ExactSim algorithm. Note that to cope with Personalized PageRank, we use the fact that $\boldsymbol{\pi}_i^\ell = \left(1-\sqrt{c}\right)\cdot\left(\sqrt{c}P\right)^\ell \cdot \mathbf{e}_i$ and

Algorithm 2: Basic method for estimating $D(k,k)$

```
Input: Graph $G$, node $v_k$, number of samples $R(k)$
Output: $\hat{D}(k,k)$ as an estimation for $D(k,k)$
$\hat{D}(k,k) = 0$;
for $x$ from 1 to $R(k)$ do
    Sample two independent $\sqrt{c}$-walks from $v_k$;
    if The two $\sqrt{c}$-walks do not meet then
        $\hat{D}(k,k) = \hat{D}(k,k) + 1/R(k)$;
return $\hat{D}(k,k)$;
```

rewrite Eq. (5) as

$$S \cdot \mathbf{e}_i = \frac{1}{1-\sqrt{c}} \sum_{\ell=0}^{\infty} \left(\sqrt{c}P^\top\right)^\ell D \cdot \boldsymbol{\pi}_i^\ell. \tag{11}$$

Given a source node v_i and a maximum error ε, we first set the number of iterations L to be $L = \left\lceil \log_{\frac{1}{c}} \frac{2}{\varepsilon} \right\rceil$ (line 1). We then iteratively compute the ℓ-hop Personalized PageRank vector $\boldsymbol{\pi}_i^\ell = \left(\sqrt{c}P\right)^\ell \cdot \mathbf{e}_i$ for $\ell = 0, \ldots, L$, as well as the Personalized PageRank vector $\boldsymbol{\pi}_i = \sum_{\ell=0}^{L} \boldsymbol{\pi}_i^\ell$ (lines 2-5). To obtain an estimator $\hat{D}$ for the diagonal correction matrix D, we set the total number of samples to be $R = \frac{6\log n}{(1-\sqrt{c})^4\varepsilon^2}$ (line 6). For each $D(k,k)$, we set $R(k) = \lceil R\boldsymbol{\pi}_i(k)\rceil$ and invoke Algorithm 2 to estimate $D(k,k)$ (lines 7-8). Algorithm 2 essentially simulates $R(k)$ pairs of $\sqrt{c}$-walks from node v_k and uses the fraction of pairs that do not meet as an estimator $\hat{D}(k,k)$ for $D(k,k)$. Finally, we use Eq. (11) to iteratively compute $\mathbf{s}^0 = \frac{1}{1-\sqrt{c}} \hat{D} \cdot \boldsymbol{\pi}_i^L$,

$$\begin{aligned}\mathbf{s}^1 &= \sqrt{c}P^\top \cdot \mathbf{s}^0 + \frac{1}{1-\sqrt{c}} \hat{D} \cdot \boldsymbol{\pi}_i^{L-1} \\ &= \frac{1}{1-\sqrt{c}} \left(\sqrt{c}P^\top \cdot \hat{D} \cdot \boldsymbol{\pi}_i^L + \hat{D} \cdot \boldsymbol{\pi}_i^{L-1}\right)\end{aligned} \tag{12}$$

(lines 9-12),..., and

$$\begin{aligned}\mathbf{s}^L &= \frac{\left(\sqrt{c}P^\top \left(\cdots(\sqrt{c}P^\top \cdot \hat{D} \cdot \boldsymbol{\pi}^L + \hat{D} \cdot \boldsymbol{\pi}^{L-1}) + \cdots\right) + \hat{D} \cdot \boldsymbol{\pi}^0\right)}{1-\sqrt{c}} \\ &= \frac{1}{1-\sqrt{c}} \sum_{\ell=0}^{L} \left(\sqrt{c}P^\top\right)^\ell \hat{D} \cdot \boldsymbol{\pi}_i^\ell.\end{aligned} \tag{13}$$

We return $\mathbf{s}^L$ as the single-source query result (line 13).

Analysis To derive the running time and space overhead of the basic ExactSim algorithm, note that computing and storing each ℓ-hop Personalized PageRank vector $\boldsymbol{\pi}_i^\ell$ takes $O(m)$ time and $O(n)$ space. This results in a running time of $O(mL)$ and a space overhead of $O(nL)$. To estimate the diagonal correction matrix D, the algorithm simulates R pairs of $\sqrt{c}$-walks, each of which takes $\frac{1}{\sqrt{c}} = O(1)$ time. Therefore, the running time for estimating D can be bounded by $O(R)$. Finally, computing each $\mathbf{s}^\ell$ also takes $O(m)$ time, resulting an additional running time of $O(mL)$. Summing up all costs, we have the total running time is bounded by $O(mL+R) = O\left(\frac{\log n}{\varepsilon^2} + m\log\frac{1}{\varepsilon}\right)$, and the space overhead is bounded by $O(nL) = O\left(n\log\frac{1}{\varepsilon}\right)$.

We now analyze the error of the basic ExactSim algorithm. Recall that ExactSim returns $\mathbf{s}^L(j)$ as the estimator for $S(i,j)$, the SimRank similarity between the source node v_i and any other node v_j. We have the following theorem.

Theorem 1 *With probability at least $1-1/n$, for any source node $v_i \in V$, the basic ExactSim provide an single-source SimRank vector $\mathbf{s}^L$ such that, for any node $v_j \in V$, we have $\left|\mathbf{s}^L(j) - S(i,j)\right| \le \varepsilon$.*

Theorem 1 essentially states that with high probability, the basic ExactSim algorithm can compute any single-source SimRank query with additive ε. The proof of Theorem 1 is fairly technical. However, the basic idea is to show that the variance of the estimator $\mathbf{s}^L(j)$ can be bounded by $O(\frac{1}{R}) = O(\varepsilon^2)$. In particular, we first note that by Eq. (13), $\mathbf{s}^L(j)$ can be expressed as

$$\begin{aligned}\mathbf{s}^L(j) = \mathbf{e}_j^\top \cdot \mathbf{s}^L &= \frac{1}{1-\sqrt{c}} \mathbf{e}_j^\top \cdot \sum_{\ell=0}^{L} \left(\sqrt{c}P^\top\right)^\ell \hat{D} \cdot \boldsymbol{\pi}_i^\ell \\ &= \frac{1}{(1-\sqrt{c})^2} \sum_{\ell=0}^{L} \left((1-\sqrt{c})\left(\sqrt{c}P\right)^\ell \cdot \mathbf{e}_j\right)^\top \cdot \hat{D} \cdot \boldsymbol{\pi}_i^\ell.\end{aligned}$$

Since $(1-\sqrt{c})\left(\sqrt{c}P\right)^\ell \cdot \mathbf{e}_j = \boldsymbol{\pi}_j^\ell$, we have

$$\mathbf{s}^L(j) = \frac{1}{(1-\sqrt{c})^2} \sum_{\ell=0}^{L} \left(\boldsymbol{\pi}_j^\ell\right)^\top \cdot \hat{D} \cdot \boldsymbol{\pi}_i^\ell. \tag{14}$$

Summing up over the diagonal elements of D follows that

$$\mathbf{s}^L(j) = \frac{1}{(1-\sqrt{c})^2} \sum_{\ell=0}^{L} \sum_{k=1}^{n} \boldsymbol{\pi}_i^\ell(k) \cdot \boldsymbol{\pi}_j^\ell(k) \cdot \hat{D}(k,k). \tag{15}$$

We observe that there are two discrepancies between $\mathbf{s}^L(j)$ and the actual SimRank value $S(i,j)$ (10): 1) We change the number of iterations from ∞ to L, and 2) we use the estimator $\hat{D}$ to replace actual diagonal correction matrix D. For the first approximation, we can bound the error by $c^L \le \varepsilon/2$ if ExactSim sets $L = \left\lceil \log_{\frac{1}{c}} \frac{2}{\varepsilon} \right\rceil$. Consequently, we only need to bound the error from replacing D with $\hat{D}$. In particular, we will make use of the following Bernstein Inequality.

Lemma 1 *(Bernstein Inequality [5]) Let $X_1, \cdots, X_R$ be independent random variables with $|X_i| < b$ for $i =$*

$1, \ldots, R$. *Let* $X = \frac{1}{R} \cdot \sum_{i=1}^{R} X_i$, *we have*

$$\Pr[|X - \mathrm{E}[X]| \geq \lambda] \leq 2 \cdot \exp\left(-\frac{\lambda^2 \cdot R}{2R \cdot \mathrm{Var}[X] + 2b\lambda/3}\right), \tag{16}$$

where $\mathrm{Var}[X]$ *is the variance of* X.

To make use of Lemma 1, we need to express $\mathbf{s}^L(j)$ as the average of independent random variables. In particular, let $\hat{D}_r(k,k)$, $r = 1, \ldots, R(k)$ denote the r-th estimator of $D(k,k)$ by Algorithm 2. We observe that each $\hat{D}_r(k,k)$ is a Bernoulli random variable, that is, $\hat{D}_r(k,k) = 1$ with probability $D(k,k)$ and $\hat{D}_r(k,k) = 0$ with probability $1 - D(k,k)$. We have

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

Let $\rho(k) = R(k)/R$ be the fraction of pairs of $\sqrt{c}$-walks assigned to v_k, it follows that

$$\mathbf{s}^L(j) = \frac{1}{R} \cdot \frac{1}{(1-\sqrt{c})^2} \sum_{k=1}^{n} \sum_{r=1}^{R\rho(k)} \frac{\sum_{\ell=0}^{L} \boldsymbol{\pi}_i^{\ell}(k) \cdot \boldsymbol{\pi}_j^{\ell}(k)}{\rho(k)} \cdot \hat{D}_r(k,k) \tag{17}$$

We will treat each $\frac{\sum_{\ell=0}^{L} \boldsymbol{\pi}_i^{\ell}(k) \cdot \boldsymbol{\pi}_j^{\ell}(k)}{\rho(k)} \cdot \hat{D}_r(k,k)$ as an independent random variable. The number of such random variables is $\sum_{k=1}^{n} R\rho(k) = R$, so we have expressed $\mathbf{s}^L(j)$ as the average of R independent random variables. To utilize Lemma 1, we first bound the variance of $\mathbf{s}^L(j)$.

Lemma 2 *The variance of* $\mathbf{s}^L(j)$ *is bounded by*

$$\mathrm{Var}[\mathbf{s}^L(j)] \leq \frac{1}{(1-\sqrt{c})^4 R} \sum_{k=1}^{n} \frac{\boldsymbol{\pi}_i(k)^2 \boldsymbol{\pi}_j(k)^2}{\rho(k)} \cdot D(k,k). \tag{18}$$

In particular, by setting $\rho(k) = R(k)/R = \lceil R\boldsymbol{\pi}_i(k) \rceil / R$ *in the basic ExactSim algorithm, we have*

$$\mathrm{Var}[\mathbf{s}^L(j)] \leq \frac{1}{(1-\sqrt{c})^4 R}. \tag{19}$$

Note that we only need Inequality (19) to derive the error bound for the basic ExactSim algorithm. The more complex Inequality (18) will be used to design various optimization techniques.

Proof of Lemma 2 Note that $\hat{D}_r(k,k)$ is a Bernoulli random variable with expectation $D(k,k)$ and thus has variance $\mathrm{Var}[\hat{D}_r(k,k)] = D(k,k)(1 - D(k,k)) \leq D(k,k)$. Since $\hat{D}_r(k,k)$'s are independent random variables, we have

$$\begin{aligned}&\mathrm{Var}[\mathbf{s}^L(j)] \\ &= \frac{1}{(1-\sqrt{c})^4 R^2} \sum_{k=1}^{n} \sum_{r=1}^{R\rho(k)} \left(\frac{\sum_{\ell=0}^{L} \boldsymbol{\pi}_i^{\ell}(k) \cdot \boldsymbol{\pi}_j^{\ell}(k)}{\rho(k)}\right)^2 \\ &\quad \cdot \mathrm{Var}[\hat{D}_r(k,k)] \\ &= \frac{1}{(1-\sqrt{c})^4 R} \sum_{k=1}^{n} \frac{\left(\sum_{\ell=0}^{L} \boldsymbol{\pi}_i^{\ell}(k) \cdot \boldsymbol{\pi}_j^{\ell}(k)\right)^2}{\rho(k)} \\ &\quad \cdot D(k,k)(1 - D(k,k)).\end{aligned}$$

By the Cauchy–Schwarz inequality, we have

$$\begin{aligned}\left(\sum_{\ell=0}^{L} \boldsymbol{\pi}_i^{\ell}(k) \cdot \boldsymbol{\pi}_j^{\ell}(k)\right)^2 &\leq \left(\sum_{\ell=0}^{L} \boldsymbol{\pi}_i^{\ell}(k)\right)^2 \left(\sum_{\ell=0}^{L} \boldsymbol{\pi}_j^{\ell}(k)\right)^2 \\ &\leq \boldsymbol{\pi}_i(k)^2 \boldsymbol{\pi}_j(k)^2.\end{aligned}$$

Combining with the fact that $1 - D(k,k) \leq 1$, we have

$$\mathrm{Var}[\mathbf{s}^L(j)] \leq \frac{1}{(1-\sqrt{c})^4 R} \sum_{k=1}^{n} \frac{\boldsymbol{\pi}_i(k)^2 \boldsymbol{\pi}_j(k)^2}{\rho(k)} \cdot D(k,k). \tag{20}$$

and the first part of the lemma follows.

Plugging $\rho(k) = R(k)/R = \lceil R\boldsymbol{\pi}_i(k) \rceil / R \geq \boldsymbol{\pi}_i(k)$ into Lemma 2, we have

$$\begin{aligned}\mathrm{Var}[\mathbf{s}^L(j)] &\leq \frac{1}{(1-\sqrt{c})^4 R} \sum_{k=1}^{n} \frac{\boldsymbol{\pi}_i(k)^2 \boldsymbol{\pi}_j(k)^2}{\boldsymbol{\pi}_i(k)} \cdot D(k,k) \\ &\leq \frac{1}{(1-\sqrt{c})^4 R} \sum_{k=1}^{n} \boldsymbol{\pi}_i(k).\end{aligned}$$

For the last inequality, we use the fact that $D(k,k) \leq 1$ and $\boldsymbol{\pi}_j(k) \leq 1$. Finally, since $\sum_{k=1}^{n} \boldsymbol{\pi}_i(k) = 1$, we have $\mathrm{Var}[\mathbf{s}^L(j)] \leq \frac{1}{(1-\sqrt{c})^4 R}$, and the second part of the lemma follows. □

Proof of Theorem 1 We are now ready to prove Theorem 1. To utilize Bernstein Inequality given in Lemma 1, we also need to bound b, the maximum value of the random variables $\sum_{\ell=0}^{L} \frac{\boldsymbol{\pi}_i^{\ell}(k) \cdot \boldsymbol{\pi}_j^{\ell}(k)}{\rho(k)} \cdot \hat{D}_r(k,k)$. We have

$$\frac{\sum_{\ell=0}^{L} \boldsymbol{\pi}_i^{\ell}(k) \cdot \boldsymbol{\pi}_j^{\ell}(k)}{\boldsymbol{\pi}_i(k)} \cdot \hat{D}_r(k,k) \leq \frac{\sum_{\ell=0}^{L} \boldsymbol{\pi}_i^{\ell}(k)}{\boldsymbol{\pi}_i(k)} \leq \frac{\boldsymbol{\pi}_i(k)}{\boldsymbol{\pi}_i(k)} = 1.$$

Applying Bernstein Inequality with $b = 1$ and $\mathrm{Var}[\mathbf{s}^L(j)] \le \frac{1}{(1-\sqrt{c})^4 R}$, where $R = \frac{6\log n}{(1-\sqrt{c})^4 \varepsilon^2}$, we have $\Pr[|\mathbf{s}^L(j) - E[\mathbf{s}^L(j)]| > \varepsilon/2] < 1/n^3$. Combining with the $\varepsilon/2$ error introduced by the truncation L, we have $\Pr[|\mathbf{s}^L(j) - S(i,j)| > \varepsilon] < 1/n^3$. By union bound over all possible target nodes $j = 1, \ldots, n$ and all possible source nodes $i = 1, \ldots, n$, we ensure that for all n possible source node and n target nodes,

$$\Pr[\forall i, j, |\mathbf{s}^L(j) - S(i,j)| > \varepsilon] < 1/n,$$

and the theorem follows. □

4 Optimizations

Although the basic ExactSim algorithm is a working algorithm for exact single-source SimRank computation on large graphs, it suffers from some drawbacks. First of all, the $O(n \log \frac{1}{\varepsilon})$ space overhead can be several times larger than the actual graph size m. Secondly, we still need to simulate $R = O\left(\frac{\log n}{\varepsilon^2}\right)$ of pairs of $\sqrt{c}$-walks, which is a significant cost for $\varepsilon_{min} = 10^{-7}$. Although parallelization can help, we are still interested in developing algorithmic techniques that reduces the number of random walks. In this section, we provide three optimization techniques that address these drawbacks.

Sparse Linearization We design a sparse version of Linearization that significantly reduces the $O\left(n \log \frac{1}{\varepsilon}\right)$ space overhead while retaining the $O(\varepsilon)$ error guarantee. Recall that this space overhead is causing by storing the ℓ-hop Personalized PageRank vectors $\boldsymbol{\pi}_i^\ell$ for $\ell = 0, \ldots, L$. We propose to make the following simple modification: Instead of storing the dense vector $\boldsymbol{\pi}_i^\ell$, we sparsify the vector by removing all entries of with $\boldsymbol{\pi}_i^\ell(k) \le (1-\sqrt{c})^2\varepsilon$. To understand the effectiveness of this approach, recall that a nice property of the ℓ-hop Personalized PageRank vectors is that all possible entries sum up to $\sum_{\ell=0}^{\infty}\sum_{k=1}^{n} \boldsymbol{\pi}_i^\ell(k) = \sum_{k=1}^{n} \boldsymbol{\pi}^\ell(k) = 1$. By the Pigeonhole principle, the number of $\boldsymbol{\pi}_i^\ell(k)$'s that are larger than $(1-\sqrt{c})^2\varepsilon$ is bounded by $\frac{1}{(1-\sqrt{c})^2\varepsilon}$. Thus, the space overhead is reduced to $O\left(\frac{1}{\varepsilon}\right)$. This overhead is acceptable for exact computations where we set $\varepsilon = \varepsilon_{min} = 10^{-7}$, as it does not scale with the graph size.

Sampling according to $\boldsymbol{\pi}_i(k)^2$ Recall that in the basic ExactSim algorithm, we simulate R pairs of $\sqrt{c}$-walks in total, and distribute $\boldsymbol{\pi}_i(k)$ fraction of the R samples to estimate $D(k,k)$. A natural question is that, is there a better scheme to distribute these R samples? It turns out if we distribute the samples according to $\boldsymbol{\pi}_i(k)^2$, we can further reduce the variance of the estimator and hence achieve a better running time. More precisely, we will set $R(k) = R\left\lceil \frac{\boldsymbol{\pi}_i(k)^2}{\|\boldsymbol{\pi}_i\|^2} \right\rceil$, where $\|\boldsymbol{\pi}_i\|^2 = \sum_{k=1}^{n} \boldsymbol{\pi}_i(k)^2$ is the squared norm of the Personalized PageRank vector $\boldsymbol{\pi}_i$.

Algorithm 3: Improved method for estimating $D(k,k)$

```
Input: Graph $G$, node $v_k$, sample number $R(k)$
Output: An estimator $\hat{D}(k,k)$ for $D(k,k)$
if $d_{in}(v_k) = 0$ then
    return $\hat{D}(k,k) = 1$;
else if $d_{in}(v_k) = 1$ then
    return $\hat{D}(k,k) = 1 - c$;
$P^\ell(x,k) = 0$ for $\ell \ge 0, x \in V$;
$P^0(k,k) = 1$;
$E_k = 0$;
for $\ell$ from 0 to $\infty$ do
    for each $v_q$ with nonzero $\left(P^\top\right)^\ell(k,q)$ do
        Calculate $Z_\ell(k,q)$ using equation (22);
    for $\ell'$ from 0 to $\ell$ do
        for each $v_{q'}$ with nonzero $\left(P^\top\right)^{\ell-\ell'}(k,q')$ do
            for each $v_x$ with nonzero $\left(P^\top\right)^{\ell'}(q',x)$ do
                for each $v_q \in \mathcal{I}(v_x)$ do
                    $\left(P^\top\right)^{\ell'+1}(q',q) += \frac{\left(P^\top\right)^{\ell'}(q',x)}{d_{in}(v_x)}$;
                    $E_k += 1$;
                    if $E_k \ge \frac{2R(k)}{\sqrt{c}}$ then
                        $\ell(k) = \ell$ and goto OUTLOOP;
    $\ell = \ell + 1$;
OUTLOOP;
$\hat{D}(k,k) = 1 - \sum_{\ell=1}^{\ell(k)} \sum_{q=1}^{n} Z_\ell(k,q)$;
for $z$ from 1 to $R(k)$ do
    Sample two independent non-stop random walks from $v_k$;
    if Two random walks reaches nodes $v_x$ and $v_y$ at the $\ell(k)$ steps without meeting then
        Sample a $\sqrt{c}$-walks from $v_x$ and $v_y$;
        if the two $\sqrt{c}$-walks meet then
            $\hat{D}(k,k) = \hat{D}(k,k) - c^{\ell(k)}/R(k)$;
return $\hat{D}(k,k)$;
```

Local deterministic exploitation for D The inequality (18) in Lemma 2 also suggests that we can reduce the variance of the estimator $\mathbf{s}^L(j)$ by refining the Bernoulli estimator $\hat{D}(k,k)$. Recall that we sample $R(k) = \lceil R\boldsymbol{\pi}_i(k) \rceil$ or $R(k) = R\left\lceil \frac{\boldsymbol{\pi}_i(k)^2}{\|\boldsymbol{\pi}_i\|^2} \right\rceil$ pairs of $\sqrt{c}$-walks to estimate $D(k,k)$. If $\boldsymbol{\pi}_i(k)$ is large, we will simulate a large number of $\sqrt{c}$-walks from v_k to estimate $D(k,k)$. In that case, the first few steps of these random walks will most likely visit the same local structures around v_k, so it makes sense to exploit these local structures deterministically, and use the random walks to approximate the global structures. More precisely, let $Z_\ell(k)$ denote the probability that two $\sqrt{c}$-walks from v_k first meet at the ℓ-th step. Since these events are mutually exclusive for different

ℓ's, we have

$$D(k,k) = 1 - \Pr[\text{two } \sqrt{c}\text{-walks from } v_k \text{meet}]$$
$$= 1 - \sum_{\ell=1}^{\infty} Z_\ell(k).$$

The idea is to deterministically compute $\sum_{\ell=1}^{\ell(k)} Z_\ell(k)$ for some tolerable step $\ell(k)$, and using random walks to estimate the other part $\sum_{\ell=\ell(k)+1}^{\infty} Z_\ell(k)$. It is easy to see that by deterministically computing $Z_\ell(k)$ for the first $\ell(k)$ levels, we reduce the variance $\mathrm{Var}(D(k,k))$ by at least $c^{\ell(k)}$.

A simple algorithm to compute $Z_\ell(k)$ is to list all possible paths of length ℓ from v_k and aggregate all meeting probabilities of any two paths. However, the number of paths increases rapidly with the length ℓ, which makes this algorithm inefficient on large graphs. Instead, we will derive the close form for $Z_\ell(k)$ in terms of the transition probabilities. In particular, let $Z_\ell(k,q)$ denote the probability that two $\sqrt{c}$-walks first meet at node v_q at their ℓ-th steps. We have $Z_\ell(k) = \sum_{q=1}^{n} Z_\ell(k,q)$, and hence

$$D(k,k) = 1 - \sum_{\ell=1}^{\infty}\sum_{q=1}^{n} Z_\ell(k,q). \tag{21}$$

Recall that P^ℓ (the ℓ-th power of the (reverse) transition matrix P) is the ℓ-step (reverse) transition matrix. We have the following lemma that relates $Z_\ell(k,q)$ with the transition probabilities.

Lemma 3 *$Z_\ell(k,q)$ satisfies the following recursive form:*

$$\begin{aligned} Z_\ell(k,q) =& c^\ell \left(P^\top\right)^\ell (k,q)^2 \\ & - \sum_{\ell'=1}^{\ell-1}\sum_{q'=1}^{n} c^{\ell'} \left(P^\top\right)^{\ell'} (q',q)^2 Z_{\ell-\ell'}(k,q'). \end{aligned} \tag{22}$$

Proof Note that $\left(\sqrt{c}\right)^\ell \left(P^\top\right)^\ell (k,q)$ is the probability that a $\sqrt{c}$-walk from v_k visits v_q at its ℓ-th step. Consequently, $c^\ell \left(P^\top\right)^\ell (k,q)^2$ is the probability that two $\sqrt{c}$-walks from node v_k visit node v_q at their ℓ-th step simultaneously. To ensure this is the first time that the two $\sqrt{c}$-walks meet, we subtract the probability mass that the two $\sqrt{c}$-walks have met before. In particular, recall that $Z_{\ell'}(k,q')$ is the probability that two $\sqrt{c}$-walks from node v_k first meet at $v_{q'}$ in exactly ℓ' steps. Due to the memoryless property of the $\sqrt{c}$-walk, the two $\sqrt{c}$-walks will behave as two new $\sqrt{c}$-walks from $v_{q'}$ after their ℓ'-th step. The probability that these two new $\sqrt{c}$-walks visit is v_q in exact $\ell-\ell'$ steps is $c^{\ell-\ell'} \left(P^\top\right)^{\ell-\ell'} (q',q)^2$. Summing up q' from 1 to n and ℓ' from 1 to $\ell-1$, the lemma follows. □

Given a node v_k and a pre-determined target level $\ell(k)$, Lemma 3 also suggests a simple algorithm to compute $Z_\ell(k,q)$ for all $\ell \le \ell(k)$. We start by performing BFS from node v_k for up to $\ell(k)$ levels to calculate the transition probabilities $\left(P^\top\right)^\ell (k,q)$ for $\ell = 0,\ldots,\ell(k)$ and $v_q \in V$. For each node q' visited at the ℓ'-th level, we start a BFS from q' for $\ell(k)-\ell'$ levels to calculate $\left(P^\top\right)^{\ell(k)-\ell'} (q',q)$ for $\ell = 1,\ldots,\ell(k)$ and $v_q \in V$. Then, we use equation (22) to calculate $Z_\ell(k,q)$ for $\ell = 0,\ldots,\ell(k)$ and $q \in V$. Note that this approach exploits strictly less edges than listing all possible paths of length $\ell(k)$, as some of the paths are combined in the computation of the transition probabilities.

However, a major problem with the above method is that the target level $\ell(k)$ has to be predetermined, which makes the running time unpredictable. An improper value of $\ell(k)$ could lead to the explosion of the running time. Instead, we will use an adaptive algorithm to compute $Z_\ell(k)$.

Algorithm 3 illustrates the new method for estimating $D(k,k)$. Given a node v_k and a sample number $R(k)$, the goal is to give an estimator for $D(k,k)$. For the two trivial case $d_{in}(k) = 0$ and $d_{in}(k) = 1$, we return $D(k,k) = 1$ and $1-c$ accordingly (lines 1-4). For other cases, we iteratively compute all possible transition probabilities $\left(P^\top\right)^{\ell'+1} (q',q)$ for all $v_{q'}$ that is reachable from k with $\ell-\ell'$ steps (lines 5-10). Note that these $v_{q'}$'s are the nodes with $\left(P^\top\right)^{\ell-\ell'} (k,q') > 0$. To ensure the deterministic exploitation stops in time, we use a counter E_k to record the total number of edges traversed so far (line 11). If E_k exceeds $\frac{2R(k)}{\sqrt{c}}$, the expected number of steps for simulating $R(k)$ pairs of $\sqrt{c}$-walks, we terminate the deterministic exploitation and set $\ell(k)$ as the current target level for v_k (lines 12-13). After we fix $\ell(k)$ and compute $\sum_{\ell=1}^{\ell(k)} Z_\ell(k)$ (lines 14-17), we will use random walk sampling to estimate $\sum_{\ell=\ell(k)+1}^{\infty} Z_\ell(k)$ (lines 18-23). In particular, we start two special random walks from v_k. The random walks do not stop in its first $\ell(k)$ steps; after the $\ell(k)$-th step, each random walk stops with probability $\sqrt{c}$ at each step. It is easy to see that the probability of the two special random walks meet after $\ell(k)$ steps is $\frac{1}{c^{\ell(k)}} \sum_{\ell=\ell(k)+1}^{\infty} Z_\ell(k)$. Consequently, we can use the fraction of the random walks that meet multiplied by $c^{\ell(k)}$ as an unbiased estimator for $\sum_{\ell=\ell(k)+1}^{\infty} Z_\ell(k)$.

Parallelization. The ExactSim algorithm is highly parallelizable as it only uses two primitive operations: matrix-(sparse) vector multiplication and random walk simulation. Both operations are embarrassingly parallelizable on GPUs or multi-core CPUs. The only exception is the local deterministic exploitation for $D(k,k)$. To parallelize this operation, we can apply Algorithm 3 to multiple v_k simultaneously. Furthermore, we can balance the load of each thread by applying Algorithm 3 to nodes v_k's with similar number of samples $R(k)$ in each epoch.

4.1 Analysis

Recall that Algorithm 3 provides an improved method for estimating $D(k,k)$. By invoking Algorithm 3 into the whole ExactSim structure (line 8 in Algorithm 1), we can derive the optimized version of ExactSim. The following theorem presents the complexity analysis of the optimized ExactSim in terms of time cost and space overhead.

Theorem 2 *Let $\boldsymbol{\pi}_i$ denote the Personalized PageRank vector with regards to node v_i. Then, with probability at least $1-\frac{1}{n}$, for any source node $v_i \in V$, the optimized ExactSim can return a single-source SimRank vector $\mathbf{s}^L$ with $O\left(\frac{\|\boldsymbol{\pi}_i\|^2\log n}{\varepsilon^2}+m\log\frac{1}{\varepsilon}\right)$ time cost and $O\left(\frac{1}{\varepsilon}\right)$ space overhead, such that for any node $v_j \in V$, we have $\left\|\mathbf{s}^L(j)-S(i,j)\right\|\le\varepsilon$.*

Concerning the three optimization techniques mentioned above, sparse Linearization may influence the space overhead; Sampling according to $\boldsymbol{\pi}_i(k)^2$ reduces the number of random walks, which can impact the time cost of estimating D. Local deterministic exploitation can reduce the variance $\mathrm{Var}(D(k,k))$, while the level of time and space complexity remains the same due to the setting of $\ell(k)$. Consequently, to prove Theorem 2, we can only analysis sparse Linearization for space bound and Sampling according to $\boldsymbol{\pi}_i(k)^2$ for time cost, respectively.

Firstly, as for the space overhead, the following lemma proves that the sparse Linearization will only introduce an extra additive error of ε. If we scale down ε by a factor of 2, the total error guarantee and the asymptotic running time of ExactSim will remain the same, and the space overhead is reduced to $O\left(\frac{1}{\varepsilon}\right)$.

Lemma 4 *The sparse Linearization introduces an extra additive error of ε and reduces the space overhead to $O\left(\frac{1}{\varepsilon}\right)$.*

Proof We note that the sparse Linearization introduces an extra error of $(1-\sqrt{c})^2\varepsilon$ to each $\boldsymbol{\pi}_i^{\ell}(k)$, $k=1,\ldots,n$, $\ell=0,\ldots,\infty$. According to Eq. (15), the estimator $\mathbf{s}^L(j)$ can be expressed as

$$\mathbf{s}^L(j)=\frac{1}{(1-\sqrt{c})^2}\sum_{\ell=0}^{L}\sum_{k=1}^{n}\left(\boldsymbol{\pi}_i^{\ell}(k)\pm(1-\sqrt{c})^2\varepsilon\right)\cdot\boldsymbol{\pi}_j^{\ell}(k)\cdot\hat{D}(k,k). \tag{23}$$

Thus, the error introduced by sparse Linearization can be bounded by

$$\frac{1}{(1-\sqrt{c})^2}\sum_{\ell=0}^{\infty}\sum_{k=1}^{n}(1-\sqrt{c})^2\varepsilon\cdot\boldsymbol{\pi}_j^{\ell}(k)\cdot\hat{D}(k,k). \tag{24}$$

Using the fact that $\sum_{\ell=0}^{\infty}\sum_{k=1}^{n}\boldsymbol{\pi}_j^{\ell}(k)=1$ and $\hat{D}(k,k)\le 1$, the above error can be bounded by $\frac{1}{(1-\sqrt{c})^2}\cdot(1-\sqrt{c})^2\varepsilon=\varepsilon$, and the lemma follows. □

Then, we analysis the time cost of Algorithm 3. The following lemma shows that by sampling according to $\boldsymbol{\pi}_i(k)^2$, we can reduce the number of sample R by a factor of $\|\boldsymbol{\pi}_i\|^2$.

Lemma 5 *By sampling according to $\boldsymbol{\pi}_i(k)^2$, the number of random samples required is reduced to $O\left(\frac{\|\boldsymbol{\pi}_i\|^2\log n}{\varepsilon^2}\right)$.*

Proof Recall that $\rho(k)$ is the fraction of sample assigned to $D(k,k)$. We have $\rho(k)=\left\lceil\frac{R\boldsymbol{\pi}_i(k)^2}{\|\boldsymbol{\pi}_i\|^2}\right\rceil/R\ge\frac{\boldsymbol{\pi}_i(k)^2}{\|\boldsymbol{\pi}_i\|^2}$. By the inequality (18) in Lemma 2, we can bound the variance of estimator $\mathbf{s}^L(j)$ as

$$\begin{aligned}\mathrm{Var}[\mathbf{s}^L(j)]&\le\frac{1}{(1-\sqrt{c})^4R}\sum_{k=1}^{n}\frac{\boldsymbol{\pi}_i(k)^2\boldsymbol{\pi}_j(k)^2}{\rho(k)}\cdot D(k,k)\\&\le\frac{1}{(1-\sqrt{c})^4R}\|\boldsymbol{\pi}_i\|^2\sum_{k=1}^{n}\boldsymbol{\pi}_j(k)^2\\&=\frac{1}{(1-\sqrt{c})^4R}\|\boldsymbol{\pi}_i\|^2\|\boldsymbol{\pi}_j\|^2.\end{aligned}$$

Here, we use the fact that $\|\boldsymbol{\pi}_j\|^2=\sum_{k=1}^{n}\boldsymbol{\pi}_j(k)^2$ and $D(k,k)\le 1$. Since we need to bound the variance for all possible nodes v_j (and hence all possible $\|\boldsymbol{\pi}_j\|^2$), we make the relaxation that $\|\boldsymbol{\pi}_j\|^2\le\|\boldsymbol{\pi}_j\|_1^2=1$, where $\|\boldsymbol{\pi}_j\|_1^2=(\sum_{k=1}^{n}|\boldsymbol{\pi}_j(k)|)^2$. And thus

$$\mathrm{Var}[\mathbf{s}^L(j)]\le\frac{1}{(1-\sqrt{c})^4R}\|\boldsymbol{\pi}_i\|^2.$$

This suggests that by sampling according to $\boldsymbol{\pi}_i(k)^2$, we reduce the variance of the estimators by a factor $\|\boldsymbol{\pi}_i\|^2$. Recall that the ExactSim algorithm computes the Personalized PageRank vector $\boldsymbol{\pi}_i$ before estimating D; we can obtain the value of $\|\boldsymbol{\pi}_i\|^2$ and scale R down by a factor of $\|\boldsymbol{\pi}_i\|^2$. This simple modification will reduce the running time to $O\left(\frac{\|\boldsymbol{\pi}_i\|^2\log n}{\varepsilon^2}\right)$.

One small technical issue is that the maximum of the random variables $\frac{\sum_{\ell=0}^{\infty}\boldsymbol{\pi}_i^{\ell}(k)\cdot\boldsymbol{\pi}_j^{\ell}(k)}{\rho(k)}\cdot\hat{D}_r(k,k)$ may gets too large as the fraction $\rho(k)$ gets too small. However, by the fact that $\rho(k)=\left\lceil\frac{R\boldsymbol{\pi}_i(k)^2}{\|\boldsymbol{\pi}_i\|^2}\right\rceil/R$ and $\hat{D}_r(k,k)\le 1$, we have

$$\begin{aligned}\frac{\sum_{\ell=0}^{\infty}\boldsymbol{\pi}_i^{\ell}(k)\cdot\boldsymbol{\pi}_j^{\ell}(k)}{\rho(k)}\cdot\hat{D}_r(k,k)&\le\frac{\boldsymbol{\pi}_i(k)}{\rho(k)}\\&=R\boldsymbol{\pi}_i(k)/\left\lceil\frac{R\boldsymbol{\pi}_i(k)^2}{\|\boldsymbol{\pi}_i\|^2}\right\rceil.\end{aligned}$$

If we view the right side of the above equality as a function of $\boldsymbol{\pi}_i(k)$, it takes maximum when $\frac{R\boldsymbol{\pi}_i(k)^2}{\|\boldsymbol{\pi}_i\|^2}=1$, or equivalently $\boldsymbol{\pi}_i(k) = \sqrt{\frac{\|\boldsymbol{\pi}_i\|^2}{R}}$. Thus, the random variables in Eq. (17) can be bounded by $R\sqrt{\frac{\|\boldsymbol{\pi}_i\|^2}{R}} = \|\boldsymbol{\pi}_i\|\sqrt{R}$. Plugging $b = \|\boldsymbol{\pi}_i\|\sqrt{R}$ and $\mathrm{Var}[\mathbf{s}^L(j)] \le \frac{\|\boldsymbol{\pi}_i\|^2}{(1-\sqrt{c})^4 R}$ into Bernstein Inequality, and the lemma follows. □

To demonstrate the effectiveness of sampling according to $\boldsymbol{\pi}_i(k)^2$, notice that in the worst case, $\|\boldsymbol{\pi}_i\|^2$ is as large as $\|\boldsymbol{\pi}_i\|_1^2 = 1$, so this optimization technique is essentially useless. However, it is known [4] that on scale-free networks, the Personalized PageRank vector $\boldsymbol{\pi}_i$ follows a power-law distribution: let $\boldsymbol{\pi}_i(k_j)$ denote the j-th largest entry of $\boldsymbol{\pi}_i$, we can assume $\boldsymbol{\pi}_i(k_j) \sim \frac{j^{-\beta}}{n^{1-\beta}}$ for some power-law exponent $\beta \in (0, 1)$. In this case, $\|\boldsymbol{\pi}_i\|^2$ can be bounded by $O\left(\sum_{j=1}^{n}\left(\frac{j^{-\beta}}{n^{1-\beta}}\right)^2\right) = O\left(\max\left\{\frac{lnn}{n}, \frac{1}{n^{2-2\beta}}\right\}\right)$, and the $\|\boldsymbol{\pi}_i\|^2$ factor becomes significant for any power-law exponent $\beta < 1$.

Note that the expected length of every random walk is $\frac{1}{1-\sqrt{c}}$, which is a constant. Hence, by Lemma 5, the time cost of Algorithm 3 can be bounded by $O\left(\frac{\|\boldsymbol{\pi}_i\|^2 \log n}{\varepsilon^2}\right)$. Recall that after we derive the estimated matrix D, the linearized summation for $\mathbf{s}^L$ takes $O(m \log \frac{1}{\varepsilon})$ time. Consequently, the total time cost of the optimized ExactSim is $O\left(\frac{\|\boldsymbol{\pi}_i\|^2 \log n}{\varepsilon^2} + m \log \frac{1}{\varepsilon}\right)$, which follows Theorem 2.

5 Experiments

In this section, we experimentally study ExactSim and the other single-source algorithms. We first evaluate ExactSim against four methods MC, ParSim, Linearization, and PRSim to prove ExactSim's ability of exact computation (i.e., $\varepsilon_{min} = 10^{-7}$). Then, we conduct an ablation study to demonstrate the effectiveness of the optimization techniques. Finally, based on the ground truths computed by ExactSim, we conduct a comprehensive empirical study on existing single-source SimRank algorithms and SimRank distributions.

Datasets and Environment We use six small datasets, six large datasets, and two dynamic datatsets[1][2][3] The details of these datasets can be found in Table 5. All experiments are conducted on a machine with an Intel(R) Xeon(R) E7-4809 @2.10GHz CPU and 196GB memory.

5.1 Evaluation towards ExactSim

Methods and Parameters We evaluate ExactSim with the four state-of-the-art methods, including one Monte Carlo method: MC [6], two iterative methods: Linearization [26] and ParSim [42], and one Local push/ sampling methods: PRSim [36]. For a fair comparison, we run each algorithm in the single thread mode on static graphs.

Table 5 Datasets

Data set	Type	n	m
PPI (PI)	Undirected	3,890	38739
ca-GrQc (GQ)	Undirected	5,242	28,968
AS-2000(AS)	Undirected	6,474	25,144
CA-HepTh(HT)	Undirected	9,877	51,946
Wikivote (WV)	Directed	7,115	103,689
CA-HepPh (HP)	Undirected	12,008	236978
DBLP-Author (DB)	Undirected	5,425,963	17,298,032
LiveJournal (LJ)	Directed	4,847,571	68,475,391
IndoChina (IC)	Directed	7,414,768	191,606,827
Orkut-Links (OL)	Undirected	3,072,441	234,369,798
It-2004 (IT)	Directed	41,290,682	1,135,718,909
Twitter (TW)	Directed	41,652,230	1,468,364,884
Wiki-Pl (WP)	Dynamic	1,033,050	25,026,208
Wiki-De (WD)	Dynamic	2,166,669	86,337,879

MC has two parameters: the length of each random walk L and the number of random walks per node r. We vary (L, r) from $(5, 50)$ to $(5000, 50000)$ on small graphs and from $(5, 50)$ to $(50, 500)$ on large graphs. ParSim has one parameter L, the number of iterations. We vary it from 50 to 5×10^5 on small graphs and from 5 to 500 on large graphs. Finally, Linearization, PRSim, and ExactSim share the same error parameter ε, and we vary ε from 10^{-1} to 10^{-7} (if possible) on both small and large graphs. We evaluate the optimized ExactSim unless otherwise stated. In all experiments, we set the decay factor c of SimRank as 0.6.

Metrics We use $MaxError$ and $Precision@k$ to evaluate the quality of the single-source and top-k results. Given a source node v_i and an approximate single-source result with n similarities $\hat{S}(i, j)$, $j = 1, \ldots, n$, $MaxError$ is defined to be the maximum error over n similarities: $MaxError = \max_{j=1}^{n}\left|\hat{S}(i, j) - S(i, j)\right|$. Given a source node v_i and an approximate top-k result $V_k = \{v_1, \ldots, v_k\}$, $Precision@k$ is defined to be the percentage of nodes in V_k that coincides with the actual top-k results. In our experiments, we set k to be 500. Note that this is the first time that top-k queries with $k > 100$ are evaluated on large graphs. On each dataset, we generate 50 query nodes for each dataset. For each set of parameters and each method, we issue one query from each query node and report the average $MaxError$ and *Precision@500* among the 50 query nodes.

Experiments on small graphs We first evaluate ExactSim against other single-source algorithms on six small graphs. We compute the ground truths of the single-source and top-k

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

[2] http://law.di.unimi.it/datasets.php

[3] http://konect.cc/categories/Hyperlink/

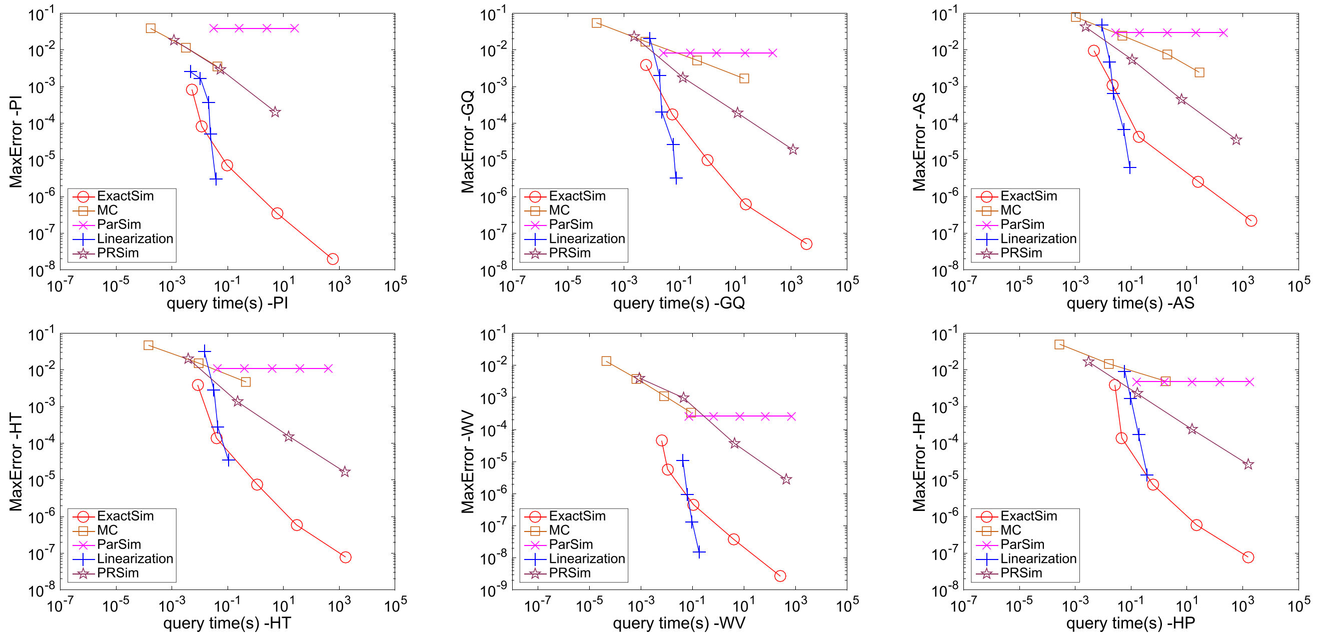


Fig. 1 MaxError v.s. Query time on small graphs

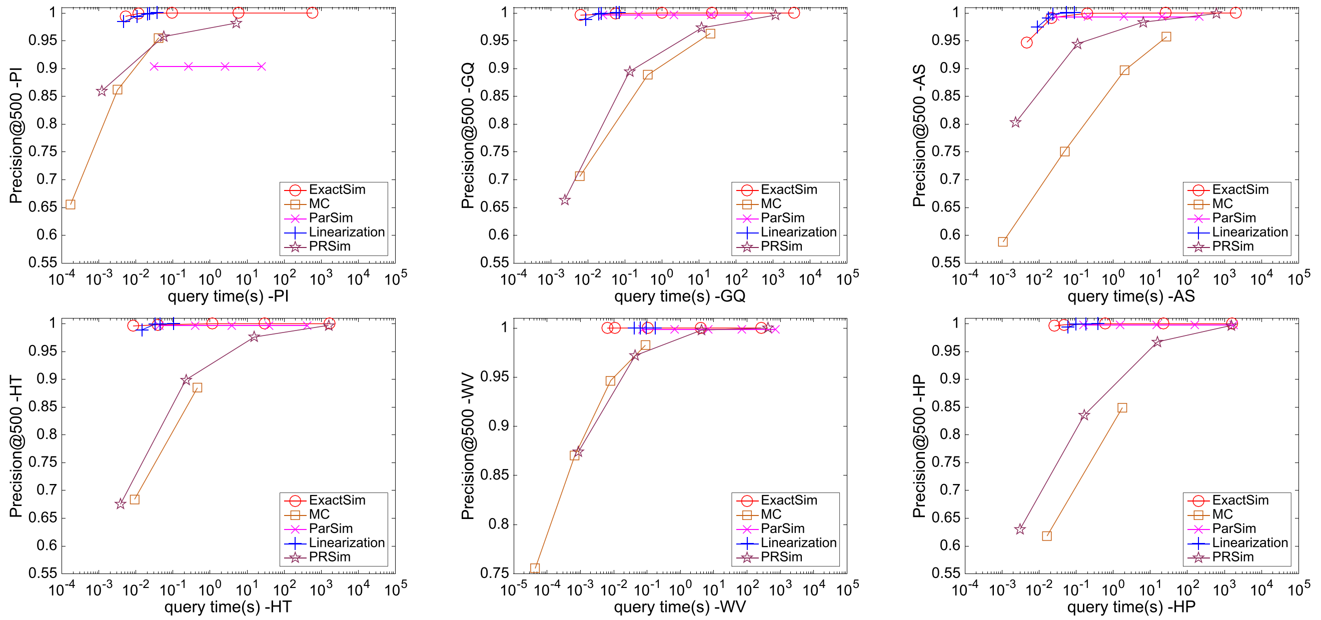


Fig. 2 Precision@500 v.s. Query time on small graphs

queries using Power Method [10]. We omit a method if its query or preprocessing time exceeds 24 hours.

Figure 1 shows the trade-offs between $MaxError$ and the query time of each algorithm. The first observation is that ExactSim is the only algorithm that consistently achieves an error of 10^{-7} within 10^4 seconds. Linearization is able to achieve a faster query time when the error parameter ε is large. However, as we set $\varepsilon \leq 10^{-5}$, Linearization is troubled by its $O\left(\frac{n \log n}{\varepsilon^2}\right)$ preprocessing time and is unable to finish the computation of the diagonal matrix D in 24 hours.

Figure 2 presents the trade-offs between *Precision@500* and query time of each algorithm. We observe that ExactSim with $\varepsilon = 10^{-7}$ is able to achieve a precision of 1 on all six graphs. This confirms the exactness of ExactSim. We also note that ParSim is able to achieve high precisions on most of graphs despite its large $MaxError$ in Fig. 1. This observation demonstrates the effectiveness of

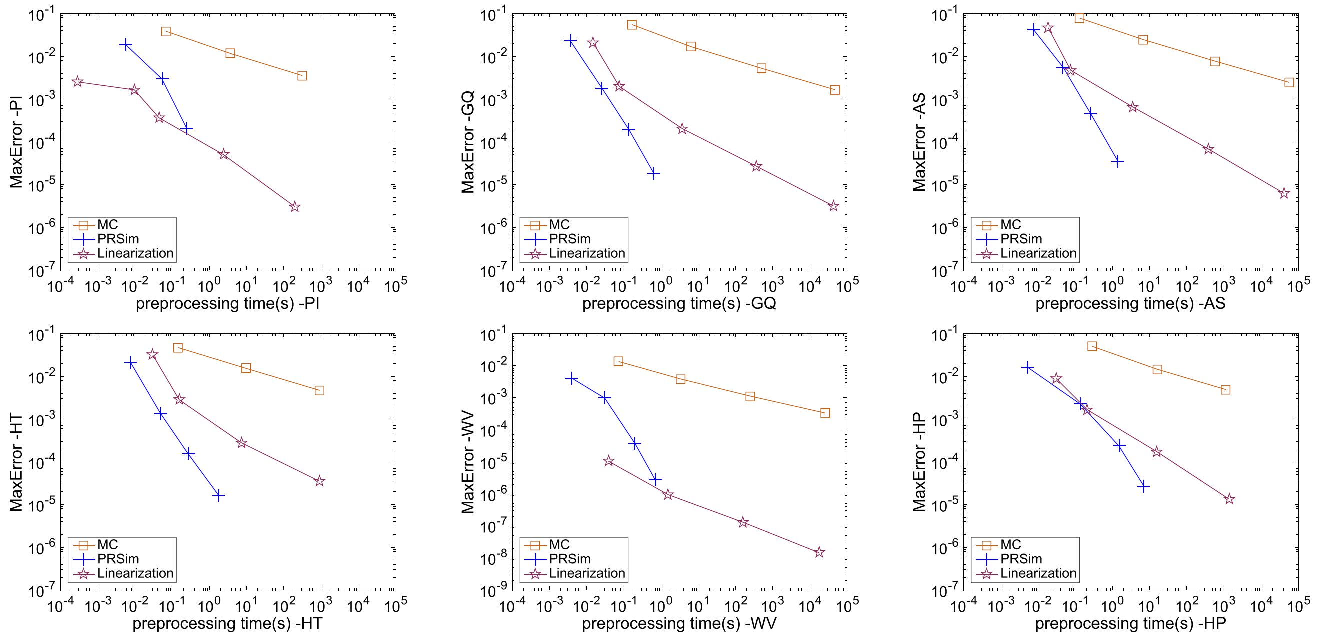


Fig. 3 MaxError v.s. Preprocessing time on small graphs

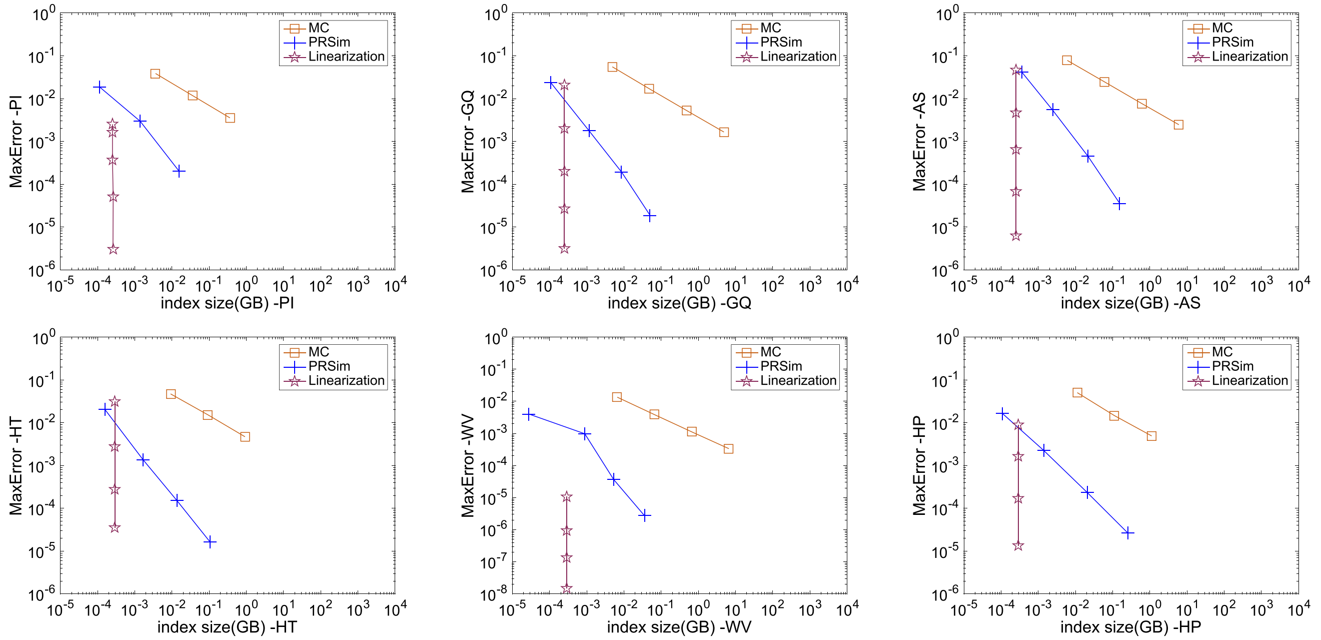


Fig. 4 MaxError v.s. Index size on small graphs

the $D \sim (1-c)I$ approximation on small datasets. Finally, for the index-based methods MC, PRSim, and Linearization, we also plot the trade-offs between $MaxError$ and preprocessing time/index size in Figs. 3 and 4. The index sizes of Linearization form a vertical line, as the algorithm only recomputes and stores a diagonal matrix D. PRSim generally achieves the smallest error given a fixed amount of preprocessing time and index size.

Experiments on large graphs. For now, we have both theoretical and experimental evidence that ExactSim is able to obtain the exact single-source and top-k SimRank results. In this section, we will treat the results computed by ExactSim with $\varepsilon = 10^{-7}$ as the ground truths to evaluate the performance of ExactSim with larger ε on large graphs.

Figures 5 and 6 show the trade-offs between the query time and $MaxError$/*Precision@500* of each algorithm. Figures 7 and 8 display the $MaxError$ and preprocessing time/index

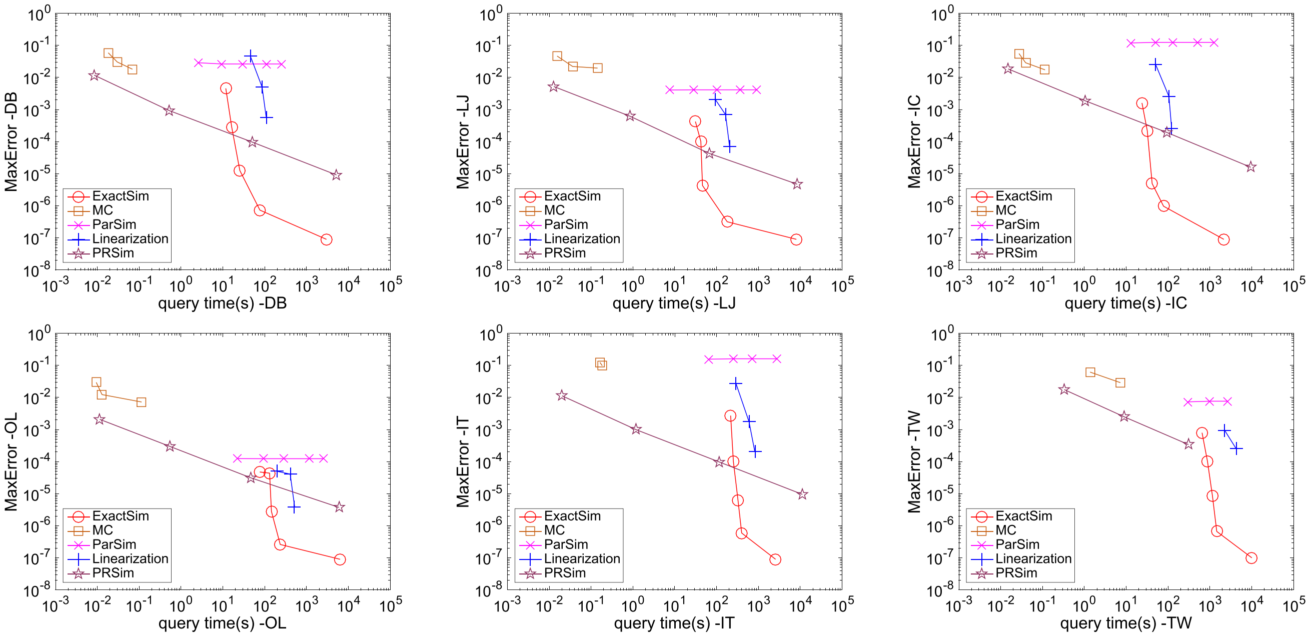


Fig. 5 MaxError v.s. Query time on large graphs

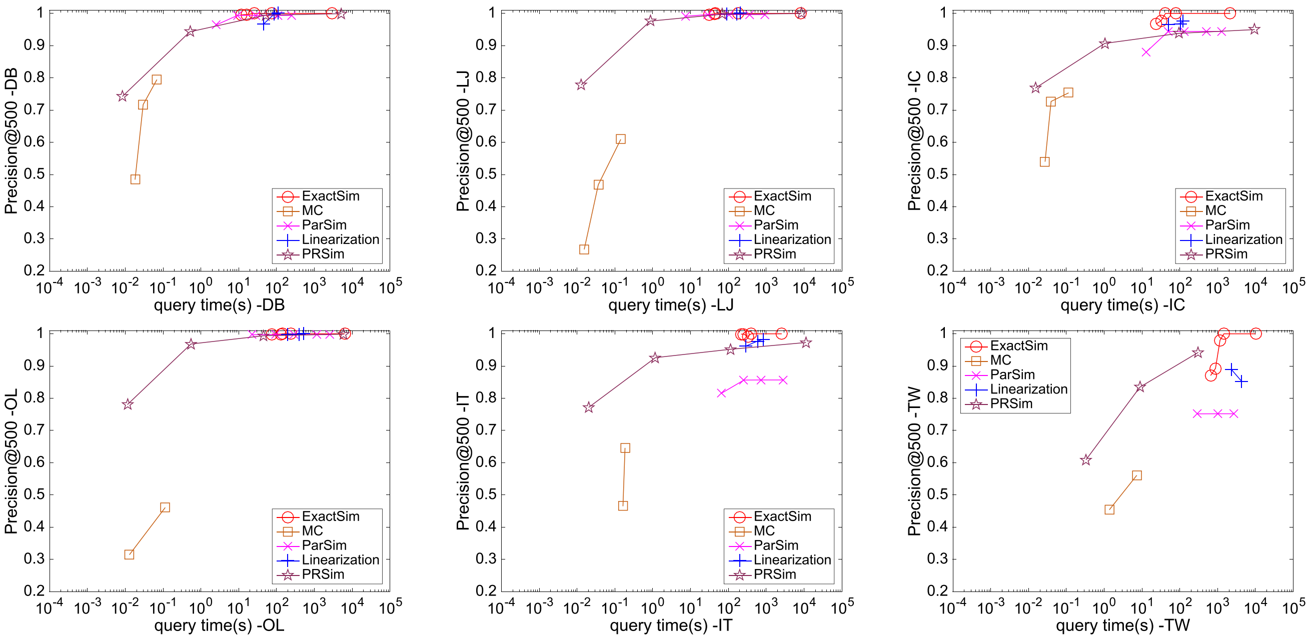


Fig. 6 Precision@500 v.s. Query time on large graphs

size plots of the index-based algorithms. For ExactSim with $\varepsilon = 10^{-7}$, we set its $MaxError$ as 10^{-7} and *Precision@500* as 1. We observe from Fig. 6 that ExactSim with $\varepsilon = 10^{6}$ also achieves a precision of 1 on all four graphs. This suggests that the top-500 results of ExactSim with $\varepsilon = 10^{-6}$ are the same as that of ExactSim with $\varepsilon = 10^{-7}$. In other words, the top-500 results of ExactSim actually *converge* after $\varepsilon = 10^{-6}$. This is another strong evidence of the exact nature of ExactSim. From Fig. 5, we also observe that ExactSim is the only algorithm that achieves an error of less than 10^{-6} on all six large graphs. In particular, on the TW dataset, no existing algorithm can achieve an error of less than 10^{-4}, while ExactSim is able to achieve exactness within 10^{4} seconds.

Ablation study We now evaluate the effectiveness of the optimization techniques. Recall that we use sampling according to $\boldsymbol{\pi}_i(k)^2$ and local deterministic exploitation to reduce the query time and sparse Linearization to reduce the space over-

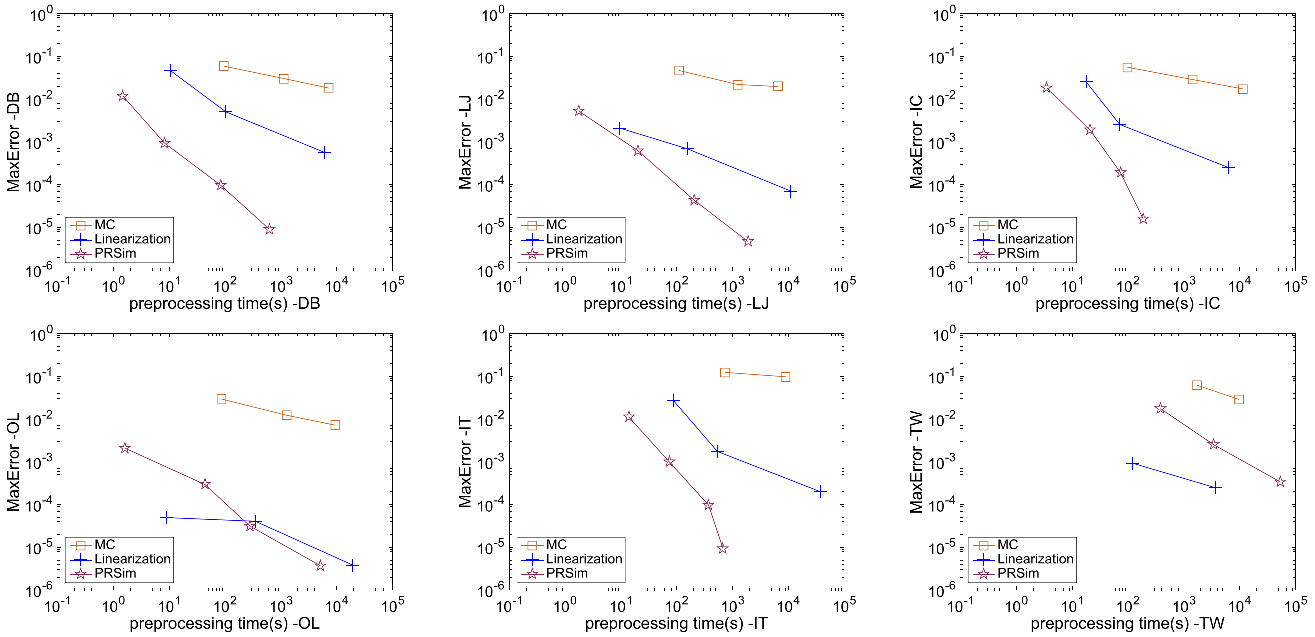


Fig. 7 MaxError v.s. Preprocessing time on large graphs

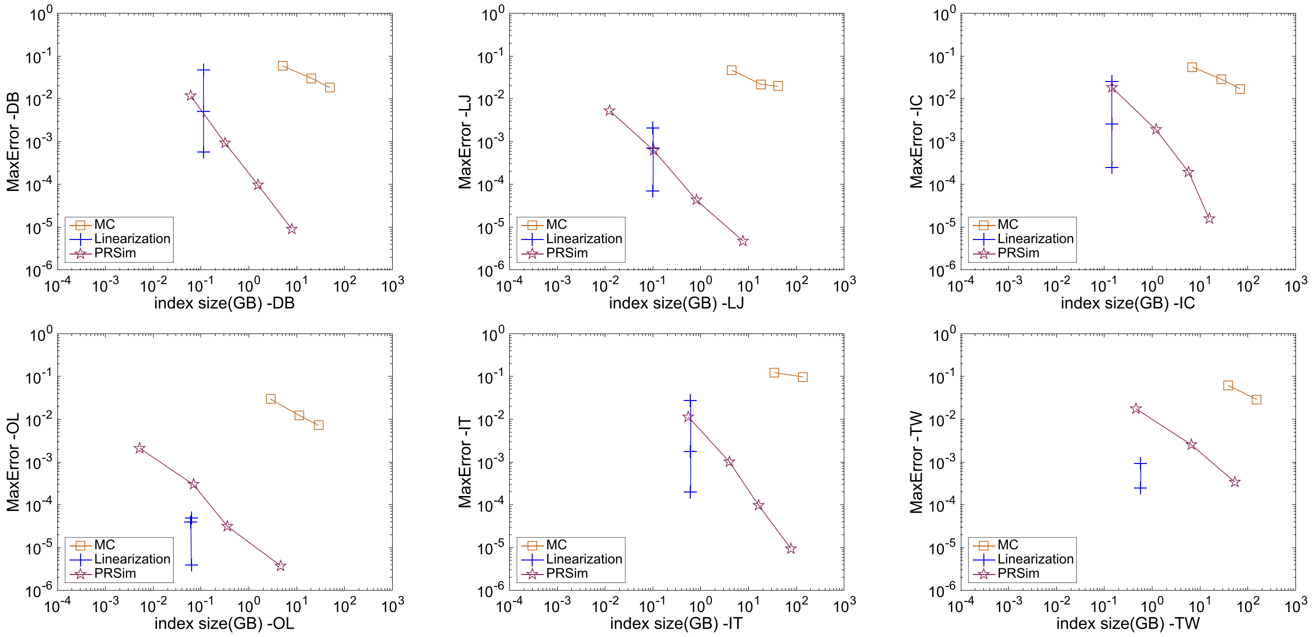


Fig. 8 MaxError v.s. Index size on large graphs

head. Figure 9 shows the time/error trade-offs of the basic ExactSim and the optimized ExactSim algorithms. Under similar actual error, we observe a speedup of 10 – 100 times. Table 6 shows the memory overhead of the basic ExactSim and the optimized ExactSim algorithms. We observe that the space overhead of the basic ExactSim algorithm is usually larger than the graph size, while sparse Linearization reduces the memory usage by a factor of 3 — 5 times. This demonstrates the effectiveness of our optimizing techniques.

5.2 Benchmarking approximate SimRank algorithms

We have proved the effectiveness of ExactSim on both small and large graphs against the state-of-the-art methods in each category. In the following, we will use the ground truths computed by ExactSim to evaluate the performances of existing single-source SimRank algorithms. To the best of our knowl-

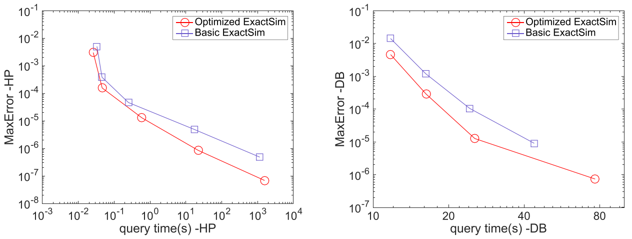


Fig. 9 Basic ExactSim v.s. Optimized ExactSim

Table 6 Memory overhead on large graphs

Memory overhead (GB)	*DB*	*IC*	*IT*	*TW*
Basic ExactSim	2.49	3.40	18.95	19.12
Optimized ExactSim	0.47	0.58	3.26	3.54
Graph size (GB)	0.48	1.88	10.94	13.30

edge, this is the first experimental study on the accuracy/cost trade-offs of SimRank algorithms on large graphs.

Methods. Recall that in Sect. 2, we present a detailed analysis about all existing single-source SimRank algorithms which can support large graphs. Because Uniwalk only supports undirected graphs, we omit it methods in our evaluation and consider the other nine single-source algorithms, including three Monte Carlo methods: MC [6], READS [11], and TSF [29], two iterative methods: Linearization [26] and ParSim [42], and four Local push/sampling methods: ProbeSim [21], PRSim [36], SLING [31], and TopSim [14]. Among them, ProbeSim and ParSim are index-free methods, and the others are index-based methods; READS, TSF, ProbeSim, TopSim, and ParSim can handle *dynamic* graphs, and the other methods can only handle *static* graphs. For the fairness of evaluation, we conduct each method in the single thread mode.

Experiments on Real-World Graphs We first evaluate the performance of each method on real-world graphs. The parameters of MC, ParSim, Linearization, and PRSim are the same as that in Sect. 5.1. Besides, READS has two parameters: the length of each random walk L and the number of random walks per node r. To cope with its better optimization, we vary (L, r) in larger ranges, from $(10^2, 10^3)$ to $(10^6, 10^7)$ on small graphs and from (10, 100) to (500, 5000) on large graphs. TSF has three parameters R_g, R_q, and T, where R_g is the number of one-way graphs, R_q is the number of samples at query time, and T is the number of iterations/steps. We vary (R_g, R_q, T) from (100, 20, 10) to (10000, 2000, 1000) on small graphs and from (100, 20, 10) to (4000, 800, 400) on large graphs. TopSim has four parameters T, h, η, and H, which correspond to the maximum length of a random walk, the lower bound of the degree to identify a high degree node, the probability threshold to eliminate a path, and the size of priority pool, respectively. As advised in paper [14], we fix $1/h = 100$ and $\eta = 0.001$ and vary (T, H) from (3, 100) to $(20, 10^9)$ on small graphs and from (3, 100) to $(7, 10^6)$ on large graphs. ProbeSim and SLING share the same error parameter ε, and we vary ε from 10^{-1} to 10^{-7} (if possible) on both small and large graphs.

Figures 10, 11, 12, 13, 14, and 15 present the benchmarking studies of existing single-source algorithms against the ground truths. Specifically, Fig. 10 plots the trade-offs between query time and *MaxError*. Figure 11 shows the trade-off lines between query time and *AvgError@50*, where

$$AvgError@k = \frac{1}{k} \sum_{v_j \in V_k} \left| \hat{S}(i, j) - S(i, j) \right|,$$

where V_k denotes the set of approximate top-k nodes. Figure 12 draws the trade-off plots between query time and *Precision@500*. Figure 13 shows the relations between memory cost and *MaxError*. Besides, as for those index-based methods, Figs. 14 and 15 plot the trade-offs between preprocessing time/index size and *MaxError*, respectively.

From these experimental results, we can derive the following observations. First of all, PRSim generally provides the best overall performance among the index-based methods in terms of query-time/error trade-offs. This suggests that the local push/sampling approach is more suitable for large graphs. Secondly, the two recent dynamic methods, ProbeSim and READS, achieve similar accuracy on large graphs for the typical query time range ($<$ 10 seconds) of the approximate algorithms. However, ProbeSim is an index-free algorithm and thus has better scalability. In particular, READS runs out of memory on the TW dataset with the number of samples per node $r > 1000$. Thirdly, ParSim is unable to achieve the same high precisions as it does on small graphs, which suggests that the $D \sim (1-c)I$ approximation is not as effective on large graphs. SLING and Linearization also quickly become unbearable on large graphs due to their $O\left(\frac{n \log n}{\varepsilon^2}\right)$ preprocessing time. This shows the necessity of evaluating the accuracy on large graphs. Finally, Fig. 13 shows iterative methods (ParSim and Linearization) perform the best in terms of space overhead.

Besides, we evaluate sensitivity of each method to the choice of k as for the *Precision@k*. Figure 16 shows the precision plots with varying k from 10 to 1000 on DB and TW datasets. For each method, we only pick one group of parameters to view the change of *Precision@k*. For fairness, we try to keep each method staying in the same level of precision by appropriate parameter settings. In detail, we set $L = 20$, $r = 200$ for MC; $L = 5$ for ParSim; $L = 100$, $r = 10$ for READS; $R_g = 100$, $R_q = 20$, $T = 10$ for TSF;

$T = 4$ and $H = 1000$ for TopSim; $\varepsilon = 0.1$ for Linearization, PRSim, ProbeSim, and SLING. We observe that larger k always leads to low precisions. The only exception is ParSim on TW, which shows a slightly increment with larger k. This reflects that ParSim can maintain the relative order of top-k nodes well.

Experiments on Synthetic Datasets We also analyze the trade-off of each method with fixed parameters on synthetic datasets to vary network structures and sizes. For fairness, we choose the parameters to guarantee the accuracy of each method remains in the same level. In particular, we set $L = 50$ and $r = 500$ for MC; $L = 500$ for ParSim; $L = 10$ and $r = 100$ for READS; $R_g = 100$, $R_q = 20$, and $T = 10$ for TSF; $T = 3$ and $H = 100$ for TopSim; $\varepsilon = 0.1$ for Linearization, PRSim, ProbeSim, and SLING. On each dataset, we also generate 50 query nodes for each dataset. For each set of parameters and each method, we issue one query from

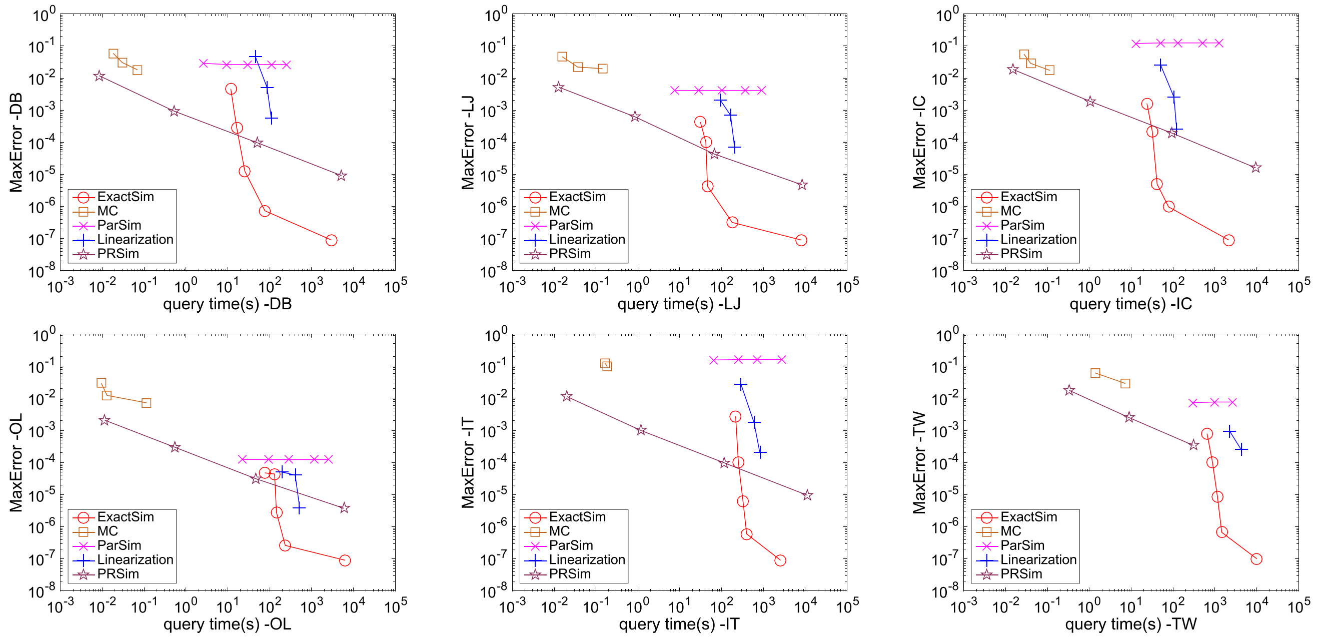


Fig. 10 Trade-offs: MaxError v.s. Query time on large graphs

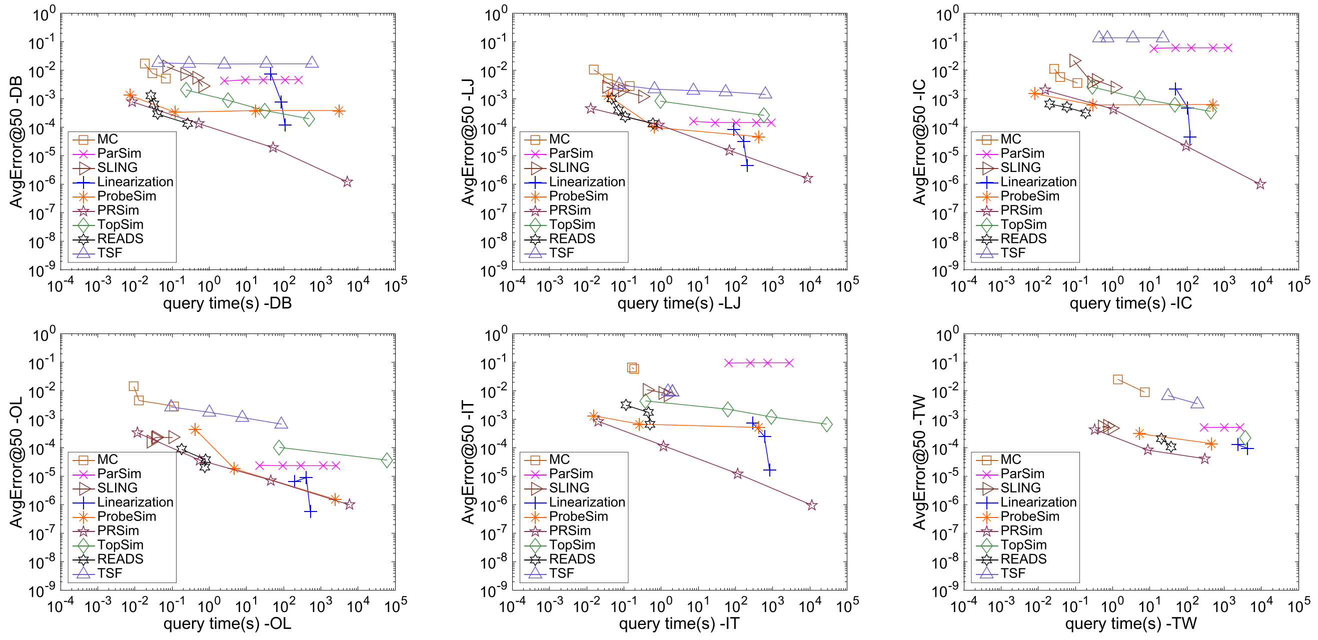


Fig. 11 Trade-offs: AvgError@50 v.s. Query time on large graphs

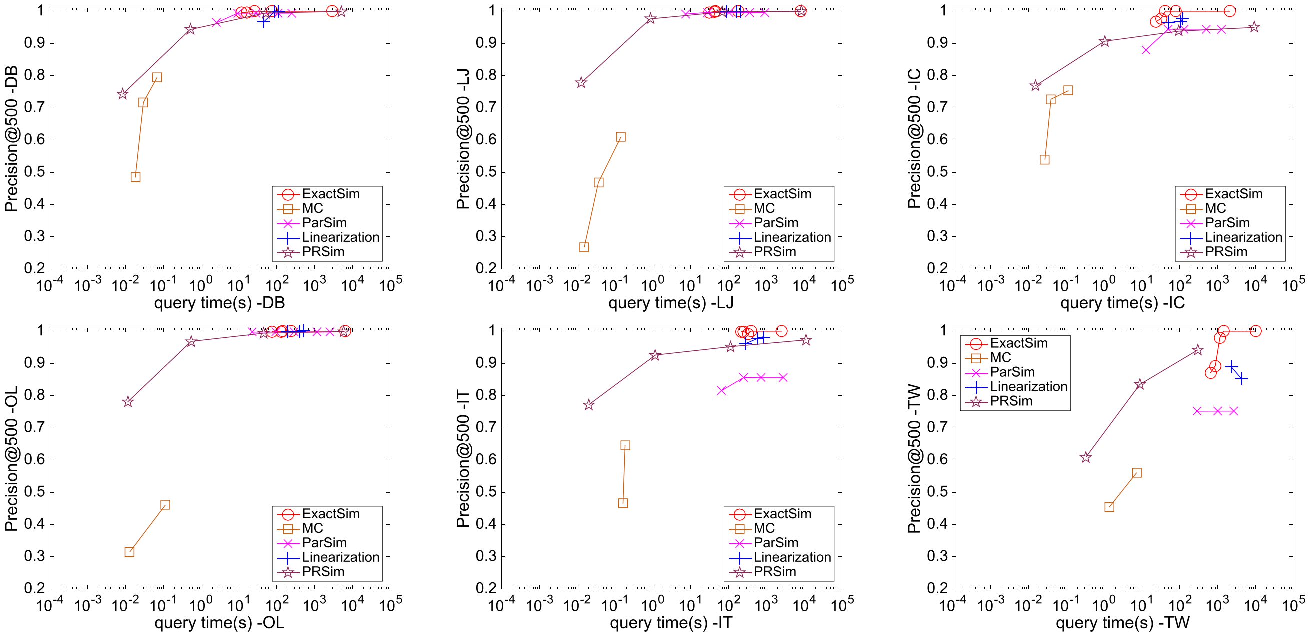


Fig. 12 Trade-offs: Precision@500 v.s. Query time on large graphs

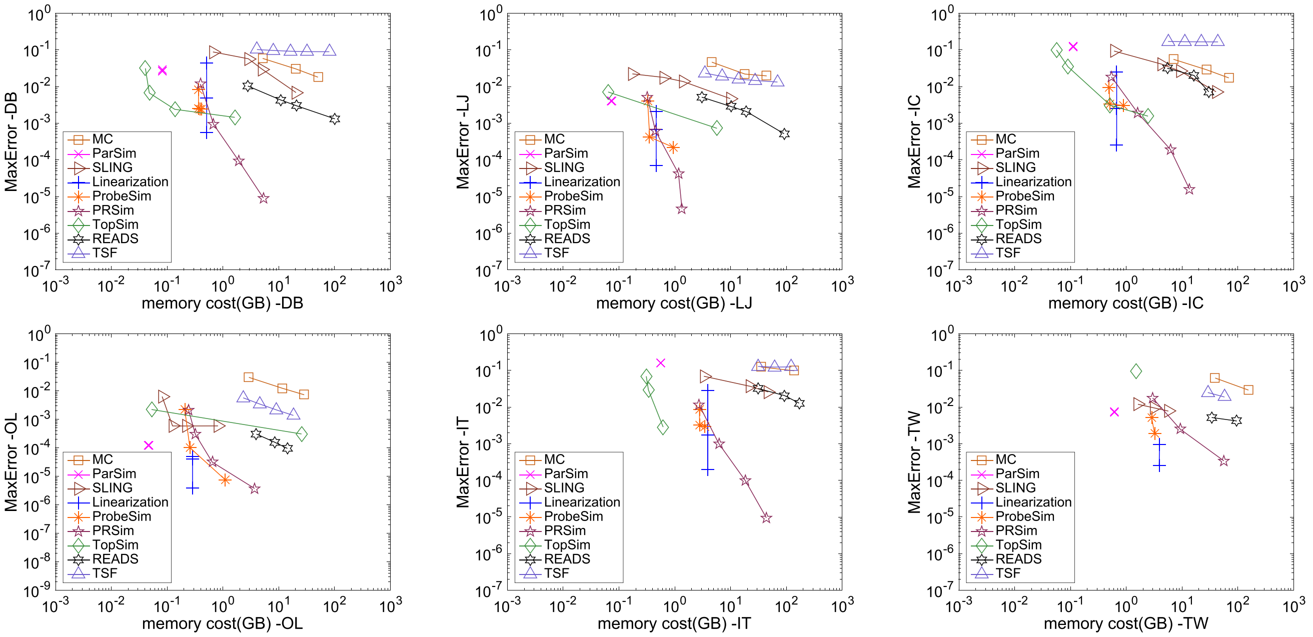


Fig. 13 Trade-offs: MaxError v.s. Memory Cost on large graphs

each query node and report the average *MaxError* and *Precision@500* among the 50 query nodes.

We first evaluate the performance of each method on power-law graphs. Using the hyperbolic graph generator given in [1,12], we generate a set of graphs with various power-law exponent γ, graph size n, and average degree $\bar{d}$. We fix the graph size $n = 100,000$ and the average degree $\bar{d} = 10$ and vary γ from 2.0 to 3.0. Figure 17a reports the query time of each γ. From Fig. 17a, we observe that the query time of most of methods increase with $1/\gamma$ except for Linearization, ParSim and SLING. For Linearization and ParSim, in the query phase, the two iterative methods repeat to do matrix multiplications with fixed times, leading to the unchanged query time. As for SLING, it heavily relies on the index and its query time with large ε is too short to be impacted by γ. In Fig. 17b, we fix $\gamma = 3$ and $\bar{d} = 10$ and vary n from 10^4 to 10^7 to evaluate the trade-offs between query time and the graph size n. We observe that

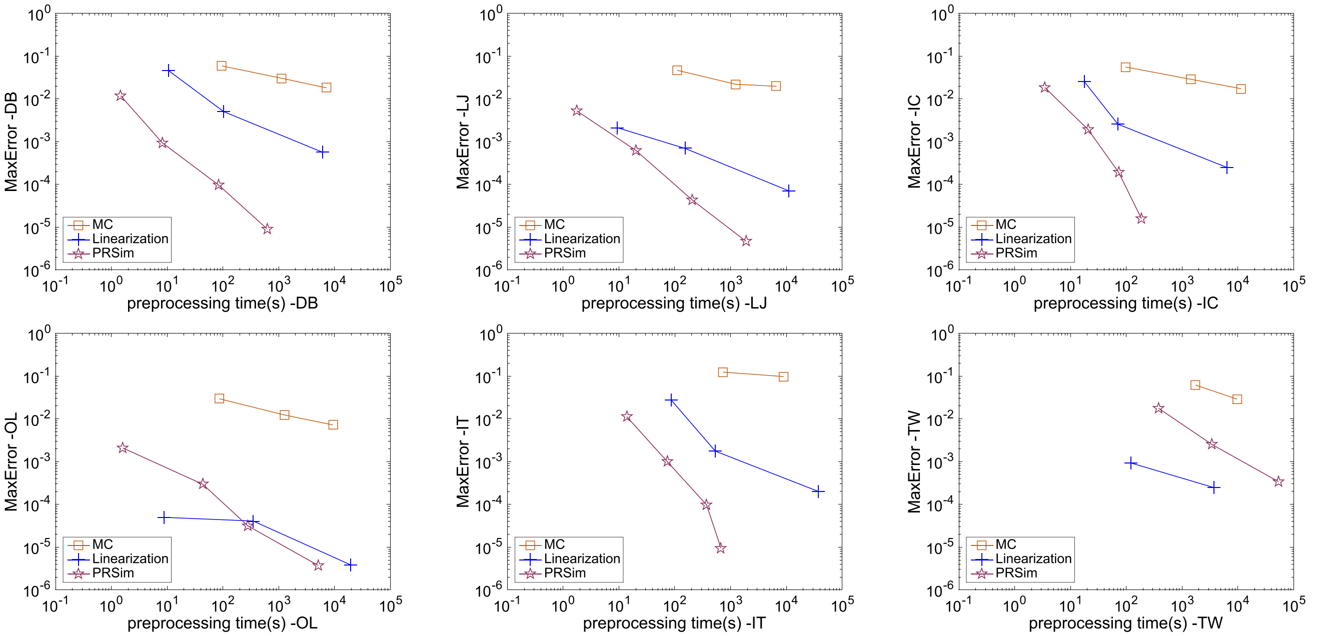


Fig. 14 Trade-offs: MaxError v.s. Preprocessing time on large graphs

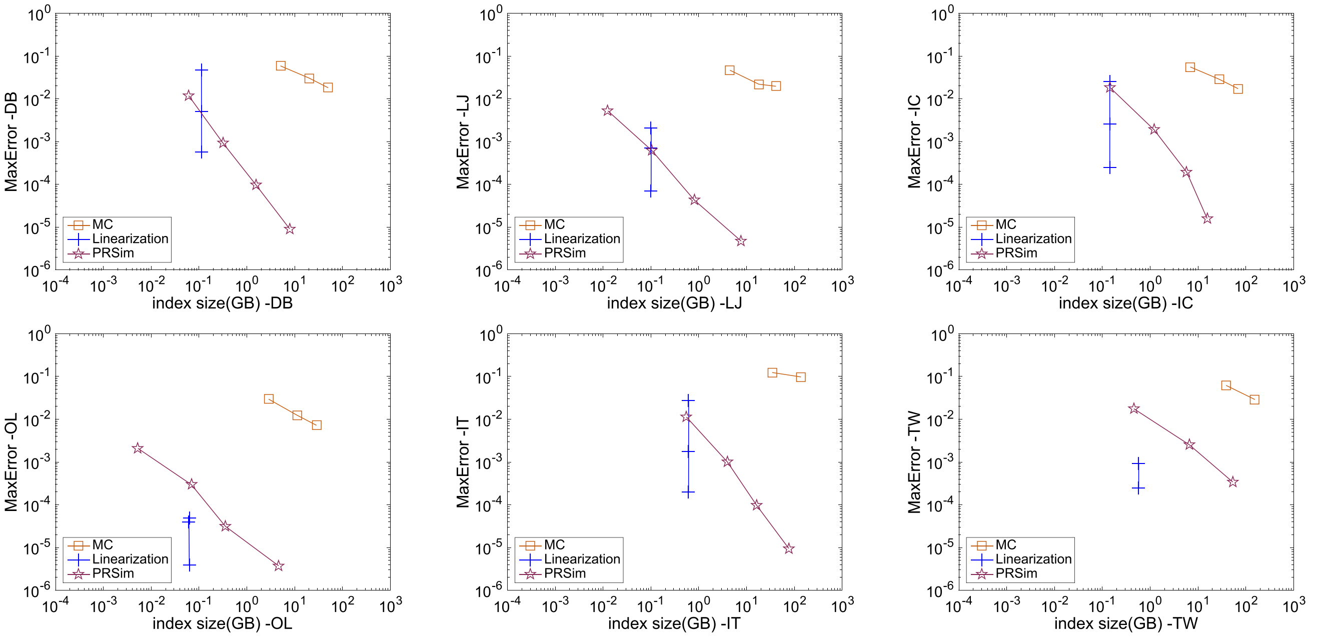


Fig. 15 Trade-offs: MaxError v.s. Index size on large graphs

local push/sampling methods' scalabilities outperform other methods in general. This is because these methods mainly focus on local information and are less influenced by the graph size. For Fig. 17c, we try to explore the performance of each method on the power-law graphs with different average degrees. Specifically, we fix $\gamma = 3$ and $n = 100,000$, and vary $\bar{d}$ from 5 to 1,000. We observe that the query time of PRSim increases at the slowest speed among these methods. This reveals the ability of PRSim to support dense graphs. On the contrary, TopSim shows a rapidly growing query time as the average degree increases.

Besides, we use Erdős and Rényi (ER) model to generate non-power-law graphs for evaluations. According to ER model, any pair of node will be assigned an edge with a specified probability p. In Fig. 18a, we vary the graph size n from 10^4 to 10^6. We adjust the probability p to fix the average degree $\bar{d} = 10$. In Fig. 18b, we vary $\bar{d}$ from 5 to 10^3 with fixed $n = 100,000$. Because by fixing the average

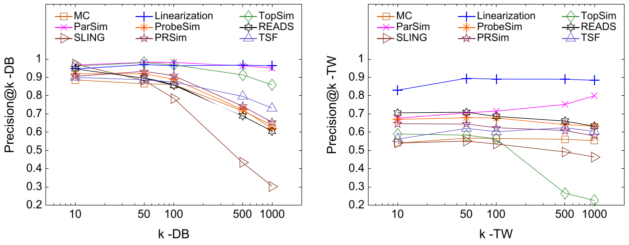
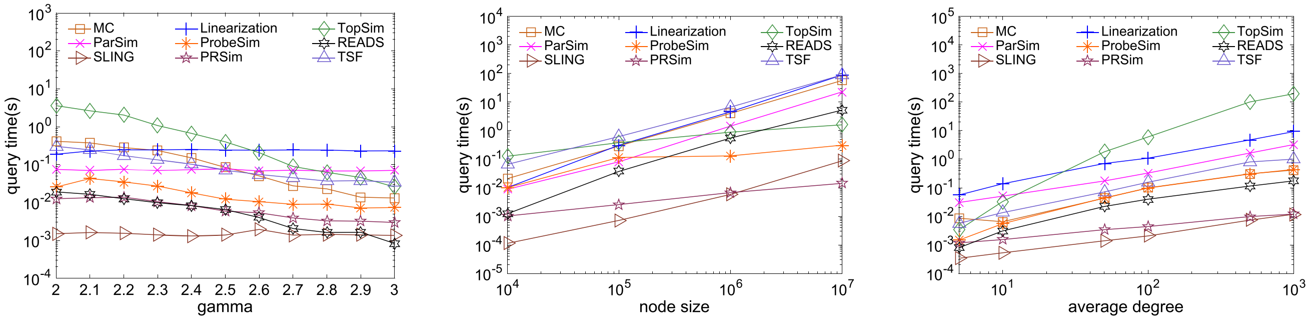


Fig. 16 Precision@k v.s. k

degree, the structures of ER graphs nearly remain unchanged with the increment of n. As shown in Fig. 18(a), the query time of MC-based methods (MC,READS and TSF) does not increase with n on the ER graphs. However, we observe that the query time of the three methods show obvious increments on power-law graphs. We attribute this difference to the existence of the hub nodes on power-law graphs.

Finally, we generate graphs using the stochastic block model with four parameters, including the graph size n, the number of clusters c, the probability p to assign an edge for any pair of node belonging to the same cluster, and the probability q to assign an edge for any two nodes belonging to different clusters. In Fig. 19a, we modulate the values of p and q to keep the average degree $\bar{d} = 10$ and the number of



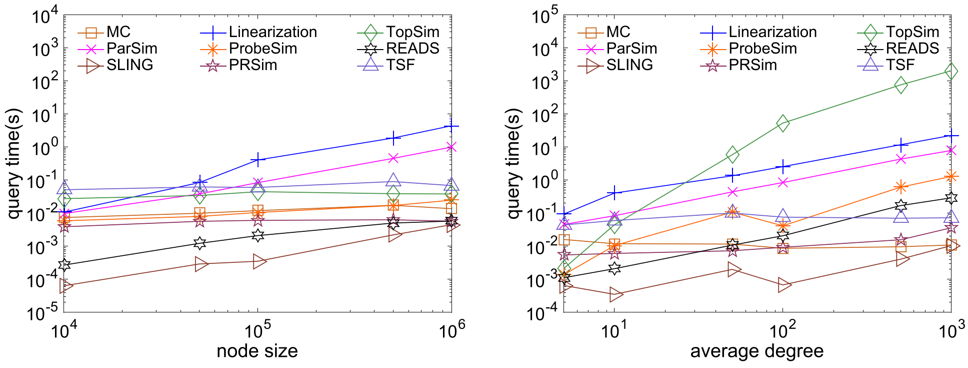
(a) *Query time* v.s. γ.

(b) *Query time* v.s. n.

(c) *Query time* v.s. $\bar{d}$.

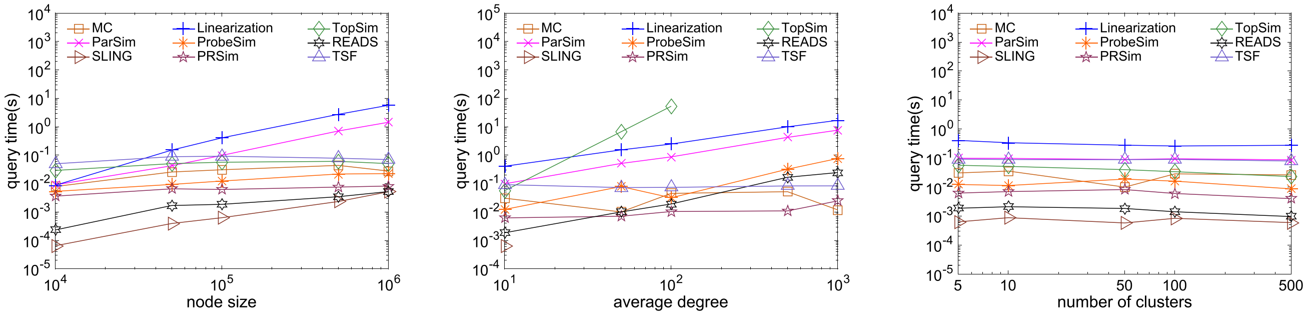
Fig. 17 Results on power-law graphs

Fig. 18 Results on non-power-law graphs



(a) *Query time* v.s. n.

(b) *Query time* v.s. $\bar{d}$.



(a) *Query time* v.s. n.

(b) *Query time* v.s. $\bar{d}$.

(c) *Query time* v.s. *clusters*.

Fig. 19 Results on stochastic block graphs

clusters $c = 5$ and vary the graph size n from 10^4 to 10^6. In Fig. 19b, we fix $n = 10^5$ and $c = 5$ and adjust p and q to vary the average degree $\bar{d}$ from 10 to 1000. In Fig. 19c, we vary the number of clusters c from 5 to 500 and fix $n = 10^5$, $\bar{d} = 10$. We observe that the result of each method is similar with that on ER graphs, which reflects that stochastic block model is a generalized version of ER model. Figure 19c shows that the number of clusters does not has a significant effect on the query time of these methods.

Experiments on Dynamic Datasets. In this section, we evaluate the performances of the methods which can support dynamic graphs. Recall that ParSim [42], ProbeSim [21], and TopSim [14] are index-free methods and can support dynamic graphs naturally. READS [11] and TSF [29] are two index-based methods which can support dynamic graphs by modifying index structures. Since the vertex modification can be treated as several edge modifications, we use the two dynamic graphs WD and WP which only contains edge modifications for ease of readability. The parameters of each method are the same with that in Sect. 5.2. For the four index-free methods, we run them on the final graphs of WP and WD. For READS and TSF, we first load the initial graph without the last 10,000 edge modifications and construct the index. Then, we run the two methods on the dynamic graphs with 10,000 edge modifications. After the updating process, we compare the computational quality of the six methods and plot their trade-offs between the query time and MaxError/Precision@500 in Fig. 20 and Fig. 21, respectively. In Fig. 20, we observe that each method's performance is similar with that on static graphs. ProbeSim achieves the highest approximation quality within the same query time. We observe that the performances of index-free methods are similar with that on static graphs. ProbeSim still shows the best performance among these methods. However, the *MaxError* of READS is hard to be reduced with increasing query time. This is very different from what we have observed on static graphs, where READS and ProbeSim achieve similar accuracy. In Fig. 22, 23, and 24, we plot the trade-offs between MaxError and preprocessing time/index size/updating time of the two index-based methods TSF and READS. Note that the updating time is the average time per edge insertion/deletion in the updating process. We observe that the two methods both incur large maximum error. READS shows a better performance than TSF.

5.3 SimRank distribution

We now design experiments to seek the answers for two open questions regarding the distribution of SimRank:

- Does the single-source SimRank result follow the power-law distribution on real-world graphs?

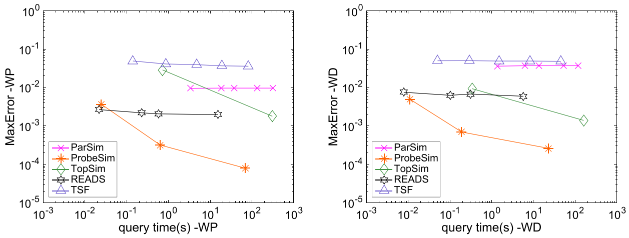


Fig. 20 MaxError v.s. Query time on dynamic graphs

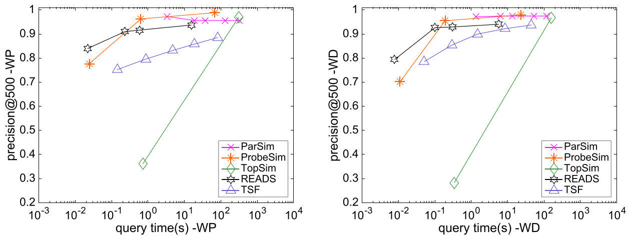


Fig. 21 Precision@500 v.s. Query time on dynamic graphs

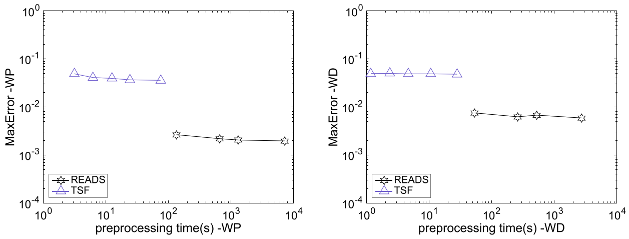


Fig. 22 MaxError v.s. Preprocessing time on dynamic graphs

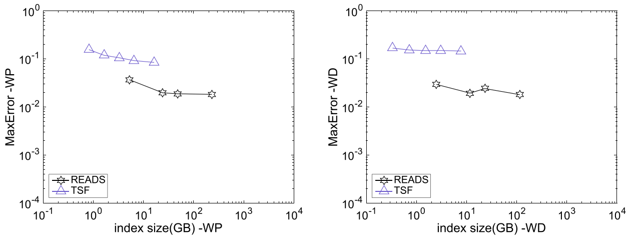


Fig. 23 MaxError v.s. Index size on dynamic graphs

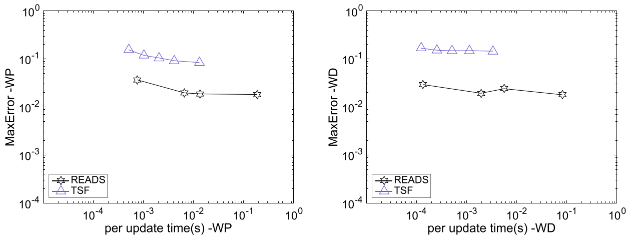


Fig. 24 MaxError v.s. Update Time per edge insertion/deletion on dynamic graphs

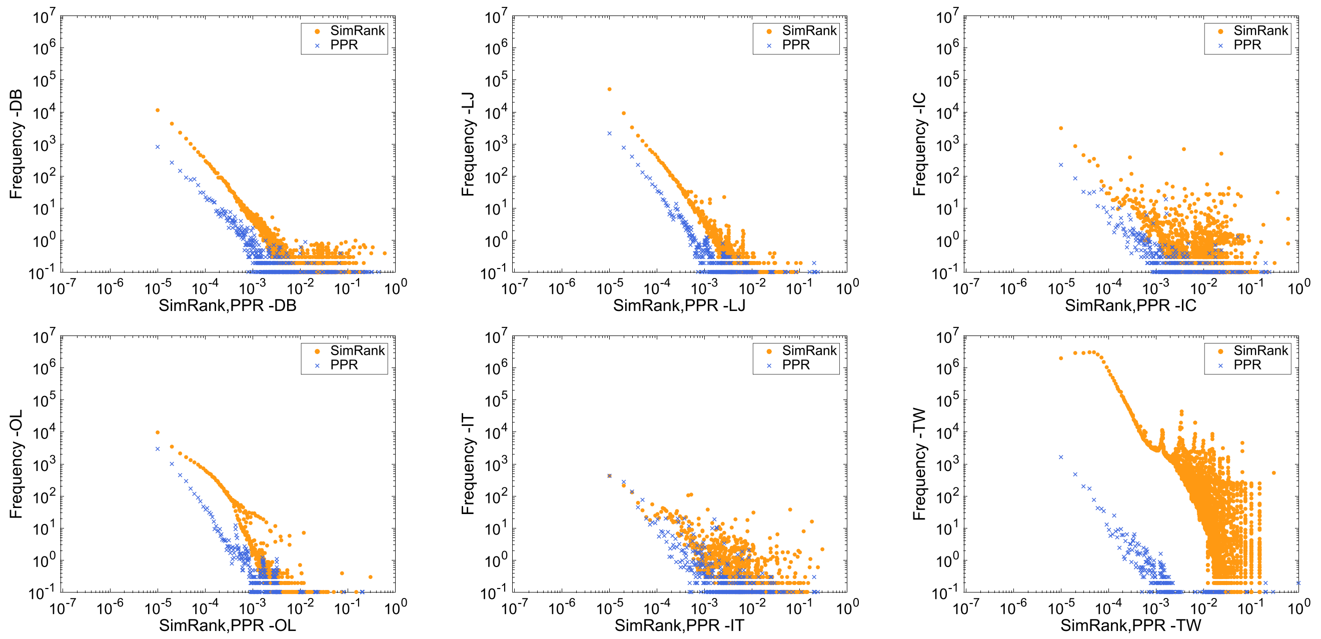


Fig. 25 The distribution of SimRank and PPR on real-world graphs

- What is the density of SimRank values on real-world graphs?

We use ExactSim to compute the ground truths of 50 random single-source queries on each of the six large graphs. Then, we compute the average frequency of SimRank values in every range of length 10^{-5} and plot these frequencies against the SimRank values in Fig. 25. Besides, we plot the frequency distribution of Personalized PageRank (PPR) computed by its Power Method [28] with teleport probability $\alpha = 0.2$, which has been proved following the power law [20]. The results suggest SimRank values indeed exhibit a power-law shaped distribution on real-world graphs as PPR does. In particular, the power-law exponent (slope) on TW appears to be significantly more skewed than that on IT, which explains why TW is a harder dataset for computing single-source SimRank queries. For sake of completeness, we also plot the degree distributions of the six graphs in Fig. 26. We compute the average frequency of each degree in every range of length 10. We observe that the largest degree can achieve 10^6 on TW, which is apparently larger than other datasets. This also demonstrates the hardness to compute SimRank on TW.

Besides, we plot the SimRank distribution on synthetic power-law graphs in Fig. 27 using the Kronecker graph model [15], which can generate large graphs of million nodes. We fix the probability seed matrix as (0.9, 0.5; 0.5, 0.1) and vary the graph size n from 10^6 to 5×10^7. On the four synthetic graphs, SimRank values still exhibit a power-law shaped distribution. We also plot the degree distribution of the four synthetic power-law graphs in Fig. 28. The degree distribution of the four synthetic graphs is all power-law shaped.

In comparison, we generate non-power-law graphs using the Erdős and Rényi(ER) model and show the SimRank distributions on the synthetic non-power-law graphs. According to the settings of ER-model, an edge is attached to each node with a user-defined probability p. We vary the number of nodes n from 10^4 to 5×10^5 and tune p to guarantee the average degree $d = 10$. Figure 29 plots the SimRank and PPR distributions. Figure 30 displays the degree distributions on these ER graphs. We observe that the distributions of SimRank and PPR both show non-power-law shaped curves on ER graphs.

Next, we analyze the density of single-source SimRank queries. The density of SimRank is the percentage of SimRank values that are larger than some threshold ε. Figure 31 shows the average density of 50 queries on six large datasets, with ε varying from 0.1 to 10^{-7}. The result shows that the densities can vary widely on different datasets. For example, on the TW dataset, the density of SimRank values quickly reaches close to 1 for $\varepsilon < 10^{-4}$. On the other hand, the density on the IT dataset seems to converge on 10^{-4}. This suggests that density-sensitive methods such as [35] can achieve satisfying results on IT and may run out of memory on dense graphs such as TW. This result also implies that it is essentially hopeless to design an exact algorithm for all-pair queries on large real-world graphs, as the number of nonzero entries in the SimRank matrix can be as large as $O(n^2)$.

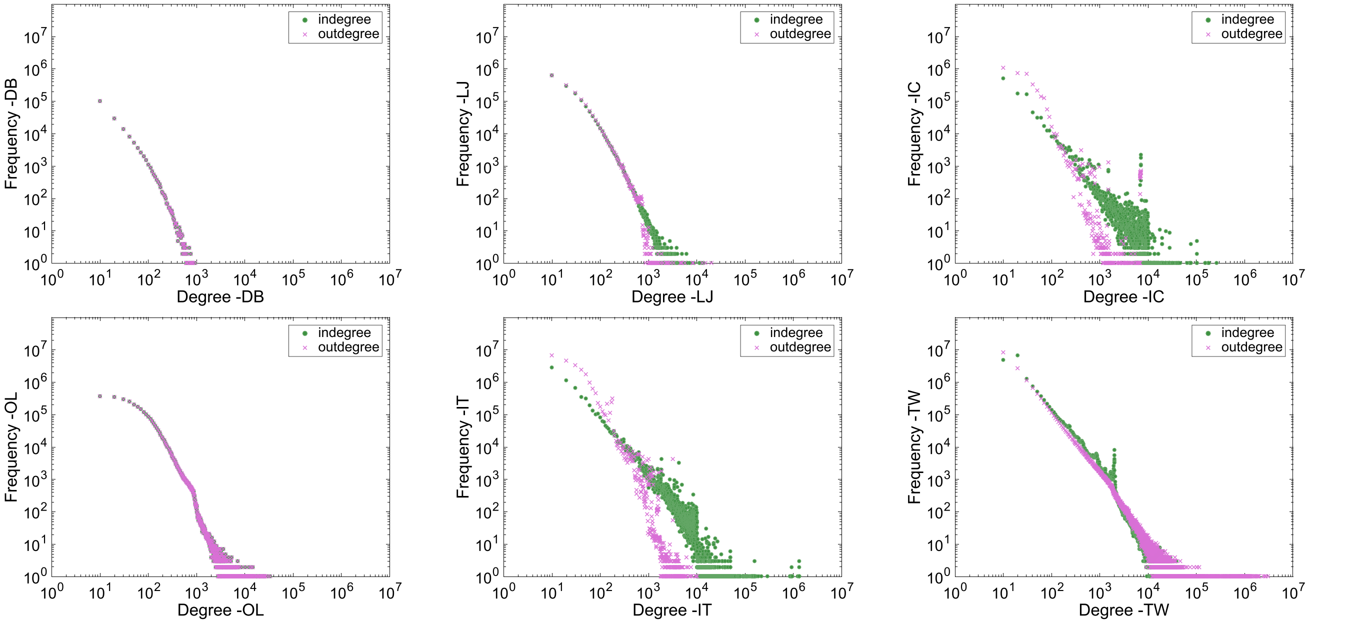


Fig. 26 Degree distribution of real-world graphs

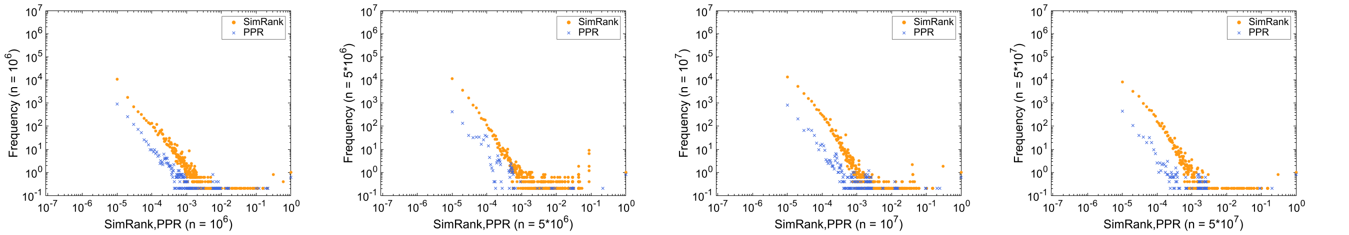


Fig. 27 SimRank and PPR distribution of the Kronecker graphs (varying n from 10^6 to $5 * 10^7$)

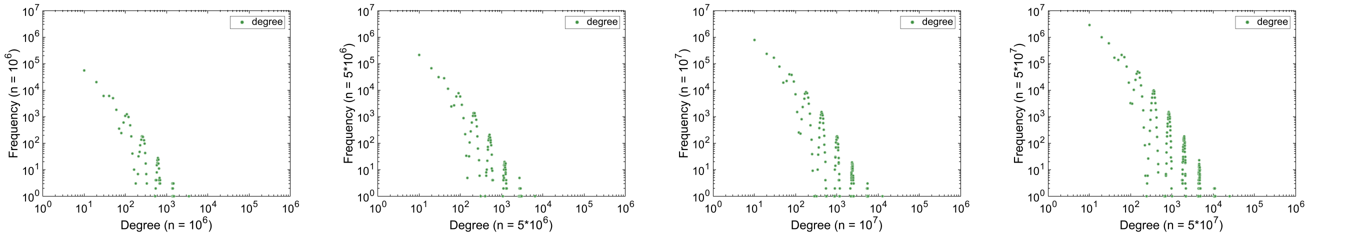


Fig. 28 Degree distribution of the Kronecker graphs (varying n from 10^6 to $5 * 10^7$)

6 Conclusions

This paper presents ExactSim, an algorithm that produces the ground truths for single-source and top-k SimRank queries with precision up to 7 decimal places on large graphs. Using the ground truths computed by ExactSim, we present the first experimental study of the accuracy/cost trade-offs of existing SimRank algorithms on large graphs. We also exploit various properties of the distributions of SimRank on large real-world graphs. For future work, we note that the $O\left(\frac{\log n}{\varepsilon^2}\right)$ complexity of ExactSim prevents it from achieving a precision of 10^{-14} (i.e., the precision of the double type). To achieve such extreme precision, we need an algorithm with $O\left(\frac{\log n}{\varepsilon}\right)$ complexity, which remains a major open problem in SimRank study.

Acknowledgements Zhewei Wei was supported by National Natural Science Foundation of China (NSFC) No. 61972401 and No. 61932001, by the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China under Grant 18XNLG21, and by Alibaba Group through Alibaba Innovative Research Program. The work is partially done at Beijing Key Laboratory of Big Data Management and Analysis Methods, MOE Key Lab DEKE, Renmin University of China, and Pazhou Lab, Guangzhou, 510330, China. Hanzhi Wang was supported by the Outstanding Innovative Talents Cultivation Funded Programs 2020 of Renmin University of China.

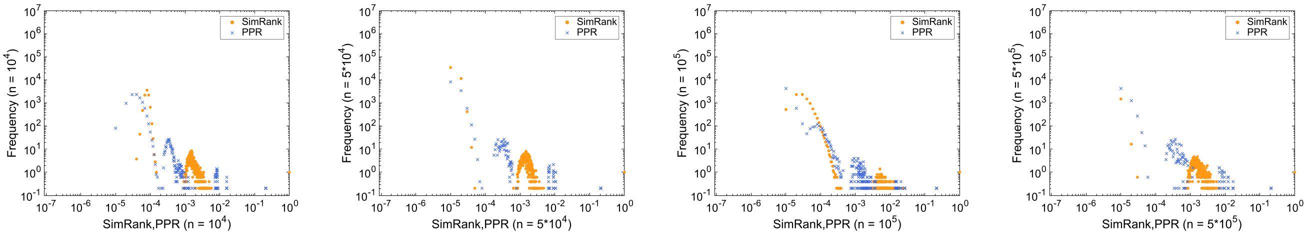


Fig. 29 The distribution of SimRank and PPR on E-R graphs (varying n from 10^4 to 5×10^5)

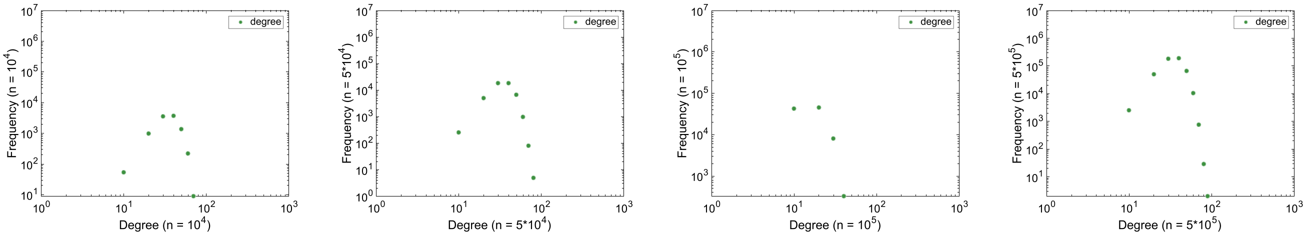


Fig. 30 Degree distribution on E-R graphs (varying n from 10^4 to 5×10^5)

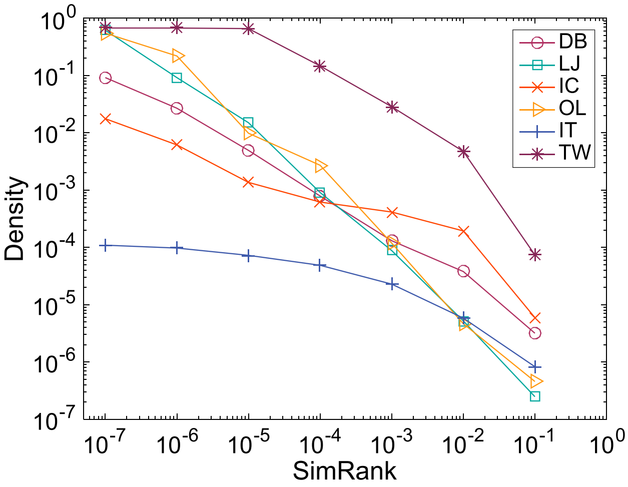


Fig. 31 SimRank density on large graphs

Ye Yuan was supported by NSFC No. 61932004 and No. 61622202 and by FRFCU No. N181605012. Ji-Rong Wen was supported by NSFC No. 61832017 and by Beijing Outstanding Young Scientist Program NO. BJJWZYJH012019100020098. Xiaoyong Du was supported by NSFC No. U1711261.

References

1. Aldecoa, Rodrigo, Orsini, Chiara, Krioukov, Dmitri: Hyperbolic graph generator. Computer Phys. Commun. **196**, 492–496 (2015)
2. Andersen, Reid., Chung, Fan R. K., Lang, Kevin J.: Local graph partitioning using pagerank vectors. In *FOCS*, pp. 475–486, (2006)
3. Antonellis, Ioannis, Molina, Hector Garcia, Chang, Chi Chao: Simrank++: query rewriting through link analysis of the click graph. PVLDB **1**(1), 408–421 (2008)
4. Bahmani, Bahman, Chowdhury, Abdur, Goel, Ashish: Fast incremental and personalized pagerank. VLDB **4**(3), 173–184 (2010)
5. Chung, Fan R.K., Lu, Lincoln: Concentration inequalities and martingale inequalities: a survey. Internet Math. **3**(1), 79–127 (2006)
6. Fogaras, Daniel., Racz, Balazs.: Scaling link-based similarity search. In: *WWW*, pp. 641–650, (2005)
7. Fogaras, Dániel, Rácz, Balázs, Csalogány, Károly, Sarlós, Tamás: Towards scaling fully personalized pagerank: algorithms, lower bounds, and experiments. Internet Math. **2**(3), 333–358 (2005)
8. Fujiwara, Yuichiro., Nakatsuji, Makoto., Shiokawa, Hiroaki., Onizuka, Makoto.: Efficient search algorithm for simrank. In: *ICDE*, pp. 589–600, (2013)
9. He, Guoming., Feng, Haijun., Li, Cuiping., Chen, Hong.: Parallel simrank computation on large graphs with iterative aggregation. In: *KDD*, pp. 543–552, (2010)
10. Jeh, G., Widom, J.: Simrank: a measure of structural-context similarity. In: *SIGKDD*, pp. 538–543, (2002)
11. Jiang, M., Fu, A.W.C., Wong, R.C.W.: Reads: a random walk approach for efficient and accurate dynamic simrank. PPVLDB **10**(9), 937–948 (2017)
12. Krioukov, Dmitri, Papadopoulos, Fragkiskos, Kitsak, Maksim, Vahdat, Amin, Boguná, Marián: Hyperbolic geometry of complex networks. Phys. Rev. E **82**(3), 036106 (2010)
13. Kusumoto, M., Maehara, T., Kawarabayashi, K-I.: Scalable similarity search for simrank. In: *SIGMOD*, pp. 325–336, (2014)
14. Lee, P., Lakshmanan, LVS., Yu, JX.: On top-k structural similarity search. In: *ICDE*, pp. 774–785, (2012)
15. Leskovec, J, Chakrabarti, D, Kleinberg, J, Faloutsos, C, Ghahramani, Z: Kronecker graphs: an approach to modeling networks. J. Mach. Learn. Res. **11**(2), (2010)
16. Li, C., Han, J., He, G., Jin, X., Sun, Y., Yu, Y., Wu, T.: Fast computation of simrank for static and dynamic information networks. In: *EDBT*, pp. 465–476, (2010)
17. Li, L., Li, C., Chen, H., Du, X.: Mapreduce-based simrank computation and its application in social recommender system. In: *2013 IEEE International Congress on Big Data*, pp. 133–140. IEEE, (2013)
18. Li, Zhenguo, Fang, Yixiang, Liu, Qin, Cheng, Jiefeng, Cheng, Reynold, Lui, John: Walking in the cloud: parallel simrank at scale. PVLDB **9**(1), 24–35 (2015)
19. Lin, Zhenjiang, Lyu, Michael R., King, Irwin: Matchsim: a novel similarity measure based on maximum neighborhood matching. KAIS **32**(1), 141–166 (2012)

20. Litvak, N., Scheinhardt, W.R.W., Volkovich, Y.: In-degree and pagerank: why do they follow similar power laws? Internet Math. **4**(2–3), 175–198 (2007)
21. Liu, Y., Zheng, B., He, X., Wei, Z., Xiao, X., Zheng, K., Jiaheng, L.: Probesim: scalable single-source and top-k simrank computations on dynamic graphs. PVLDB **11**(1), 14–26 (2017)
22. Lizorkin, D., Velikhov, P., Grinev, M., Turdakov, D.: Accuracy estimate and optimization techniques for simrank computation. VLDB J. **19**(1), 45–66 (2010)
23. Lizorkin, D., Velikhov, P., Grinev, M.N., Turdakov, D.: Accuracy estimate and optimization techniques for simrank computation. VLDB J. **19**(1), 45–66 (2010)
24. Lü, Linyuan, Zhou, Tao: Link prediction in complex networks: a survey. Phys. A: Stat. Mech. Appl. **390**(6), 1150–1170 (2011)
25. Luo, X., Gao, J., Zhou, C., Yu, J. X.: Uniwalk: Unidirectional random walk based scalable simrank computation over large graph. In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 325–336, (2017)
26. Maehara, T., Kusumoto, M., Kawarabayashi, K.: Efficient simrank computation via linearization. *CoRR*, abs/1411.7228, (2014)
27. Maehara, T., Kusumoto, M., Kawarabayashi, K.: Scalable simrank join algorithm. In: *ICDE*, pp. 603–614, (2015)
28. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: bringing order to the web. (1999)
29. Shao, Y., Cui, B., Chen, L., Liu, M., Xie, X.: An efficient similarity search framework for simrank over large dynamic graphs. PVLDB **8**(8), 838–849 (2015)
30. Tao, W., Minghe, Y., Li, G.: Efficient top-k simrank-based similarity join. PVLDB **8**(3), 317–328 (2014)
31. Tian, B., Xiao, X.: SLING: a near-optimal index structure for simrank. In: *SIGMOD*, pp. 1859–1874, (2016)
32. Tsitsulin, A., Mottin, D., Karras, P., Müller, E.: Verse: Versatile graph embeddings from similarity measures. In: *WWW*, pp. 539–548. International World Wide Web Conferences Steering Committee, (2018)
33. Wang, H., Wei, Z., Yuan, Y., Du, X., Wen, J.: Exact single-source simrank computation on large graphs. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 653–663, (2020)
34. Wang, Y, Che, Y, Lian, X, Chen, L, Luo, Q: Fast and accurate simrank computation via forward local push and its parallelization. In: IEEE Transactions on Knowledge and Data Engineering (2020)
35. Wang, Y., Chen, L., Che, Y., Luo, Q.: Accelerating pairwise simrank estimation over static and dynamic graphs. VLDB J. **28**(1), 99–122 (2019)
36. Wei, Z., He, X., Xiao, X., Wang, S., Liu, Y., Du, X., Wen, J.: Prsim: sublinear time simrank computation on large power-law graphs. In: *SIGMOD*, pp. 1042–1059. ACM, (2019)
37. Xi, W., Fox, EA., Fan, W., Zhang, B., Chen, Z., Yan, J., Zhuang, D.: Simfusion: measuring similarity using unified relationship matrix. In: *SIGIR*, pp. 130–137. ACM, (2005)
38. Yu, W., Lin, X., Zhang, W.: Fast incremental simrank on link-evolving graphs. In: *ICDE*, pp. 304–315, (2014)
39. Weiren, Y., Lin, X., Zhang, W., Chang, L., Pei, J.: More is simpler: effectively and efficiently assessing node-pair similarities based on hyperlinks. PVLDB **7**(1), 13–24 (2013)
40. Yu, W., McCann, J.: Gauging correct relative rankings for similarity search. In: *CIKM*, pp. 1791–1794, (2015)
41. Weiren, Y., McCann, J.A.: Efficient partial-pairs simrank search for large networks. PVLDB **8**(5), 569–580 (2015)
42. Yu, W., McCann, J.A.: Efficient partial-pairs simrank search on large networks. Proc. VLDB Endow. **8**(5), 569–580 (2015)
43. Yu, W., McCann, JA.: High quality graph-based similarity search. In: *SIGIR*, pp. 83–92, (2015)
44. Weiren, Y., Zhang, W., Lin, X., Zhang, Q., Le, J.: A space and time efficient algorithm for simrank computation. World Wide Web **15**(3), 327–353 (2012)
45. Zhang, J., Tang, J., Ma, C., Tong, H., Jing, Y., Li, J.: Panther: Fast top-k similarity search on large networks. In: *SIGKDD*, pp. 1445–1454. ACM, (2015)
46. Zhao, P., Han, J., Sun, Y.: P-rank: a comprehensive structural similarity measure over information networks. In: *CIKM*, pp. 553–562. ACM, (2009)
47. Zhao, P., Han, J., Sun, Y.: P-rank: a comprehensive structural similarity measure over information networks. In: *CIKM*, pp. 553–562, (2009)
48. Zheng, W., Zou, L., Feng, Y., Chen, L., Zhao, D.: Efficient simrank-based similarity join over large graphs. PVLDB **6**(7), 493–504 (2013)