CrashSim: An Efficient Algorithm for Computing SimRank over Static and Temporal Graphs

Mo Li[1,2], Farhana M. Choudhury[2], Renata Borovica-Gajic[2], Zhiqiong Wang[4], Junchang Xin[1,*], and Jianxin Li[3]

[1]School of Computer Science and Engineering, Northeastern University, China
[2]School of Computing and Information Systems, University of Melbourne, Australia
[3]School of Info Technology, Deakin University, Australia
[4]College of Medicine and Biological Information Engineering, Northeastern University, China

***Abstract*—SimRank is a significant metric to measure the similarity of nodes in graph data analysis. The problem of SimRank computation has been studied extensively, however there is no existing work that can provide one unified algorithm to support the SimRank computation both on static and temporal graphs. In this work, we first propose *CrashSim*, an *index-free* algorithm for single-source SimRank computation in static graphs. *CrashSim* can provide provable approximation guarantees for the computational results in an efficient way. In addition, as the real-life graphs are often represented as temporal graphs, *CrashSim* enables efficient computation of SimRank in temporal graphs. We formally define two typical SimRank queries in temporal graphs, and then solve them by developing an efficient algorithm based on CrashSim, called *CrashSim-T*. From the extensive experimental evaluation using five real-life and synthetic datasets, it can be seen that the *CrashSim* algorithm and *CrashSim-T* substantially improve the efficiency of the state-of-the-art SimRank algorithms by about 30%, while achieving the precision of the result set with about 97%.**

I. Introduction

SimRank has been studied extensively in the past years as a key technique to support graph data analytics [10], [18], [20], [25]. Simply put, SimRank is used to measure node-to-node similarity based on the topology of graphs. The assumption is that two nodes will be similar if they are both highly relevant to similar nodes, and each node is maximally similar to itself [7]. SimRank has broad adoption in many applications, since it can be applied not only to discover the similarity of nodes in terms of the graph structure, but also advance many real-life applications in graph data analytics, such as graph clustering [23], collaborative filtering [27], web mining [14].

Numerous efforts in the literature have focused on studying SimRank on static graphs. Jeh and Widom [7] were the first to propose the SimRank algorithm, which is to recursively return the SimRank value of all node-pairs in a given graph G with time complexity $O(m^2 \log \frac{1}{\varepsilon})$, where m is the number of edges and ε is the worst-case error. Fogaras *et al.* [4] developed a Monte-Carlo method that interprets the SimRank values as the expected function of the total time from the start of nodes u and v to the last encounter of two random walkers. To address the problem of the Top-k SimRank Query, Pei *et al.* [13] proposed two heuristic algorithms based on the truncated random walk strategy and prioritized the propagation respectively. A recent approach to SimRank computation is SLING [18], where the authors designed an efficient index structure for SimRank computation with at most ε additive error, also giving another interpretation of SimRank based on $\sqrt{c}$-walk. This approach is however not efficient since its index structure needs to be rebuilt from scratch whenever the input graph is updated, and its index construction requires several hours even on medium-size graphs with 1 million nodes [10]. The state-of-the-art approach to SimRank computation on static graphs with no index is ProbeSim [10]– an approach shown to considerably outperform existing solutions in terms of efficiency and scalability.

In addition, SimRank computation has also been studied in dynamic graphs by considering the updates of edges and/or nodes. Li *et al.* [9] provided another formulation of SimRank by using Kronecker product and vectorization operators that can support static and dynamic SimRank computations. Yu *et al.* [25] proposed a fast incremental algorithm to compute all-pairs SimRank by defining the SimRank update matrix of every link's changes, using an 'Affected Area' to skip unnecessary computation. Wong *et al.* [12] presented an index schema based on a random walk to compute SimRank over large dynamic graphs. However, all these studies did not fully explore the nature of time in the temporal graphs [2]. Temporal graphs can be expressed through a set of snapshots, where each snapshot contains the structure of the graph in a particular moment in time (i.e., a time instant). For example, in social networks, the interaction network formed among users changes throughout time as new interactions take place and active interactions become inactive with time [15]. Similarly, in DBLP networks, the cooperative relationship between authors are established and dissolved over time [1].

Example 1. *In a product recommendation system, given a user u, the item purchased by u might be recommended to other users who are similar to u. However, users' interests may change frequently. If we define a group of users where the similarities between these users and u are greater than a threshold θ continuously for a period of time, the items recommended for such a similar group based on u's interests will be more appropriate. At the same time, the similarity*

* Corresponding author (email: xinjunchang@mail.neu.edu.cn)

trend between users is also an important property. Sometimes, the similarity between the user u and a user v is high at the current time instant, but then their similarity reduces gradually. Thus, it may not be worthwhile to recommend u's items of interest to v.

The above example requires a new type of SimRank queries - *temporal* SimRank queries that focus on finding a node set that meets certain requirements such as *trend* or *threshold* on temporal graphs. To handle such queries, the most intuitive method is to adopt the existing SimRank algorithms proposed over static networks applied on each individual snapshot in order to compute the SimRank between the source u and any other node v during the entire query interval. In that way, the node set is updated at every time instant filtering out nodes that do not satisfy query requirements. Similarly, we can consider adjusting the SimRank computation algorithms for dynamic graphs to the temporal graphs, but even in such a case, we would need to re-compute the SimRank value every time an edge or a node gets updated. These straightforward methods are obviously not efficient when either the graph or the query interval is large.

Therefore, in this paper, we design an efficient and effective algorithm - CrashSim that can support the SimRank computation on both static and temporal graphs altogether. CrashSim can compute the approximate SimRank value with an error bound and is more efficient than the state-of-the-art algorithms for static graphs. Specifically, we adopt the computation of SimRank from [10] as computing the total probability that two random walks starting from u and v first meet at node u_i. The ProbeSim algorithm presented in [10] starts a $\sqrt{c}$-walk [18] (a variant of random walk where c is the decay factor for SimRank computation) from each u_i to find every node v that has a non-negligible probability to meet u_i. This process is repeated several times to guarantee an error bound. The ProbeSim algorithm is computationally expensive since: (i) the lengths of the walks can be quite long; and (ii) from each node the algorithm may traverse to any other node several times. To overcome these drawbacks, we truncate the length of the walk, while generating the $\sqrt{c}$-walk. The limited length and the number of iterations of the walk are set as a function of c such that the error bound remains the same as obtained by the ProbeSim algorithm. Moreover, with the help of the truncated $\sqrt{c}$-walk, we can compute a reverse reachable tree to generate a $\sqrt{c}$-walk starting from the source u before the iteration, instead of generating reverse reachable trees for every node in each iteration. As such, CrashSim substantially outperforms the state-of-the-art algorithms on static graphs.

Furthermore, we extend our proposed CrashSim algorithm to CrashSim-T to solve SimRank queries in temporal graphs, which has not been studied before. When applied over temporal graphs to answer temporal queries, CrashSim has a natural advantage since it supports partial SimRank computation, which means it does not need to compute all single-source SimRanks, because the candidate set for which the SimRank needs to be computed is gradually reduced. In addition, CrashSim applies two pruning strategies: (i) delta pruning which skips the unaffected area of all the Δ edges and (ii) difference pruning which ignores the unchanged nodes by comparing the related area of adjacent snapshots. The pruning strategies obviate the need to calculate SimRank for each snapshot, hence resulting in a further improvement over temporal graphs.

The contributions of this work are summarised as below:

- We introduce the novel CrashSim algorithm to compute the SimRank value. At first, we present the algorithm for static graphs in Section III. CrashSim uses the idea of the SimRank estimators as the average probability of two $\sqrt{c}$-walks with constraining length first-meeting.
- To answer temporal SimRank queries, CrashSim-T uses two pruning rules to reduce the redundant computations of SimRank on adjacent snapshots (Section IV).
- The extensive experiments over static and temporal datasets verify the effectiveness and efficiency of the proposed algorithms, demonstrating that CrashSim and CrashSim-T substantially outperform the state-of-the-art.

The rest of the paper is organized as follows. In Section II, we provide preliminaries and formally define Temporal SimRank Queries. After that, we design the CrashSim algorithm in Section III, and extend it in order to solve the Temporal SimRank Queries in Section IV. In Section V, we provide the experimental study based on the evaluation of five datasets and explain the results in detail. Finally, we discuss the related work in Section VI, and conclude the work in Section VII.

II. Preliminaries and Problem Definition

In this section, we first introduce the SimRank in temporal graphs, and then formally define the Temporal SimRank Queries. The frequently used notations are given in Table I.

A. SimRank

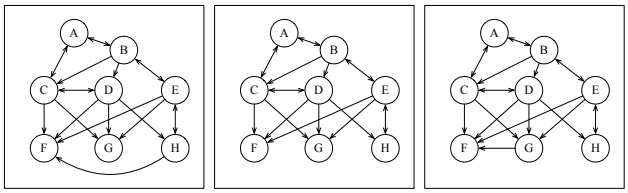
The main idea of SimRank is that two nodes are similar if they are related to similar nodes [7]. Specifically, given two nodes u and v in graph G, the SimRank value $sim(u, v)$ between them can be expressed as:

$$sim(u,v) = \begin{cases} 1, & \text{if } u = v, \\ \frac{c}{|I(u)||I(v)|} \sum_{x \in I(u), y \in I(v)} sim(x,y) & otherwise. \end{cases}$$

where $I(u)$ represents the set of neighbors (in-neighbor nodes for directed graph) of u, $|\cdot|$ denotes the number of elements in the set, and $c \in [0, 1]$ is a decay factor typically being set to 0.6 or 0.8.

As presented in [7], the intuition behind the similarity score produced by SimRank is to measure the probability that two random walks $W(u)$ and $W(v)$ first meet, starting from u and v, respectively. In [18], a new type of random walk, called $\sqrt{c}$-walk is proposed to compute an estimated SimRank value, $s'(u, v)$ as described below.

Definition 1 ($\sqrt{c}$-walk [18])**.** *Let c denotes the decay factor in the definition of SimRank, a $\sqrt{c}$-walk in G is defined such that:*



(a) T_1 (b) T_2 (c) T_3

Fig. 1: A Temporal Graph With 3 Snapshots

- *In each step of the random walk, we have* $1 - \sqrt{c}$ *probability to stop.*
- *For the remaining* $\sqrt{c}$ *probability, one of the in-neighbors of the current node is selected uniformly at random as the next step.*

According to the definition of $\sqrt{c}$-walk, Tian *et al.* [18] defined $s'(u, v)$ as the $\sqrt{c}$-walk $W(u)$ starting from u meets $\sqrt{c}$-walk $W(v)$ starting from v, which is defined as $s'(u, v) = \Pr[W(u)$ and $W(v)$ meet$]$. Liu *et al.* [10] proposed an index-free SimRank algorithm called *ProbeSim*, which further defines $s'(u, v)$ as the total probability that $\sqrt{c}$-walk $W(u)$ starting from u and $\sqrt{c}$-walk $W(v)$ starting from v first meet at each node u_i, i.e., $s'(u, v) = \Pr[W(u)$ and $W(v)$ meet$] = \sum_i \Pr[W(u)$ and $W(v)$ first meet at $u_i]$. Since [10] is the current state-of-the-art, we adopt their definition of $\sqrt{c}$-walk to compute SimRank in this paper.

TABLE I: Summary of Symbol Notations

Notation	Description
$\mathbf{G}(\mathbf{V}, \mathbf{E})$	input temporal graph
$G_t(V, E_t)$	snapshot of the temporal graph at time instant t
n, m	number of nodes and edges in G_t
$I(u)$	set of in-neighbors of u
c	decay factor in the definition of SimRank
ε	maximum error allowed in SimRank computation
δ	failure probability of a Monte Carlo algorithm
l_{max}	limited length of $\sqrt{c}$-walk
$sim(u, v)$	SimRank scores of source u and v in V
$s(u, v)$	SimRank estimator of u and v after truncating
$s'(u, v)$	SimRank estimator of u and v without truncating
$W(u)$	a reverse $\sqrt{c}$-walk starting from u
Ω	candidate node set that satisfies a certain query condition

B. Temporal Graphs

We now present the related definitions of SimRank for temporal graphs. A temporal graph can be expressed with a set of snapshots, where each snapshot is the mapping of the graph at each time instant. The formal definition of the temporal graph is presented as below.

Definition 2 (Temporal Graph)**.** *A temporal Graph* $\mathbf{G}$ *can be represented as a set of snapshots* $\mathbf{G}= \{G_1, G_2, ...G_T\}$*, where* $G_i = (V, E_i)$ *is the snapshot of pairwise interactions between the nodes at time* $i \in [1, T]$.

In other words, these temporal graphs only allow edge insertion/deletion over time. Figure 1 shows a temporal graph with 3 snapshots.

C. Temporal SimRank Queries

With the help of the definition of temporal graphs and SimRank, we define the Temporal SimRank Queries as follows.

Definition 3 (Temporal SimRank Queries)**.** *Given a temporal graph* $\mathbf{G}$*, a source* u*, a query interval* $[T_1, T_t]$*, a Temporal SimRank Query aims to find the node set* Ω*, such that the SimRank of* u *and each node* $v \in \Omega$*,* $s(u, v)$ *continuously meet a certain query requirement during the entire query interval* $[T_1, T_t]$.

Based on the real applications, we focus on the two most common queries over temporal graphs, namely temporal SimRank trend query and temporal SimRank thresholds query, formally defined as below.

Definition 4 (Temporal SimRank Trend Query)**.** *Given a temporal graph* $\mathbf{G}$*, a source* u*, a query interval* $[T_1, T_t]$*, a Temporal SimRank Trend Query aims to find the node set* Ω*, such that the SimRank of* u *and each node* v *in* Ω *is continuously increasing (or decreasing) during the entire query interval* $[T_1, T_t]$.

Definition 5 (Temporal SimRank Thresholds Query)**.** *Given a temporal graph* $\mathbf{G}$*, a source* u*, a query interval* $[T_1, T_t]$*, a Temporal SimRank Thresholds Query aims to find the node set* Ω*, such that the SimRank of* u *and each node* v *in* Ω *is greater than a given threshold* θ *during the entire query interval* $[T_1, T_t]$.

The SimRank queries in temporal graphs focus on the SimRank score sequence over a period of time, paying close attention to its temporal features. In contrast, the SimRank query algorithms over dynamic graphs continuously compute the SimRank score on every change, overlooking the temporal features.

D. Baseline

The ProbeSim algorithm is the state-of-the-art algorithm for computing the SimRank value over static graphs, and thus we regard it as our baseline. Since SimRank computation has not been explored in the context of temporal graphs in literature, we extend ProbeSim for temporal SimRank queries. In the following we first describe the key idea of the ProbeSim algorithm for static graphs, and then describe our modification of this algorithm to answer the temporal SimRank queries.

The ProbeSim algorithm traverses the entire graph from each node u_i to identify whether any node v has a non-negligible probability meeting $W(u_i)$. Specifically, the algorithm performs ${n_r}'$ iterations, and in the k-th trial the estimator ${s_k}'(u, v)$ is equal to the sum of the probabilities that v first meets the path $W(u, i)$ with different lengths starting from u, i.e., ${s_k}'(u, v) = \sum_{i=2}^{l} P(v, W(u, i))$. The ${n_r}'$ iterations are averaged to get the final $s'(u, v)$, i.e., $s'(u, v) = \frac{1}{{n_r}'} \sum_{k=1}^{n_r} {s_k}'(u, v)$. Further details of ProbeSim are presented in Section III-A. The ProbeSim algorithm is simple and does not use any index, but it requires generating a large

number of probing trees to identify whether $W(u)$ can meet every v at every step of $\sqrt{c}$-walk starting from u.

To solve the temporal SimRank query, we need to invoke the ProbeSim algorithm on each snapshot to calculate ${s_t}'(u, v_i)$ between u and each $v_i (v_i \in V)$ at each time instant t. Then, according to ${s_t}'(u, v_i)$ of each time instant and the different temporal SimRank query requirements, we will obtain the final node set by filtering out the nodes that do not meet the requirements for all time snapshots. During query processing, the number of nodes in the candidate set Ω that meet the requirement is gradually reduced. Thus, the straightforward extension of ProbeSim algorithm to solve the temporal SimRank queries will incur a repeated computational overhead due to changes in the adjacent snapshots in temporal graphs. The issue holds for both the expansion of the SimRank algorithms for static graphs, e.g. SLING algorithm [18] and the index-based SimRank algorithms over dynamic graphs, e.g. READS algorithm [12]. To address this problem, we propose two pruning rules over the already efficient CrashSim algorithm to reduce the redundant computations of SimRank on adjacent snapshots discussed next.

III. CrashSim Algorithm

In this section, we present our novel *CrashSim* algorithm, which is used to compute an approximate SimRank score of a node with a guaranteed error bound in a snapshot graph. Later in Section IV we present the extension of CrashSim for temporal graphs. Before describing the details of the algorithm, we first provide the motivation and main intuition behind CrashSim.

A. Motivation and Key Ideas

As mentioned in Section II-A, the Simrank score $s'(u, v)$ computed by the ProbeSim algorithm [10] is the total probability that two $\sqrt{c}$-walks, $W(u)$ and $W(v)$ starting from u and v, respectively first meet at each node u_i, i.e., $s'(u, v) = \sum_i \Pr[W(u) \text{ and } W(v) \text{ first meet at } u_i]$. In the ProbeSim algorithm, the graph traversal starts from each u_i to identify any node v that has a non-negligible probability to 'walk' to u_i, and this process is iterated n_r times to obtain results with approximate guarantees. Hence, the ProbeSim algorithm suffers from the following drawbacks: (i) there are many redundant computations during the traversal, since each node may traverse to any other node to identify the probability several times; and (ii) the length of $\sqrt{c}$-walk $W(u)$ and the length of $\sqrt{c}$-walk $W(v)$ determine the computation time of $s'(u, v)$, where these lengths can be quite large.

To overcome the drawbacks, we use the following key ideas when designing the CrashSim algorithm:

- We constrain the length of $\sqrt{c}$-walk from u to l_{max} instead of length $|W(u)|$. The value of l_{max} is set as a function of c for a guaranteed error bound (related proofs are provided in Section III-C).
- We compute a reverse reachable tree of source u with the limited length of $\sqrt{c}$-walk, l_{max}. We then iteratively generate a $\sqrt{c}$-walk for each node v to identify whether it can "hit" this limited $\sqrt{c}$-walk path from u with a non-negligible probability. Intuitively, this process will significantly reduce the redundant computation, since it traverses the reachable tree for only the source u, instead of traversing the graph for each node $u_i \in V \backslash u$.
- Although the walk length is constrained, we are still able to obtain SimRank estimators with the same guaranteed error bound of the ProbeSim algorithm by adding a constant multiple of the number of iterations (related proofs are provided in Section III-C).

In the following, we first describe the details of the CrashSim algorithm and present a running example to demonstrate the steps. We then prove the error bound guarantee and analyze the time complexity.

B. Algorithm Description

Given a snapshot graph G, a source $u \in V$, the candidate node set Ω, the maximum tolerable error ε, and the failure probability δ, the CrashSim algorithm returns approximate SimRank scores between u and each node $v \in \Omega$.

We design the CrashSim algorithm such that, for every node $v \in \Omega$, the difference between the returned SimRank and the actual SimRank is no more than ε, and this inequality holds for any v with at least $1 - \delta$ probability. This error bound is presented formally in the following:

Definition 6 (Approximation Guarantee)**.** *Given a source u in graph G, an absolute error threshold ε, and a failure probability δ, CrashSim returns a SimRank estimator $s(u, v)$ for each node v in G, such that,*

$$|s(u, v) - sim(u, v)| \leq \varepsilon$$

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

The pseudo-code of the algorithm is shown in Algorithm 1. The algorithm consists of the following steps.

- We initialize the SimRank estimators $s(u, v)$ with the value 0 for each $v \in \Omega \backslash u$.
- The value of the required limited length of $\sqrt{c}$-walk (i.e., l_{max}) is calculated according to Theorem 1 (see Section III-C) in Lines 1-2.
- We then invoke **revReach** algorithm (see Algorithm 2) to construct the reverse reachable tree for u. The **revReach** algorithm takes a graph $G(V, E)$, a source node $u \in V$, and a 'step length' (here, l_{max}) as parameters, and returns a matrix U, where each element $(step, v)$ is the probability of the $\sqrt{c}$-walk stopping at v with the length of step (Line 3), where the walk starts from source u.
- Next, we compute the minimum number of iterations, n_r that is needed to obtain the required error bound, according to Theorem 1 (Lines 4-5).
- After that, the algorithm runs n_r independent trials (Line 6). For the k-th iteration, it will generate a $\sqrt{c}$-walk for each $\nu \in \Omega$ to compute the total probability of such

Further in Section V, we compare our approach against both state-of-the-art algorithms for static, and dynamic graphs.

$\sqrt{c}$-walk hitting (crashing) the $\sqrt{c}$-walk starting from u (Lines 7-11). In details, for the node v_j, we first generate a $\sqrt{c}$-walk starting from the node and constrain the length of the walk to maximum l_{max} (Lines 8-9). Then, for the $\sqrt{c}$-walk with length i (where $i \in [2, l_{max}]$), the probability of this walk "hitting" (or "crashing") the $\sqrt{c}$-walk starting from u at the i-th element of the $W(v_j)$, i.e., $P(v_j, W(u,i))$ will be added to the SimRank estimator between u and v_j (Lines 10-11). In other words, this value is the total first-meeting probability of $\sqrt{c}$-walks starting from u and v_j with length i.

- After the n_r trials finish, for each node v, we take the average of n_r times SimRank estimators to compute the final SimRank estimators $s(u,v) = 1/n_r \sum_{k=1}^{n_r} s_k(u,v)$ (Lines 12-13). Finally, we return S as the SimRank estimators for each node $v \in \Omega$.

revReach Algorithm. We now describe the revReach algorithm that is used to generate the reverse reachable tree for the source u. Given a snapshot graph $G(V,E)$, the source u and the limited length 'step' of $\sqrt{c}$-walks, the revReach returns a matrix U, in which each element $(step, v)$ corresponds to the probability of the $\sqrt{c}$-walk starting from u and stopping at v with length of $step$.

The pseudo-code of the algorithm is illustrated in Algorithm 2. At first, we initialize U, which represents the probabilities of the $\sqrt{c}$-walk starting from u stopping at different nodes with different lengths, and set $U(0,u)$ to 1 indicating that the probability of the $\sqrt{c}$-walk stopping at u with no length is 1 (Line 1). Then a queue Q is initialized for recording the current node with different lengths, and the item $(0,u)$ is pushed into Q (Line 2). To record the parent node of the current node in the reverse reachable tree, another queue PR is initialized and the value -1 is pushed into it, since there is no parent node of the source u (Line 3). After that, the algorithm iteratively visits each element of Q (Line 4). During the process, it first pops up the top element (tl, tu) and tpr of Q and PR respectively (Lines 5-6). If the current length of $\sqrt{c}$-walk is larger than l_{max}, there is no need to continue computing the reverse reachable tree (Lines 7-8). Then for each in-degree node v of tu, we push the in-degree nodes of v into Q and push v into PR (Lines 10-11). After that, we set $U(tl+1, v)$ to $\sqrt{c}/I(v) U(tl, tu)$ (Line 12). Note that in this iteration, the in-degree of tu that is equal to the parent node of tu is ignored to avoid recomputing the probability due to the cycle in the graph (Line 9). Finally, we obtain the matrix U that represents the reverse reachable tree of u (Line 13).

Example 2 (A running example of the CrashSim algorithm)**.** *We use the graph shown in Fig. 2 and Fig. 3 to illustrate the steps of the CrashSim algorithm. For simplicity, we set the decay factor* $c = 0.25$ *so that* $\sqrt{c} = 0.5$*. The SimRank scores between A and any other nodes are listed in TABLE II, which are computed by the Power Method within* 10^{-5} *error.*

We first generate the reverse reachable tree U for the source A, which is shown in Fig. 3. At the same time, the probability of the $\sqrt{c}$*-walk stopping at different nodes with different lengths will be computed. In particular, the algorithm first inserts* $(1, B)$ *and* $(1, C)$ *to U, since they are in-neighbours of A. The probabilities of the* $\sqrt{c}$*-walk stopping at these node are* $U(1,B) = U(0,A) \cdot \frac{\sqrt{c}}{|I(B)|} = 1 \cdot \frac{0.5}{2} = 0.25$ *and* $U(1,C) = U(0,A) \cdot \frac{\sqrt{c}}{|I(C)|} = 1 \cdot \frac{0.5}{3} = 0.167$*. Similarity, the algorithm inserts* $(2,E)$*,* $(2,B)$*,* $(2,D)$ *with probabilities* 0.0625*,* 0.0417*, and* 0.0417 *into the reverse reachable tree* U*. For the next iteration, we insert* $(3,H)$*,* $(3,A)$*,* $(3,E)$*,* $(3,B)$ *from the in-neighbours of E, B, and D into the reverse reachable tree, and with the probabilities* 0.0156*,* 0.0104*,* 0.0104*, and* 0.0104*.*

When computing the SimRank scores between A and C, $s(A,C)$*, suppose that at the k-th trial, a reverse random walk starting from C, i.e.,* $W(C) = (C, D, B, A)$ *will meet the reverse random walk starting from A, i.e.,* $W(A)$*, and the probability of them "crashing" is* $s_k(A,C) = U(0,C) +$

Algorithm 1: CrashSim Algorithm

Input : $G(V,E)$, $u \in V$, Ω, ε, δ
Output: $S = \{s(u,v) | v \in \Omega\}$

1 Initialize $s(u,v)$, for each $v \in \Omega$
2 $l_{\max} \leftarrow \frac{1+\sqrt{c}}{(1-\sqrt{c})^2}$ (according to Theorem 1)
3 Set $U \leftarrow revReach(u, l_{max})$
4 $p \leftarrow \sum_{k=1}^{l_{\max}} (\sqrt{c})^{k-1}(1-\sqrt{c})$
5 $n_r \leftarrow \frac{3c}{(\varepsilon - p\varepsilon_t)^2} \log \frac{n}{\delta}$
6 **for** $k = 1$ *to* n_r **do**
7 **for** *each* $v_j \in \Omega$ **do**
8 Generate a $\sqrt{c}$-walk from v_j, $W(v_j) = (v_1, v_2, \ldots, v_l)$, where $v_l = v_j$
9 limit $l < l_{\max}$
10 **for** $i = 2, \ldots, l$ **do**
11 $s_k(u, v_j) += U(i-1, W_i(v_j))$
12 **for** *each* $v_j \in \Omega$ **do**
13 $s(u, v_j) = \frac{1}{n_r} s_k(u, v_j)$
14 **return** S;

Algorithm 2: revReach Algorithm

Input : $G(V,E)$, $u \in V$, $l_{\max}$
Output: U

1 Initialize U, and set $U(0,u) = 1$
2 Initialize a queue Q, and push $(0,u)$ into Q
3 Initialize a queue PR, and push -1 into PR
4 **while** Q *is not empty* **do**
5 (tl, tu) is the top element of Q, and pop it
6 tpr is the top element of PR, and pop it
7 **if** $tl > l_{max}$ **then**
8 break;
9 **for** *each* $v \in I(tu)$ ***and*** $v \neq tpr$ **do**
10 push indegree nodes of v into Q
11 push v into PR
12 $U(tl+1, v) = \frac{\sqrt{c}}{|I(v)|} U(tl, tu)$
13 **return** U;

TABLE II: SimRank Scores With Respect To Node A

	A	B	C	D	E	F	G	H
$s(A,*)$	1.0	0.0064	0.048	0.132	0.074	0.040	0.048	0.0064

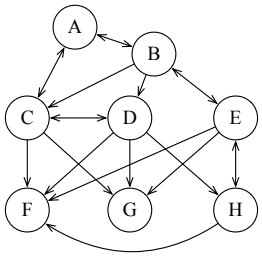


Fig. 2: Sample Graph

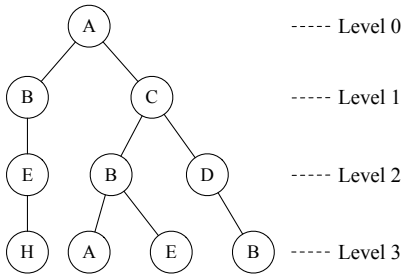


Fig. 3: Reverse Reachable Tree

$U(1,D)+U(2,B)+U(3,A)=0+0+0.0417+0.0104=0.0521$. *After n_r iterations, the CrashSim algorithm will get the SimRank estimator between A and C.*

C. Theoretical Analysis

Correctness and Error Bound Guarantee. We now show that CrashSim indeed gives an estimation $s(u,v)$ with provable error guarantees to the SimRank scores $sim(u,v)$ for each $v \in \Omega$. To prove the error guarantee, we use the definition of first-meeting probability presented in [10].

Definition 7 (First-meeting Probability)**.** *Given a source u and its reverse random walk $W(u,i)$ with length i, and a node $v \in \Omega$, $v \neq u$, the first-meeting probability of v and the reverse reachable walk $W(u,i)$ is defined as:*

$$P(v,W(u,i)) = \Pr_{W(v)}[v_i = u_i, v_{i-1} \neq u_{i-1}, ..., v_1 \neq u_1],$$

where $W(v) = (v = v_1, v_2, \cdots, v_i)$ is a reverse random walk starting from v.

The following theorem and lemma prove that an absolute error still holds with a high probability after constraining the length of $\sqrt{c}$-walks.

Theorem 1. *For any node $v \in \Omega$, $sim(u,v)$ and its estimator $s(u,v)$ satisfies* $\Pr\{|sim(u,v)-s(u,v)| \leq \varepsilon\} \geq 1-\delta$, *where* $s(u,v)=\frac{1}{n_r}\sum_{k=1}^{n_r}\sum_{i=2}^{\min(l,l_{\max})} P(v,W(u,i))$, $l_{\max} = \frac{1+\sqrt{c}}{(1-\sqrt{c})^2}$, $n_r = \frac{3c}{(\varepsilon-p\varepsilon_t)^2}\log\frac{n}{\delta}$, $\varepsilon_t = (\sqrt{c})^{l_{\max}}$, $p=\sum_{k=1}^{l_{\max}}(\sqrt{c})^{k-1}(1-\sqrt{c})$.

For simplicity, we split the Theorem 1 into Lemma 1, Lemma 2, and Lemma 3 and prove them separately.

Lemma 1. *The upper bound of the length l of $\sqrt{c}$-walk has at least p probability being $l_{\max} = \frac{1+\sqrt{c}}{(1-\sqrt{c})^2}$, i.e.,* $\Pr(l \leq l_{\max}) = p$, *where* $p=\sum_{k=1}^{l_{\max}}(\sqrt{c})^{k-1}(1-\sqrt{c})$.

Proof. This lemma defines an upper bound of the length of $\sqrt{c}$-walk with p probability according to the property of geometric distribution and Central Limit Theorems. Due to the principle of $\sqrt{c}$-walk, each step has the probability of $(1-\sqrt{c})$ to stop the random walk, so the length l of $\sqrt{c}$-walk obeys the geometric distribution, and its probability distribution is $P(l=k)=(1-\sqrt{c})(\sqrt{c})^{k-1}$, denoted as $l \sim GE(1-\sqrt{c})$. Its expectation is $El=\frac{1}{1-\sqrt{c}}$, the variance is $Dl=\frac{1-(1-\sqrt{c})}{(1-\sqrt{c})^2}=\frac{\sqrt{c}}{(1-\sqrt{c})^2}$. Due to the Central Limit Theorems and the Law of Large Numbers, $\bar{l} \sim N(El, Dl)$, and $P(El-2\cdot\frac{\sqrt{Dl}}{\sqrt{n}} \leq l \leq El+2\cdot\frac{\sqrt{Dl}}{\sqrt{n}}) \geq 0.95$, thus we set $l_{\max}$ to $El+2Dl=\frac{1+\sqrt{c}}{(1-\sqrt{c})^2}$. With the help of cumulative function of geometric distribution, we can calculate the probability of $l \leq l_{\max}$, i.e., $\Pr(l \leq l_{\max})=\sum_{k=1}^{l_{\max}}(\sqrt{c})^{k-1}(1-\sqrt{c})$. □

Lemma 2. *Let $s'(u,v)$ denotes the SimRank estimator between u and v when the length of l is not truncated, i.e., $s'(u,v)=\frac{1}{n_r}\sum_{k=1}^{n_r}\sum_{i=2}^{l}P(v,W(u,i))$. Then $s'(u,v)-s(u,v)=p\varepsilon_t$, where $\varepsilon_t=(\sqrt{c})^{l_{\max}}$.*

Proof. This lemma defines the SimRank estimator error with and without truncating the length of $\sqrt{c}$-walk. To prove the lemma, we discuss the relationship between l and $l_{\max}$. There are two cases: (a) when $l > l_{\max}$, $s'(u,v)-s(u,v)=\varepsilon_t$; and (b) when $l \leq l_{\max}$, $s'(u,v)-s(u,v)=0$. Since the probability that l is greater than $l_{\max}$ is p, then for any l, $s'(u,v)-s(u,v)=p\varepsilon_t$.

We first prove the case (a). Let ${s_k}'(u,v)$ denotes the SimRank estimator between u and v in the k-th iteration, and $s_k(u,v)$ denotes the SimRank estimator between u and v after truncating the length l of the $\sqrt{c}$-walk in the k-th iteration. The equation can be rewritten as,

$${s_k}'(u,v)-s_k(u,v) < \varepsilon_t = (\sqrt{c})^{l_{\max}}$$

when $l > l_{\max}$, where ${s_k}'(u,v)=\sum_{j=2}^{l}P(v,W(u,j))$, $s_k(u,v)=\sum_{j=2}^{l_{\max}}P(v,W(u,j))$. So if

$$\sum_{j=2}^{l}P(v,W(u,j))-\sum_{j=2}^{l_{\max}}P(v,W(u,j)) < (\sqrt{c})^{l_{\max}}$$

can be proved, then the Lemma 1.(a) holds. Obviously,

$$\sum_{j=2}^{l_{\max}+1}P(v,W(u,j))=\sum_{j=2}^{l_{\max}}P(v,W(u,j))(1+\sum_{y_1\in I(v)}\frac{\sqrt{c}}{|I(y_1)|}),$$

$$\sum_{j=2}^{l}P(v,W(u,j))=\sum_{j=2}^{l_{\max}}P(v,W(u,j))(1+\sum_{y_1\in I(v)}\frac{\sqrt{c}}{|I(y_1)|}\cdot\sum_{y_2\in I(y_2)}\frac{\sqrt{c}}{|I(y_2)|}\cdots\sum_{y_{l-l_{\max}}\in I(y_{l-l_{\max}})}\frac{\sqrt{c}}{|I(y_{l-l_{\max}})|}),$$

when in the l-th iteration. Because of $c \in [0,1]$,

$$\sum_{y_1\in I(v)}\frac{\sqrt{c}}{|I(y_1)|}\cdots\sum_{y_{l-l_{max}}\in I(y_{l-l_{max}-1})}\frac{\sqrt{c}}{|I(y_{l-l_{max}})|}<1,$$

Based on these formulas, the difference satisfies that

$$\sum_{j=2}^{l}P(v,W(u,j))-\sum_{j=2}^{l_{\max}}P(v,W(u,j)) < \sum_{j=2}^{l_{\max}}P(v,W(u,j)).$$

Because the probability satisfies that

$$\sum_{j=2}^{l_{\max}} P(v, W(u,j)) < (\sqrt{c})^{l_{\max}},$$

We get that

$$s_k{}'(u,v) - s_k(u,v) < \varepsilon_t = (\sqrt{c})^{l_{\max}}.$$

We next prove the case (b). When $l \leq l_{\max}$, then $l = \min(l, l_{\max})$. Since $s'(u,v)=\frac{1}{n_r}\sum_{k=1}^{n_r}\sum_{i=2}^{l} P(v, W(u,i))$, the estimator $s(u,v)=\frac{1}{n_r}\sum_{k=1}^{n_r}\sum_{i=2}^{\min(l,l_{\max})} P(v, W(u,i))$, then $s'(u,v) - s(u,v) = 0$. □

Lemma 3. *After constraining the length of $\sqrt{c}$-walk, it is necessary to iterate at least $n_r = \frac{3c}{(\varepsilon - p\varepsilon_t)^2}\log\frac{n}{\delta}$ times, to ensure the absolute error between estimator $s'(u,v)$ and $sim(u,v)$ has an upper bound with $1-\delta$ probability.*

The proof of Lemma 3 needs the help of the definition of the Chernoff bound.

Lemma 4. *(Chernoff bound [3]) For any set $\{x_i\}$ ($i \in [1, N]$) i.i.d random variables with mean μ and $x_i \in [0,1]$,* $\Pr\{\left|\sum_{i=1}^{N} x_i - N\cdot\mu\right| \geq N\cdot\varepsilon\} \leq exp\left(-\frac{N\cdot\varepsilon^2}{\frac{2}{3}\varepsilon + 2\mu}\right)$.

Proof. Lemma 3 defines the minimum number of iterations that guarantees the error less than ε with at least $1-\delta$. [10] has proved that when the length of the $\sqrt{c}$-walk is not truncated, in order to make the error of the estimated value $s'(u,v)$ and the actual value $sim(u,v)$ have an upper bound with high probability, at least $n_r = \frac{3c}{\varepsilon^2}\log\frac{n}{\delta}$ iterations are required. Meantime, $s'(u,v)$ is the average of each iterator, i.e., $\Pr|\forall v \in V, |s'(u,v) - sim(u,v)| \leq \varepsilon| \geq 1-\delta$, where $s'(u,v)=\frac{1}{n_r}\sum_{i=1}^{n_r} s'_k(u,v)$.

According to Lemma 2, $s'(u,v) - s(u,v) = p\varepsilon_t$. To ensure the probability that the error between $s(u,v)$ and $sim(u,v)$ is no greater than ε is $1-\delta$, it is necessary to ensure that the probability of the error between $s'(u,v)$ and $sim(u,v)$ is no greater than $\varepsilon - p\varepsilon_t$ is $1-\delta$.

The SimRank estimators $s_k(u,v)$ and $s_k{}'(u,v)$ are values in $[0,1]$ because of the original definition of SimRank, and $s'(u,v)=\frac{1}{n_r}\sum_{i=1}^{n_r} s'_k(u,v)$. Therefore, the Chernoff bound can be applied.

$$Pr[|s'(u,v) - sim(u,v)| \geq (\varepsilon - p\varepsilon_t)] \leq exp\left(-\frac{n_r\cdot(\varepsilon - p\varepsilon_t)^2}{3s(u,v)}\right)$$

Let $n_r = \frac{3c}{(\varepsilon - p\varepsilon_t)^2}\log\frac{n}{\delta}$ and notice $sim(u,v) < c$, we can get,

$$\Pr[|s(u,v) - sim(u,v)| \geq (\varepsilon - p\varepsilon_t)] \leq exp(-log\frac{n}{\delta}) = \frac{\delta}{n}$$

Combining the lower bounds of all nodes together, i.e., $\Pr[\forall v \in V, |s'(u,v) - sim(u,v)| \geq (\varepsilon - p\varepsilon_t)] \leq \delta$, so $\Pr[\forall v \in V, |s(u,v) - sim(u,v)| \geq \varepsilon] \leq \delta$. □

Time Complexity. The CrashSim algorithm first invokes the revReach algorithm to compute the probability of the $\sqrt{c}$-walk starting from u stopping at each node with different lengths, in which the worst case of the revReach algorithm is to traverse each edge only once. Thus, the complexity of the revReach algorithm is $O(m)$, where m is the number of edges.

Then the CrashSim algorithm has n_r iterations, and in the k-th trial, it will visit each node in the candidate set Ω and provide the total probability of "crashing" the $\sqrt{c}$-walk starting from u and each node $v \in \Omega$ with the maximum length l_{max}. The time complexity of this process is $O(|\Omega| \cdot \sum_{i=1}^{l_{\max}} i) = O(n \cdot l_{\max}^2)$, where $|\Omega|$ is the number of nodes in the candidate node set Ω, and l_{max} is the limited length of $\sqrt{c}$-walks. Note that, the limited length of $\sqrt{c}$-walks l_{max} is $\frac{1+\sqrt{c}}{(1-\sqrt{c})^2}$, which is a constant. Summing up for the n_r iterations, the time complexity of the iteration process is $O(n_r \cdot |\Omega|)$. Overall, the time complexity of the CrashSim algorithm is bounded by $O(m + n_r \cdot |\Omega|) = O(m + \frac{3c}{(\varepsilon - p\varepsilon_t)^2}\log\frac{n}{\delta} \cdot |\Omega|)$.

IV. CrashSim-T for Temporal SimRank Queries

When solving the temporal SimRank queries, CrashSim employs two pruning rules discussed in the following. We refer to the CrashSim algorithm with the pruning rules as *CrashSim-T* and describe it now in more details. Finally, we analyze the time complexity and correctness of the *CrashSim-T* algorithm.

A. Rationale

Recall that, the aim of a Temporal SimRank Query is to find the node set Ω, such that the SimRank of u and each node $v \in \Omega$, $s(u,v)$ continuously meet a certain query requirement during the entire query interval. As mentioned in Section I, there are two opportunities to further improve the efficiency of CrashSim for temporal SimRank queries. (i) Due to small changes between adjacent snapshots in temporal graphs, it is unnecessary to calculate the SimRank between u and the candidate set Ω at each time instant. (ii) Since a node must meet the query requirements at every time instant in order to be included in Ω, the size of node set Ω can only gradually reduce over time. Hence, it is unnecessary to compute the SimRank of all the nodes (i.e., use single-source SimRank computation), since we may only need the SimRank estimators of the source u and a partial set of nodes.

If we extend the static SimRank computation algorithms (e.g., ProbeSim) to solve the temporal SimRank queries, they will lose both opportunities, since they need to recompute the single-source SimRank estimators at every snapshot. If we however extend the index-based SimRank methods, such as READS [12], the opportunity (i) may lead to a large memory footprint to update the index, and the method still ignores opportunity (ii). The memory footprint will become critical in large graphs or for a long duration of the query interval.

Based on these, we extend the CrashSim algorithm to solve the temporal SimRank queries, since it naturally supports the computation of the SimRank of the source u and a partial set of nodes. Specifically, one of the inputs of CrashSim is the candidate node set Ω. When Ω is the entire set of nodes in a graph, the algorithm computes the single-source SimRank like other algorithms, but when Ω is given as particular nodes of the graph (i.e., partial set of nodes), CrashSim computes

the SimRank between the source u and only that partial set instead of the entire graph. This is a key difference between CrashSim and other single-source SimRank algorithms [10].

We now define two pruning rules, namely delta pruning and difference pruning to take the full advantage of the two discussed opportunities.

Delta Pruning. Let G_t, G_{t+1} be the t-th and $(t+1)$-th snapshots of the temporal graph G, respectively. Then $\Delta = G_{t+1} - G_t = \{\pm(x,y)|x,y \in V\}$. For each changed edge $x \to y$, we can define the "Affected Area".

Theorem 2 (Affected Area)**.** *The affected area of a changed edge $x \to y$ consists of: (i) the altered nodes in the reverse reachable tree of u, and (ii) $l_{max}-1$ length reachable nodes of y.*

Proof. Recall that $s(u,v) = \sum_i \Pr[W(u) \text{ and } W(v) \text{ first meet at } u_i]$, and in CrashSim, $s(u,v)$ is computed in two steps, computing the reverse reachable tree of u and performing n_r times random $\sqrt{c}$-walk starting from v. For the first part, the SimRank estimators of u and the altered nodes in the reverse reachable tree of u must be changed. For the second part, the SimRank estimators of u and $l_{max}-1$ length reachable nodes of y will be altered. Suppose there is a reachable path starting from y with the maximum length of l_{max}, it is obvious that $y_{l_{max}}$ are not affected by adding the edge $x \to y$, since the $\sqrt{c}$-walk with maximum length of l_{max} cannot reach x.

$$\underbrace{\overbrace{x \to y \to y_1 \to \cdots \to y_{l_{max}-1}}^{l_{max}} \to y_{l_{max}}}_{l_{max}+1}$$

□

The main idea of delta pruning is thus to reduce recomputation by ignoring the nodes of an unaffected area.

Next, we analyze the time complexity of computing the affected area to understand when delta pruning is efficient. The complexity of computing the affected area for a single-changed edge is similar to that of revReach algorithm, i.e., $O(|E(\Omega)|)$ where $E(\Omega)$ are the edges between nodes in Ω, and for $|E(\Delta)|$ changing edges, the complexity becomes $O(|E(\Omega)| \cdot |E(\Delta)|)$. Recall that the complexity of CrashSim is $O(|\Omega| \cdot n_r)$, *thus only when $|E(\Delta)| < \frac{|\Omega| \cdot n_r}{|E(\Omega)|}$, the delta pruning rule will speedup the process*. With the theorem of the affected area and this condition, we can formally define the property of the delta pruning rule as follows.

Property 1 (Delta Pruning)**.** *Given the changed edges between adjacent snapshots of temporal graphs, i.e., $\Delta = G_{t+1} - G_t$, when the number of changed edges satisfy $|E(\Delta)| < \frac{|\Omega| \cdot n_r}{|E(\Omega)|}$, the unaffected area of the candidate node set Ω can be exempted from computing SimRank estimators for instant $t+1$.*

Example 3 (A running example of Delta Pruning)**.** *Suppose that l_{max} is 2. The reverse reachable tree of A is shown in Fig. 4(a). Consider Fig. 1(a) and (b), if we delete the edge $H \to F$, the reverse reachable tree of A (part i of the affected area) will not be altered. Meanwhile, since F has no out-neighbours, the $l_{max}-1$ length reachable nodes of F is F itself. Therefore, only the SimRank score of A and F will be changed after deleting the edge $H \to F$.*

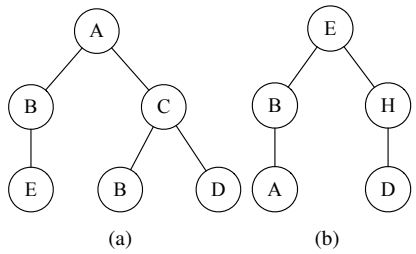


Fig. 4: Reverse Reachable Tree of A and E

Difference Pruning. Before presenting the difference pruning, we first introduce the notion of related area of the SimRank estimator $s(u,v)$.

Definition 8 (Related Area)**.** *$s(u,v)$ is related to the l_{max} length reverse reachable tree of u and v.*

Proof. Recall the definition of SimRank based on $\sqrt{c}$-walk,

$$\begin{aligned} s(u,v) &= Pr[W(u) \text{ and } W(v) \text{ meet}] \\ &= \sum_i Pr[W(u) \text{ and } W(v) \text{ first meet at } u_i]. \end{aligned}$$

It is obvious that $s(u,v)$ depends on $W(u)$ and $W(v)$. In CrashSim, we have proven that we can gain reliable SimRank estimators after constraining the length of $\sqrt{c}$-walk. We also use the reverse reachable tree to list all the reverse reachable paths starting from u, i.e., $W(u)$ and record the probability of the random walk stopping at different nodes u_i with different lengths. Similarly, $W(v)$ can be expressed by the reverse reachable tree of v. As a result, $s(u,v)$ is related to the reverse reachable tree of u and v. □

Thus the pruning rule based on the related area compares the reverse reachable tree of u and each node in Ω. If the reverse reachable tree of u is stable, then we can filter out those nodes whose reverse reachable tree is unchanged.

Next, we analyze the time complexity of computing the related area to understand when the difference pruning is effective. The complexity of computing the related area of a single node is similar to that of the revReach algorithm, i.e., $O(|E(\Omega)|)$. The complexity of computing the related area of each node in the candidate set Ω is thus $O(|\Omega| \cdot |E(\Omega)|)$. Recall that the complexity of CrashSim is $O(|\Omega| \cdot n_r)$, thus when $|E(\Omega)| < n_r$, the difference pruning will speedup the computational process. Now we formally define the property of difference pruning.

Property 2 (Difference Pruning)**.** *Given snapshots G_t and G_{t-1}, if the related area of u is stable and $|E(\Omega)| < n_r$, the nodes whose reverse reachable trees are unchanged can be exempted from computing SimRank estimators for instant t.*

Example 4 (A running example of Difference Pruning)**.** *Suppose that the candidate node set Ω is $\{E\}$, l_{max} is 2 and*

the reverse reachable trees of A and E are shown in Fig. 4(a) and (b). Consider Fig. 1(b) and (c), if we add an edge $G \to F$, the reverse reachable trees of A and E will not be changed in the adjacent snapshots, thus there is no need to re-compute the SimRank value of A and E.

B. CrashSim-T Algorithm

The main idea behind CrashSim-T is to traverse the graph for all time snapshots and during each iteration we first check whether the conditions of delta pruning and difference pruning are satisfied. If so, we can disregard these nodes as part of the candidate node set Ω for the current time instant. Then, we invoke the CrashSim algorithm to compute the SimRank score of u and residual nodes. Finally, according to different query requirements (such as threshold or trend query) we can filter out unsatisfied nodes that do not meet the requirements.

Algorithm 3: CrashSim-T Algorithm

input : $\mathbf{G}(V, \mathbf{E})$, $u \in V$, query interval $[T_1, T_t]$, ε, δ
output: Ω

1 $\Omega \leftarrow V$
2 $S_1 = CrashSim(G_{T_1}, u, \Omega, \varepsilon, \delta)$
3 **for** *each* $T_i \in [T_2, T_t]$ **do**
4 $\Omega' = \Omega$
5 $U_i = revReach(G_{t_i}(V, E_\Omega), u)$
6 $U_{i-1} = revReach(G_{t_{i-1}}(V, E_\Omega), u)$
7 **if** $U_i = Ui - 1$ **then**
8 $\Delta = G_{T_i} - G_{T_{i-1}}$
9 **if** $|E(\Delta)| < \frac{|\Omega| \cdot n_r}{|E(\Omega)|}$ **then**
10 $A = \emptyset$
11 **for** *each* $x \to y \in E(\Delta)$ **do**
12 $A+ = revReach(G_{t_i}(V, E_\Omega), y)$
13 $\Omega' = \Omega - A$
14 **if** $|E(\Omega)| < n_r$ **then**
15 **for** *each* $v_j \in \Omega$ **do**
16 $R_i = revReach(G_{t_i}(V, E_\Omega), v_j)$
17 $R_{i-1} = revReach(G_{t_{i-1}}(V, E_\Omega), v_j)$
18 **if** $R_i = R_{i-1}$ **then**
19 $\Omega' = \Omega - v_j$
20 $S_i = CrashSim(G_{T_i}, u, \Omega')$
21 **for** *each* $s(u, v) \in S_i$ **do**
22 delete the unsatisfied node from Ω
23 **return** Ω;

We now describe the details of the CrashSim-T algorithm. Given a temporal graph $\mathbf{G}(V, \mathbf{E})$, a source $u \in V$, a query time interval $[T_1, T_t]$, a sampling error parameter ε and a failure probability δ, the algorithm returns a candidate node set Ω, in which each node meets the query requirement during the entire query interval.

The pseudo-code for the CrashSim-T algorithm is illustrated in Algorithm 3. At first, we initialize the candidate set Ω to the entire set of nodes (Line 1). We then invoke CrashSim to compute the SimRank of u and each node at T_1 (Line 2). After that, the algorithm runs for all time instants in $[T_2, T_t]$ (Line 3). For the i-th trial, the algorithm first initializes Ω' to record the residual node set for which we need to compute the SimRank scores at current query time instant (Line 4). Then, the algorithm compares the reverse reachable trees of u between the snapshots T_i and T_{i-1} (Lines 5-6). If the reverse reachable trees of u between the adjacent snapshots are the same, then there is a possibility to meet delta pruning and difference pruning conditions. Next, we explore the condition of delta pruning (Lines 7-13). During this process, we make Δ to record the difference between the adjacent snapshots (Line 8), and then explore whether the number of changed edges meets the condition of delta pruning (Line 9). If so, we initialize a set A to record the nodes in the affected area (Line 10). For each edge $x \to y$, we invoke an algorithm similar to revReach to compute the affected area of y (note that revReach computes the reverse reachable tree for y, but we just need the reverse reachable nodes) (Lines 11-12). After the iteration is finished, the residual node set Ω' is updated (Line 13).

Similarly, we check the condition of difference pruning (Lines 14-19). If the number of edges in Ω, i.e., $|E(\Omega)|$ is less than n_r, the difference pruning will be applied (Line 14). For each node v_j in Ω, we compute the reverse reachable tree of v_j for adjacent snapshots, i.e., G_t and G_{t-1} (Lines 16-17). If they are the same, we do not need to compute the SimRank of u and v_j since the value is the same as the previous one, thus we delete it from Ω' (Lines 18-19).

After that, we use CrashSim to compute the SimRank score of u and nodes in Ω', since Ω' records the nodes for which we need to compute SimRank scores (Line 20). Finally, according to different query requirements, such as threshold and trend, we delete the unsatisfied nodes from Ω (Line 21-22). After iterating this process over all time instants, we get the final candidate node set Ω, in which each node satisfies the query requirements for the entire query time interval.

C. Theoretical Analysis

Time Complexity. The algorithm is invoked over all time instants, performing t iterations in each instant. In each iteration, CrashSim is invoked, resulting in the time complexity of $O(n_r \cdot |\Omega|)$, where n_r is the number of iterations and $|\Omega|$ is the number of nodes in the candidate node set at time T_i. Since redundant computations are removed using the pruning rules, the worst case time complexity of CrashSim-T is $O(t \cdot n_r \cdot |\Omega|)$.

Correctness. We have proven that the accuracy of the CrashSim algorithm has a lower bound through Theorem 1, which is based on the $\sqrt{c}$-walks with the constrained length. The pruning rules are based on the $\sqrt{c}$-walk with the limited length as well. Thus CrashSim-T does not introduce any additional error and still has provable approximation guarantees of the SimRank estimators for every snapshot.

V. Experimental evaluation

This section experimentally evaluates the proposed algorithms, i.e., CrashSim and CrashSim-T, against the state-of-the-art SimRank methods. We conduct all the experiments on a Windows 10 machine with a 3.4GHz CPU and 8GB memory, and all of the algorithms are implemented in C++.

Datasets. To evaluate the performance of the proposed algorithms, we use five real datasets from Stanford Large Network Dataset Collection, namely AS-733, AS-Caidi, Wiki-Vote, HepTh, and HepPh. The datasets AS-733 and AS-Caidi are originally temporal graph datasets that can be used directly. Wiki-Vote, HepTh, and HepPh however do not contain temporal information, hence we generate the synthetic datasets with 100 snapshots for each. The detailed information about the datasets can be found in Table III, showing the type of graphs, the number of nodes (n), the number of edges (m), and the number of time snapshots (t).

TABLE III: Real and Synthetic Datasets

Datasets	Type	n	m	t
AS-733	Undirected	6,474	13,233	733
AS-Caidi	Directed	26,475	106,762	122
Wiki-Vote	Directed	7,155	103,689	100
HepTh	Undirected	9,877	25,998	100
HepPh	Directed	34,546	421,578	100

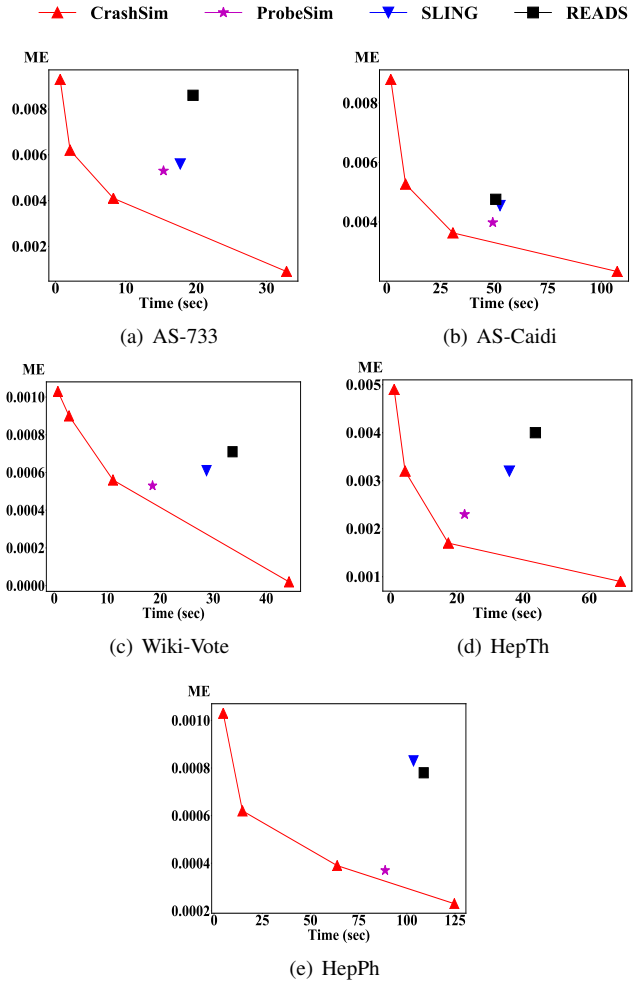
Comparison baselines. For a single snapshot, we compare CrashSim against SLING algorithm [18], ProbeSim algorithm [10], and READS algorithm [12]. The SLING algorithm is the state-of-the-art index-based algorithm for computing the single-source SimRank for static graphs; ProbeSim is the state-of-the-art index-free algorithm that naturally supports static and dynamic graphs; and READS is the state-of-the-art index-based single-source SimRank for dynamic graphs. Consistent with the previous study, we set the decay factor c of the SimRank algorithm to 0.6 [7]. For SLING and ProbeSim algorithms, we set their maximum error $\varepsilon = 0.025$. For READS algorithm, we set $r = 100$, $r_q = 10$, and $t = 10$. As for the proposed CrashSim algorithm, we vary the parameter ε from 0.0125, 0.025, 0.05, and 0.1 which corresponds to the maximum absolute error allowed in the SimRank computation. By varying the value of this parameter, we can examine the trade-off between query efficiency and accuracy of the CrashSim algorithm. The ground-truth results on each dataset are computed by the Power Method [7] with 55 iterations.

For the temporal SimRank queries, we compare CrashSim-T against SLING, ProbeSim, and READS algorithms. Since they are not designed to answer temporal SimRank queries, they need to be modified (see Section IV-A for more details). The parameters of SLING, ProbeSim, and READS algorithms are the same as that of a single snapshot, but the parameter ε used in the CrashSim algorithm is set to 0.025, which is similar to that of the ProbeSim algorithm.

Setting and metrics. For each single-source SimRank computation in a snapshot from source u, we define the maximum error ME as the maximum difference between the ground-truth results and the values returned by each algorithm on a single snapshot, that is, $ME = \max|s(u,v) - \tilde{s}(u,v)|\,(v \in V)$.

To evaluate the effectiveness of CrashSim-T on the temporal SimRank queries, we define a *precision* metric to express the accuracy of the query results. Specifically, $precision = \frac{v(k_1) \cap v(k_2)}{\max(k_1,k_2)}$, where $v(k_1)$ represents the query result set calculated by the power method, k_1 represents the number of nodes in the result set calculated by this method, $v(k_2)$ represents the set of query results obtained by our algorithm and other algorithms, and k_2 represents the number of nodes in the result set for those algorithms.

http://snap.stanford.edu/data/index.html



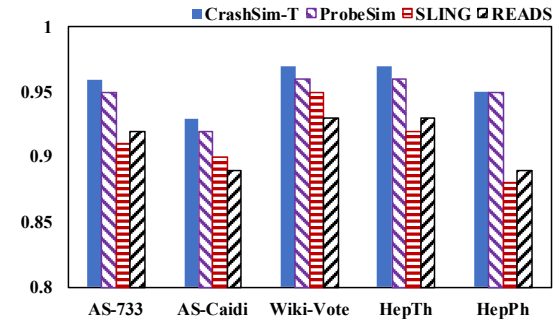
(a) AS-733 (b) AS-Caidi (c) Wiki-Vote (d) HepTh (e) HepPh

Fig. 5: Response time and Max Error (ME) when computing single-source SimRank on the static datasets

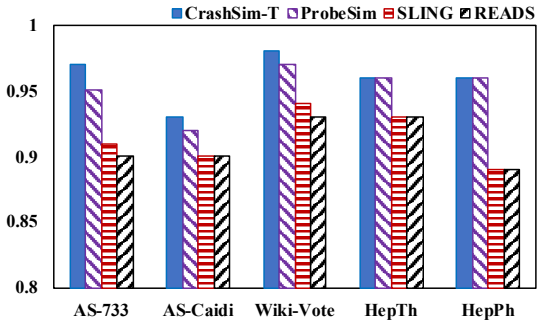
To further evaluate the efficiency of CrashSim-T on the temporal SimRank queries, we use synthetic datasets that are extracted from the AS-733 dataset. We measure the total time duration of CrashSim-T and the competitors when answering the temporal SimRank trend query over 100, 200, 500, 700 snapshots separately.

A. Evaluation over static graphs

In the first set of experiments, we randomly generate single-source SimRank computation on a single snapshot of each dataset for 100 repetitions and measure the average time duration and maximum error of each method over the static graphs. As shown in Fig. 5, the time increases gradually for CrashSim but the maximum absolute error ME declines



(a) Precision of Temporal Trend Query



(b) Precision of Temporal Threshold Query

Fig. 6: Precision of the different algorithms when answering Temporal Trend and Threshold Queries

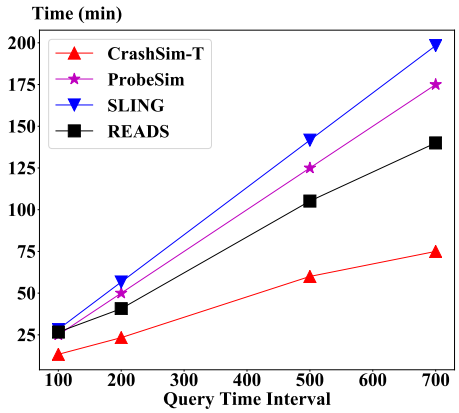


Fig. 7: The impact of the query interval on the response time of the algorithms

rapidly when the parameter ε of CrashSim algorithm varies between 0.1 (shown on the left hand side), 0.05, 0.025 to 0.0125 (shown on the right hand side). We also observe that CrashSim with ε varying from 0.1, 0.05 to 0.025 has better performance (faster response time) than ProbeSim, SLING and READS algorithms. This is consistent with the fact that CrashSim has the lowest time complexity $O(n \cdot n_r)$. Note that, the response time of SLING and READS reported here consists of indexing time and computational time.

As for the maximum absolute error, CrashSim with ε 0.025 and 0.0125 has lower ME than ProbeSim, SLING and READS algorithms. Theorem 1 has proved that CrashSim can obtain the same maximum error bound as ProbeSim algorithm by increasing the constant multiple numbers of iteration. The SLING algorithm also has an additive error bound. Since the READS algorithm has no maximum error guarantee, all three CrashSim, ProbeSim and SLING algorithms have better ME than the READS algorithm.

B. Evaluation over temporal graphs

Our second set of experiments shown in Fig. 6(a) and (b) evaluates the accuracy of different algorithms when answering the temporal SimRank trend and threshold queries. Among the competitors, CrashSim-T provides the highest precision since it has the lowest ME in a single snapshot.

Our last set of experiments focuses on the impact that the query duration has on the efficiency of CrashSim-T and the other algorithms. We select 100, 200, 500, 700 snapshots of the AS-733 dataset as the test datasets, and compare the total running time of each solution when answering the temporal trend SimRank query. Note that we have performed the same experiment over the temporal threshold SimRank query as well. The results are omitted due to lack of space, nonetheless, they are consistent with the results for the trend query (with less than 5% of discrepancy).

From Fig. 7. it can be observed that the response time of all algorithms increases with the increase in the query interval duration. The total response time of ProbeSim and SLING algorithms increases linearly since both of them re-compute the SimRank value at every snapshots. The READS algorithm needs to update the index and re-compute the SimRank scores at every edge or node update. CrashSim-T has the fastest response time during the entire query interval, since it has the lowest time complexity on a single snapshot, and reduces the unnecessary computation over the adjacent snapshots by employing the pruning strategies. When the number of snapshots increases, the benefit of CrashSim-T becomes more apparent, since the number of nodes in the candidate set that satisfy the query decreases over time.

VI. Related Work

SimRank computation in Static Graphs. Jeh and Widom [7] first presented SimRank to measure the similarity of two nodes in a graph, and proposed a recursive deterministic method to compute the SimRank scores of all pairs of nodes. Lizorkin *et al.* [11] propose a method to estimate the accuracy of SimRank scores and find the minimum number of iterations required to achieve a desired accuracy.

Another method for SimRank computation is linearization method [5], [6], [8], [24], [26]. Fujiwara *et al.* [5] propose a non-iterative method to calculate the single-pair SimRank based on the Sylvester equation. Kusumoto *et al.* [8] propose an effective algorithm for single-source SimRank top-k query; the approach cannot however guarantee the ε worst-case error. To solve the problem, Yu *et al.* [26] propose an algorithm that does not require pre-computation, and can guarantee ε worst-case error, however the computational complexity of the approach is high.

The Monte Carlo method [4], [10], [13], [17], [18] can also be used to compute the SimRank scores. Fogaras *et al.* [4] first define SimRank between two nodes as the expectation of the encounter of the reverse random walks starting from these two nodes. Pei *et al.* [13] propose a single-source SimRank Top-k query algorithm whose main idea is to search for candidate sets of neighbors and rank them, instead of scanning through the entire graph. Tian *et al.* [18] propose SLING, an index-based algorithm for answering single-source SimRank queries with ε worst-case error. To the best of our knowledge, the ProbeSim algorithm proposed in [10] is the state-of-art index-free single-source SimRank algorithm that provides a non-trivial theoretical guarantee (details described in Section II-D).

SimRank Computation in Dynamic Graphs. Li *et al.* [9] first studied the all-pair SimRank computation on dynamic graphs, with the main idea to factorize the backward transition matrix. Yu *et al.* [25] propose a fast incremental algorithm for all-pair SimRank by defining the update matrix of every link change using the rank-one Sylvester equation. Shao *et al.* [16] propose TSF schema for SimRank-based search, wherein a random walk of a set of one-way graphs is indexed. Wong *et al.* [12] present READS, an index schema based on unidirectional random walk to compute the top-k single-source SimRank over dynamic graphs. Wang *et al.* [19] propose a novel local push based algorithm to compute all-pairs SimRank in dynamic graphs.

Different from the afore-mentioned approaches, CrashSim not only efficiently computes the single-source SimRank with provable approximation guarantees, but naturally supports the SimRank queries in temporal graphs. Temporal graphs have become an advanced research hotspot in recent years [2]. Wu *et al.* [21] study the shortest path in temporal graphs, and formally defined the minimum temporal graphs. The problem of mining a set of diversified temporal subgraph patterns from a temporal graph is studied in [22]. A fast incremental approach for continuous frequent subgraph mining problem on a single large evolving graph is proposed in [1]. Such works however focus on orthogonal problems such as calculating the shortest path, and density subgraph, and not SimRank.

VII. Conclusion

In this work, we propose CrashSim, an index-free algorithm for single-source and partial SimRank computation in static graphs. The main intuition behind CrashSim is to regard the SimRank estimators as the average probability of two $\sqrt{c}$-walks with constraining length first meeting. To support the temporal SimRank queries over temporal graphs, we introduce CrashSim-T – an extension to CrashSim that employs two pruning strategies (delta and difference pruning) that substantially improve the algorithm's performance over temporal graphs. Our experiments show that both CrashSim and CrashSim-T algorithms outperform the state-of-art algorithms significantly in terms of response time and precision.

Acknowledgment. Mo Li is supported by the Chinese Scholarship Council. This work is partially supported by the ARC Linkage Projects (No. LP180100750), the National Natural Science Foundation of of China (Nos. 61472069 and 61402089), China Postdoctoral Science Foundation (Nos. 2019T120216 and 2018M641705), the Fundamental Research Funds for the Central Universities (N180408019 and N180101028), the CETC Joint Fund, the Open Program of Neusoft Institution of Intelligent Healthcare Technology, Co. Ltd. (No. NRIHTOP1802), and the fund of Acoustics Science and Technology Laboratory.

References

[1] Abdelhamid, E., Canim, M., Sadoghi, M., et al: Incremental frequent subgraph mining on large evolving graphs. TKDE pp. 2710–2723 (2017)
[2] Aggarwal, C., Subbian, K.: Evolutionary network analysis: A survey. ACM Computing Surveys pp. 1–36 (2014)
[3] Chung, F.R.K., Lu, L.: Concentration inequalities and martingale inequalities: A survey. Internet Mathematics **3**(1), 79–127 (2006)
[4] Fogaras, D., Rácz, B.: Scaling link-based similarity search. In: WWW. pp. 641–650 (2005)
[5] Fujiwara, Y., Nakatsuji, M., Shiokawa, H., Onizuka, M.: Efficient search algorithm for simrank. In: ICDE. pp. 589–600 (2013)
[6] He, G., Feng, H., Li, C., Chen, H.: Parallel simrank computation on large graphs with iterative aggregation. In: SIGKDD. pp. 543–552 (2010)
[7] Jeh, G., Widom, J.: Simrank: A measure of structural-context similarity. In: SIGKDD. pp. 538–543 (2002)
[8] Kusumoto, M., Maehara, T., Kawarabayashi, K.I.: Scalable similarity search for simrank. In: SIGMOD. pp. 325–336 (2014)
[9] Li, C., Han, J., He, G., et al: Fast computation of simrank for static and dynamic information networks. In: EDBT. pp. 465–476 (2010)
[10] Liu, Y., Zheng, B., He, X., et al: Probesim: Scalable single-source and top- k simrank computations on dynamic graphs. In: PVLDB. vol. 11, pp. 14–26 (2017)
[11] Lizorkin, D., Velikhov, P., Grinev, M., et al: Accuracy estimate and optimization techniques for simrank computation. VLDB Journal pp. 45–66 (2010)
[12] Minhao Jiang, Ada Wai-Chee Fu, R.C.W.W., Wang, K.: Reads: A random walk approach for efficient and accurate dynamic simrank. In: PVLDB. pp. 937–948 (2017)
[13] Pei, L., Lakshmanan, L.V.S., Yu, J.X.: On top-k structural similarity search. In: ICDE. pp. 774–785 (2012)
[14] Ruoming, J., Victor, E.L., Hui, H.: Axiomatic ranking of network role similarity. In: KDD. pp. 922–930 (2011)
[15] Semertzidis, K., Pitoura, E.: Top-k durable graph pattern queries on temporal graphs. TKDE **31**(1), 181–194 (2018)
[16] Shao, Y., Cui, B., Chen, L., et al: An efficient similarity search framework for simrank over large dynamic graphs. In: PVLDB. pp. 838–849 (2015)
[17] Song, J., Luo, X., Gao, J., et al: Uniwalk: Unidirectional random walk based scalable simrank computation over large graph. TKDE pp. 992–1006 (2018)
[18] Tian, B., Xiao, X.: Sling: A near-optimal index structure for simrank. In: SIGMOD. pp. 1859–1874 (2016)
[19] Wang, Y., Lian, X., Chen, L.: Efficient simrank tracking in dynamic graphs. In: ICDE. pp. 545–556 (2018)
[20] Wei, Z., He, X., Xiao, X., et al: Prsim: Sublinear time simrank computation on large power-law graphs. In: SIGMOD. pp. 1042–1059 (2019)
[21] Wu, H., Cheng, J., Huang, S., et al: Path problems in temporal graphs. In: PVLDB. pp. 721–732 (2014)
[22] Yang, Y., Yan, D., Wu, H., et al: Diversified temporal subgraph pattern mining. In: SIGKDD. pp. 1965–1974 (2016)
[23] Yin, X., Han, J., Yu, P.S.: Linkclus: Efficient clustering via heterogeneous semantic links. In: PVLDB. pp. 427–438 (2006)
[24] Yu, W., Lin, X., Zhang, W., Chang, L., Pei, J.: More is simpler: Effectively and efficiently assessing nodepair similarities based on hyperlinks. In: PVLDB. vol. 7, pp. 13–24 (2013)
[25] Yu, W., Lin, X., Zhang, W.: Fast incremental simrank on link-evolving graphs. In: ICDE. pp. 304–315 (2014)
[26] Yu, W., Mccann, J.A.: High quality graph-based similarity search. In: SIGIR. pp. 83–92 (2015)
[27] Zeinab, A., Vahab, S.M.: A recommender system based on local random walks and spectral methods. In: WebKDD/SNA-KDD. pp. 139–153 (2007)