Information Sciences 381 (2017) 1–19

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Fast top-*k* similarity join for SimRank

Ruiqi Li [a], Xiang Zhao [a,c,*], Haichuan Shang [b], Yifan Chen [a], Weidong Xiao [a,c]

[a] College of Information System and Management, National University of Defense Technology, Changsha, Hunan, 410072, China
[b] National Institute of Information and Communication Technology, Koganei, Tokyo, 184-8795, Japan
[c] Collaborative Innovation Center of Geospatial Technology, Wuhan, Hubei, 410000, China

ARTICLE INFO

Article history:
Received 10 March 2016
Revised 6 September 2016
Accepted 22 October 2016
Available online 15 November 2016

Keywords:
Top-*k*
Similarity join
Iterative pruning
Upper bound

ABSTRACT

SimRank is a well-studied similarity measure between two nodes in a network. However, evaluating SimRank of all nodes in a network is not only time-consuming but also not pragmatic, since users are only interested in the most similar pairs in many real-world applications. This paper focuses on top-*k* similarity join based on SimRank. In this work, we first present an incremental algorithm for computing SimRank. On top of that, we derive an iterative batch pruning framework, which is able to iteratively filter out unpromising nodes and obtain the top-*k* pairs in a fast mode. Specifically, we define the concept of super node such that for a node in the network, the SimRank with its super node is not less than that with any others. Based on this feature, we propose a tight upper bound for each node that can be easily calculated after each iteration. Experiments on both real-life and synthetic datasets demonstrate that our method achieves better performance and scalability, in comparison with the state-of-the-art solution.

1. Introduction

SimRank is a popular link-based similarity measure to evaluate node pair similarities in a network. It is widely adopted in many real-world applications including sponsored search, web spam detection, schema matching, etc. The basic intuition behind SimRank is based on the object-to-object relationship found in many domains of interest; that is, two objects are similar if they are referenced by similar objects. By recursively computing the similarity between two nodes based on the similarities between their neighbors, SimRank measures the similarity of structural context, which can be applied to any domain where there are enough relevant relationships among objects. For example, in a social network, SimRank is a useful function to identify similar users and suggest potential friends to users, which can further help predict possible links and trace information disseminated over the network. As to computational challenges, significant efforts have been devoted to efficient SimRank computation. Consider a (directed) network G with n nodes. The original iterative algorithm [4] computes all-pair SimRank in $O(\xi d^2 n^2)$ time in ξ iterations, where d is the average in-degree of the network. It is improved to $O(\xi d n^2)$ time via partial sum memorization [9], and further expedites through fine-grained memorization [17]. Albeit, computing all-pair SimRank in real-life large networks can be still prohibitive in terms of both time and space costs, seeing the ever-growing sizes of networks. As an alternative, identifying node pairs of highest SimRank meets the needs of retrieving only the highly similar node pairs. Most of the existing methods require a similarity threshold from users [6,18], and

* Corresponding author at: College of Information System and Management, National University of Defense Technology, Changsha, Hunan Province, 410072, China.
E-mail address: xiangzhao@nudt.edu.cn (X. Zhao).

http://dx.doi.org/10.1016/j.ins.2016.10.042
0020-0255/

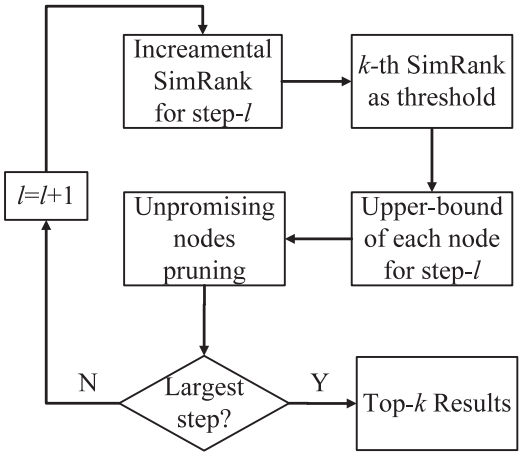


Fig. 1. Overview of KSimJoin.

only node pairs whose similarities exceed the threshold are returned. Unluckily, it can be formidable to select an appropriate threshold that guarantees decent results yet without irrelevant pairs. Lately, an appealing approach is introduced to find k most similar node pairs by SRK-Join [14], without the requirement of a threshold as input. In particular, it defines *second meeting probability* (cf. Section 3), related to each node v, to denote the probability of two random paths starting both from node v and firstly meet at another same node again. Leveraging the probabilities, it then encodes each v as a multi-dimensional vector such that SimRank of two nodes can be computed by the *dot product* of the two vectors. To retrieve the top-k pairs, SRK-Join executes two times of scanning of every path that starts from the object nodes. The first is to help to compute a candidate set containing $2k$ distinct nodes, and the second is to obtain the top-k pairs. SRK-Join is tested to be efficient on real-life networks; nonetheless, we observe margins for further improvement. In particular, (1) in order to encode every node as a vector, SRK-Join requires calculating second meeting probability of every node. Through experiments, however, we observe that calculating second meeting probability for nodes with high in-degrees can be fairly time-consuming, as calculating second meeting probability of a node requires $O(d^{\xi})$ time. Thus, evaluating the first-meeting probability of two paths, regardless of (same or not) starting nodes, via second meeting probability, incurs significant computational overhead. (2) In SRK-Join, the $2k$ candidate nodes are re-scanned for final results. This results in computational redundancy, since the candidate nodes have already been scanned once. Moreover, the second scanning is dependent on the first scanning, which means the information calculated in the first round can be somehow reused. Thus, we exploit the opportunities for better performance.

In this work, we first present an incremental algorithm of SimRank by iterations, and then encapsulate the procedure into a novel iterative batch pruning framework KSimJoin. We design an upper bound for each node that is both tight and easy to be derived. Specifically, as the iteration goes deeper, the upper bound becomes tighter based on second meeting probabilities newly calculated in the current iteration. Thus, a quantity of nodes will be removed in each iteration before SimRank is eventually calculated. Moreover, computation sharing of the required second meeting probabilities among the iterations is enabled for speedup. Fig. 1 gives an overview of our framework KSimJoin. In each iteration, we incrementally calculate SimRank for candidate nodes and their upper-bounds. The candidate nodes are the whole network in the first iteration. We select the top-k SimRank node pairs and compare the upper-bound of each node with the kth largest SimRank (threshold). Only those with upper-bound larger than the threshold can remain to the next iteration. As soon as the largest step is reached, the top-k SimRank node pairs will be returned as the answer. As for large networks, we put forward further improvement to reduce the running time and memory footprints of the algorithm.

Contribution. To summarize, we make the following contributions.

- We develop an incremental algorithm for partial SimRank, and on top of that, propose an iterative batch pruning framework for top-k similarity joins, which can efficiently discover k node pairs with the largest SimRank in the network.
- We define the concept of super node $\mathcal{V}_s$ of node v such that SimRank between v and its super node is not less than that between v and any other nodes. Based on this, we put forward a tight upper bound to prune unpromising nodes under the framework.
- The resultant algorithm KSimJoin is evaluated on real-life and synthetic networks. Empirical results witness the effectiveness and efficiency of the proposed techniques, suggesting a better option for potential applications.

Organization. Section 2 introduces the basic concepts related to our work, and formally states the problem. Then, we introduce the method for computing partial SimRank in Section 3. We present the iterative batch pruning framework in Section 4, including the design of upper bound. Further improvement for large networks is presented in Section 5. Empirical results are reported and analyzed in Section 6. We discuss related work in Section 7, and conclude the paper in Section 8.

2. Preliminaries

In this section, we brief the models of SimRank and the basic intuition behind it, and then formally define the problem.

A (directed) network $G(V, E)$ has n nodes. As to an edge $\langle u, v\rangle \in E$ from u to v, $u, v \in V$, it is an *in-coming* edge of v, and hence, u is an *in-neighbor* of v. For each node $v \in V$, $I(v)$ denotes the set of in-neighbors of v.

Definition 1 (Iterative model)**.** Given a network G, the similarity of node pair (v, u), $v, u \in V$, can be defined as

$$S(v,u) = \begin{cases} 1, & \text{if } v = u; \\ \dfrac{c}{|I(v)||I(u)|} \sum\limits_{i=1}^{|I(v)|} \sum\limits_{j=1}^{|I(u)|} S(I_i(v), I_j(u)), & \text{otherwise,} \end{cases} \tag{1}$$

where $I_i(v)$ (resp. $I_j(u)$) is the i (resp. j)th in-neighbor of v (resp. u), $i \in [1, |I(v)|]$ (resp. $j \in [1, |I(u)|]$), and c is a decay factor between 0 and 1.

Eq. (1) shows that the similarity between v and u is the average similarity between in-neighbors of v and u; as a special case, the similarity between an object and itself is defined to be 1. For real-world networks that have circles, the solution can be reached by iterations to a fixed-point. That is,

$$S_{\ell+1}(v,u) = \frac{c}{|I(v)||I(u)|} \sum_{i=1}^{|I(v)|} \sum_{j=1}^{|I(u)|} S_\ell(I_i(v), I_j(u)),$$

where $S_0(v,u) = 1$ if $v = u$, and otherwise, 0. Furthermore, $lim_{\ell\to\infty} S_\ell(v,u) = S(v,u)$. The iterative form of SimRank has fast convergence rate and the ranking of SimRank stabilizes within 5 iterations [4], i.e., $\ell \le 5$.

Besides the numerical form, $S(u, v)$ can also be interpreted by a figurative model, measuring how soon two random surfers are expected to first meet at the same node, if they start at nodes v and u, respectively, and randomly walk on the network *backwards*. By "backwards", we mean that they only walk along the *reversed* direction of an edge, i.e., one can only walk from v to u if there is an edge $\langle u, v\rangle$ in the network.

Definition 2 (Two-way path)**.** A node pair sequence $\mathcal{TP} = \{(v_1, u_1) \to (v_2, u_2) \to \cdots \to (v_{\ell+1}, u_{\ell+1})\}$ is called a two-way path if

(1) $\forall i \in [1, \ell+1]$, $v_i, u_i \in V$; and
(2) $\forall i \in [1, \ell+1)$, $\langle v_{i+1}, v_i\rangle, \langle u_{i+1}, u_i\rangle \in E$.

The length of the two-way path is ℓ. Further, it is called

- a *meeting* two-way path if $v_{\ell+1} = u_{\ell+1}$;
- a *multi-meeting* two-way path if $\exists i \in [1, \ell]$, $v_i = u_i$, and $v_{\ell+1} = u_{\ell+1}$;
- a *first-meeting* two-way path if $\forall i \in [1, \ell]$, $v_i \ne u_i$, and $v_{\ell+1} = u_{\ell+1}$.

Immediate is that the set of meeting two-way paths is constituted of the sets of first meeting and multi-meeting two-way paths.

The probability of a two-way path is defined as $P(\mathcal{TP}) = \prod_{i=1}^{\ell} \frac{c}{|I(v_i)||I(u_i)|}$. We use $\{(v_1, u_1) \overset{\ell}{\rightsquigarrow} (v_{\ell+1}, u_{\ell+1})\}$ to denote the set of two-way paths that start at (v_1, u_1) and end at $(v_{\ell+1}, u_{\ell+1})$ at step-ℓ, and $\{(v, u) \overset{\ell}{\underset{fm}{\rightsquigarrow}} (x, x)\}$ to denote the set of first-meeting two-way paths that start at (v, u) and end at (x, x) at step-ℓ. Based on the probability of first-meeting two-way paths, we can compute SimRank as formalized below.

Definition 3 (Random surfer model)**.** Given a network G, the similarity of node pair (v, u), $v, u \in V$, can be defined as

$$S(v,u) = lim_{\xi\to\infty} \sum_{\ell=1}^{\xi} \sum_{x\in V} P\left(\left\{(v,u) \overset{\ell}{\underset{fm}{\rightsquigarrow}} (x,x)\right\}\right), \tag{2}$$

where ξ is the maximum step of random walk.

We adopt this model in the rest of the paper, and shorten "the probability of a path" to "the path" when there is no ambiguity.

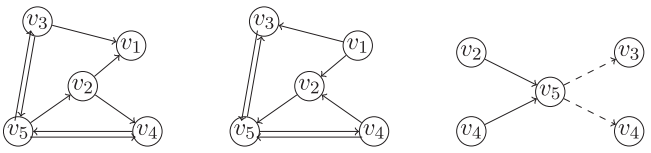


Fig. 2. Example network and two-way path.

Table 1
SimRank of 2nd iteration.

	v_1	v_2	v_3	v_4	v_5
v_1	1.000	0	0	0.123	0.155
v_2	0	1.000	**0.360**	**0.180**	0
v_3	0	**0.360**	1.000	0.180	0
v_4	0.123	**0.180**	0.180	1.000	0.049
v_5	0.155	0	0	0.049	1.000

Definition 4 (Top-k similarity join based on SimRank)**.** Given a network G and an integer k, top-k similarity join based on SimRank finds a set of k node pairs $\mathcal{K}$ such that for $(v, u) \in \mathcal{K}, v \neq u$ and $(v', u') \in V \times V \setminus \mathcal{K}, v' \neq u'$, $S(v, u) \geq S(v', u')$, and ties are broken arbitrarily[1].

Example 1. Consider the example network in Fig. 2(a), and assume $k = 2$, $\xi = 2$ and $c = 0.360$. According to their SimRank values in the second iteration in Table 1, we can verify that (v_2, v_3) and (v_2, v_4) have the highest SimRank. Therefore, top-2 similarity join based on SimRank returns $\mathcal{K} = \{(v_2, v_3), (v_2, v_4)\}$ as the answer.

Note that we focus on solve the problem exactly and adopt the in-memory setting while describing the algorithm.

3. Incremental top-*k* SimRank

Recall that by summarizing all first-meeting two-way paths which start at u and v and end within ξ steps, we can compute SimRank. However, to enumerate all possible first-meeting two-way paths is rather expensive. As a consequence, SRK-Join [14] proposes to compute SimRank as a difference between meeting and multi-meeting two-way paths. Inspired by SRK-Join, we also choose not to compute SimRank in a holistic way. Instead, we compute cumulative partial SimRank by one-way paths and keep track of the top-k results simultaneously.

3.1. Partial SimRank by one-way paths

Further to Definition 2, we denote the set of multi-meeting two-way paths, which start at (v, u) and end at (x, x) at step-ℓ as $\{(v, u) \overset{\ell}{\underset{ml}{\rightsquigarrow}} (x, x)\}$. According to the set containment relation among the two-way paths, we have the following equation.

Proposition 1. $P(\{(v, u) \overset{\ell}{\underset{fm}{\rightsquigarrow}} (x, x)\}) = P(\{(v, u) \overset{\ell}{\rightsquigarrow} (x, x)\}) - P(\{(v, u) \overset{\ell}{\underset{ml}{\rightsquigarrow}} (x, x)\})$.

Example 2. Consider the network in Fig. 2(a). Fig. 2(c) illustrates all possible meeting two-way paths that start at (v_2, v_4) and end within 2 steps. Specifically, there exists only 1 first-meeting two-way path $\{\{(v_2, v_4) \to (v_5, v_5)\}\}$, and 2 multi-meeting two-way paths $\{\{(v_2, v_4) \to (v_5, v_5) \to (v_3, v_3)\}, \{(v_2, v_4) \to (v_5, v_5) \to (v_4, v_4)\}\}$. They together make the set of meeting two-way paths.

Next, we continue to introduce the concept of one-way path.

Definition 5 (One-way path)**.** A node sequence $\mathcal{OP} = \{v_1 \to v_2 \to \cdots \to v_{\ell+1}\}$ is called a one-way path if it satisfies:

(1) $\forall i \in [1, \ell + 1]$, $v_i \in V$; and
(2) $\forall i \in [1, \ell + 1)$, $\langle v_{i+1}, v_i \rangle \in E$.

The probability of one-way path is defined as $P(\mathcal{OP}) = \prod_{i=1}^{\ell} \frac{\sqrt{c}}{|I(v_i)|}$. It is easy to see that the probability of a two-way path equals the multiplication of the probabilities of two corresponding one-way paths. That is, $P(\{(v_1, u_1) \overset{\ell}{\rightsquigarrow} (v_{\ell+1}, u_{\ell+1})\}) = P(\{v_1 \overset{\ell}{\rightsquigarrow} v_{\ell+1}\}) \cdot P(\{u_1 \overset{\ell}{\rightsquigarrow} u_{\ell+1}\})$, where $\{v_1 \overset{\ell}{\rightsquigarrow} v_{\ell+1}\}$ denotes the set of one-way paths that start at v_1 and end at $v_{\ell+1}$ at step-ℓ.

[1] When there is a tie, we choose the nodes with smaller *ids*. This method for breaking tie is adopted for all algorithms in experiment so that the results of different methods are identical.

Before delving into the method details, we differentiate two notations. Given two nodes v and u,

- *at-step-ℓ* SimRank, denoted by $\tilde{S}_\ell(v,u)$, equals the probability of first-meeting two-way paths that start at (v, u) and end *at* step-ℓ;
- *within-ℓ-step* SimRank, denoted by $S_\ell(v, u)$, equals the probability of first-meeting two-way paths that start at (v, u) and end *within* ℓ steps.

It is easy to verify that $\tilde{S}_\ell(v,u) = S_\ell(v,u) - S_{\ell-1}(v,u)$, which implies that $\tilde{S}_\ell(v,u)$ is the cumulation of SimRank for step-ℓ. Subsequently, we may use "partial SimRank " to refer "within-ℓ-step SimRank " when context is clear.

Definition 6 (Second-meeting probability)**.** The σ-step second meeting probability, denoted by $\rho(x, \sigma)$, is the sum of probabilities of all two one-way paths that satisfy

(1) starting at (x, x);
(2) ending at the same node;
(3) no other meeting node; and
(4) length is *exactly* σ [2].

To compute SimRank in a cumulative way, we need to derive $\tilde{S}_\ell(v,u)$ in each step along with the random walk for every candidate node pair. Theorem 1 describes how to derived $\tilde{S}_\ell(v,u)$.

Theorem 1.

$$\tilde{S}_\ell(v,u) = \sum_{x\in V} P(\{(v,u) \overset{\ell}{\rightsquigarrow} (x,x)\}) - \sum_{i\in[1,\ell-1], y\in V} P(\{(v,u) \overset{i}{\rightsquigarrow} (y,y)\}) \cdot \rho(y, \ell - i).$$

Proof. Since $\tilde{S}_\ell(v,u) = \sum_{x\in V} P(\{(v,u) \overset{\ell}{\rightsquigarrow} (x,x)\}) - \sum_{x\in V} P(\{(v,u) \underset{ml}{\overset{\ell}{\rightsquigarrow}} (x,x)\})$, we just need to prove that

$$\sum_{x\in V} P(\{(v,u) \underset{ml}{\overset{\ell}{\rightsquigarrow}} (x,x)\}) = \sum_{i\in[1,\ell-1], y\in V} P(\{(v,u) \overset{i}{\rightsquigarrow} (y,y)\}) \cdot \rho(y, \ell - i).$$

Recall that $P(\{(v,u) \overset{i}{\rightsquigarrow} (y,y)\})$ is the probability of all meeting two-way paths that start at (v, u) and end at (y, y) at step-i. When multiplied by $\rho(y, \ell - i)$, it advances $\ell - i$ steps from y and meets again. Then it is equivalent to all multi-meeting two-way paths which have their *penultimate* meeting at y at step-i and length is equal to ℓ. Therefore, the theorem holds. □

Example 3. Consider Fig. 2, and assume $\xi = 2,\ c = 0.360$.

$$\begin{aligned}\sum_{x\in V} P(\{(v_2,v_4) \overset{2}{\rightsquigarrow} (x,x)\}) &= P(\{(v_2,v_4) \overset{2}{\rightsquigarrow} (v_3,v_3)\}) + P(\{(v_2,v_4) \overset{2}{\rightsquigarrow} (v_4,v_4)\}) \\ &= 0.180 \times 0.090 + 0.180 \times 0.090 = 0.032.\end{aligned}$$

The slashed lines represent all two-way paths that can contribute to $\rho(v_5, 1)$. Thus, $\rho(v_5,1) = P(\{(v_5,v_5) \to (v_3,v_3)\}) + P(\{(v_5,v_5) \to (v_4,v_4)\}) = 0.180$. Hence, the probability of all multi-meeting two-way paths, which start at (v_2, v_4) and end at step-2, is

$$\begin{aligned}\sum_{x\in V} P(\{(v_2,v_4) \underset{ml}{\overset{2}{\rightsquigarrow}} (x,x)\}) &= P(\{(v_2,v_4) \overset{1}{\rightsquigarrow} (v_5,v_5)\}) \cdot \rho(v_5,1) \\ &= 0.180 \times 0.180 = 0.032.\end{aligned}$$

Consequently, $\tilde{S}_2(v_2,v_4) = 0.032 - 0.032 = 0$.

3.2. An incremental algorithm

To compute partial SimRank at step-ℓ as Theorem 1, it is necessary for every pair of nodes to derive the first and multi-meeting probabilities. However, trying to pair all nodes can be fairly time-consuming. We present a method to find for the pairable nodes of a given node quickly.

Finding pairable nodes. Definition 7 qualifies the nodes that can be used to be paired with a given node at a certain step.

Definition 7 (Step-ℓ pairable nodes)**.** Given a node v, step-ℓ pairable nodes of v are the starting nodes of a one-way path that meets another one-way path starting from v at step-ℓ, i.e., $\Pi_\ell(v) = \{u | \{x | v \overset{\ell}{\rightsquigarrow} x\} \cap \{y | u \overset{\ell}{\rightsquigarrow} y\} \neq \emptyset, u \in V \setminus \{v\}, x, y \in V\}$.

To efficiently find the pairable nodes of a given node, we rely on two inverted list-like indices that are built online via random walks. Specifically, for each node $v \in V$, we employ $NP(v, \ell)$ to record the nodes that v can reach at step-ℓ. We also

[2] Note that this condition is slightly different from that used in SRK-Join [14], which requires the path length is *no larger* than σ.

record the probabilities of the corresponding sets of one-way paths. In other words, each element in $NP(v, \ell)$ has the form of $(x, P(\{v \overset{\ell}{\rightsquigarrow} x\}))$. In the mean time, we also maintain $RNP(x, \ell)$ such that each element therein is $(v, P(\{v \overset{\ell}{\rightsquigarrow} x\}))$. In short, $NP(v, \ell)$ is used to find the ending nodes of the one-way paths that start from v at step-ℓ, while $RNP(x, \ell)$ is leveraged to retrieve the starting nodes of the one-way paths that can reach x at step-ℓ. Leveraging them, we are able to derive a pair of one-way paths starting at v and u, respectively, and both ending at x at step-ℓ.

Algorithm details. We encapsulate the major steps for computing partial SimRank in Algorithm 1. The algorithm takes as

Algorithm 1: IncrementalSimRank $(R, \mathcal{S}, \ell)$.

Input: R is a set of nodes; $\mathcal{S}_{\ell-1}$ is SimRank of node pairs within $\ell - 1$ steps; ℓ is an integer of step.
Output: $\mathcal{K}$ is a set of k node pairs with highest within-ℓ-step SimRank .

```
1  K ← ∅ ;                                                    /* K is a priority queue */
2  foreach node v ∈ R do
3      foreach element (x, P_v) ∈ NP(v, ℓ) do
4          foreach element (u, P_u) ∈ RNP(x, ℓ) do
5              P_x ← P_v · P_u;
6              if u ∉ Π_ℓ(v) then                                   /* not initialized */
7                  S̃_ℓ(v, u) ← P_x, Π_ℓ(v) ← Π_ℓ(v) ∪ {u};
8              else S̃_ℓ(v, u) ← S̃_ℓ(v, u) + P_x;
9              append (x, P_x) to M(v, u, ℓ) ;          /* for computing multi-meeting probabilities */
10     foreach node u ∈ Π_ℓ(v) do
11         j ← 1;
12         while j < ℓ do
13             foreach element (x, P_x) in M(v, u, j) do
14                 S̃_ℓ(v, u) ← S̃_ℓ(v, u) − P_x · SecondMeetingProbability(x, ℓ − j);
15             j ← j + 1;
16         S_ℓ(v, u) ← S_{ℓ−1}(v, u) + S̃_ℓ(v, u);
17         insert (v, u) into K;
18         while K.size() > k do
19             K.pop();
20 return K
```

input a set of nodes R, the partial SimRank within $\ell - 1$ steps, as well as an integer k, and outputs a set of k node pairs with the highest partial SimRank within ℓ steps. In particular, we first initialize a priority queue for keeping the current best k node pairs (Line 1). Then, for each node v in R, it first finds its pairable nodes for step-ℓ leveraging NP and RNP (Lines 3 and 4). We then use P_x to collect the meeting probability of v and u meet at step-ℓ (Lines 3 and 9). In particular, if u is not seen before, i.e., $\tilde{S}_\ell(v, u)$ is not initialized, we set it to P_x and put the new pairable node u into $\Pi_\ell(v)$; otherwise, we increment $\tilde{S}_\ell(v, u)$ by P_x. In addition, we append the meeting node with its meeting probabilities to $M(v, u, \ell)$ for future use, which is a global data structure for calculating multi-meeting probabilities of v and u.

After calculating meeting probabilities, based on Theorem 1, we have to minus the multi-meeting probabilities in order to derive at-step-ℓ SimRank. In particular, we reuse $M(v, u, j)$, $(1 \le j < \ell)$, calculated in previous steps. For each node $u \in \Pi_\ell(v)$, we enumerate every element (x, p_x) in $M(v, u_i, j)$, and let p_x multiply $\rho(x, \ell - j)$ (Line 14). Thus, we have the probability of all two-way paths that start at (v, u), penultimately meet at x at step-j, and meet again at step-ℓ. By subtracting all these multi-meeting two-way paths, we get at-step-ℓ partial SimRank of (v, u) (Lines 11–15). In addition, adding it to $S_{\ell-1}(v, u)$ produces the within-ℓ-step SimRank (Line 16). Lastly, we put the node pairs into the priority queue along with the update SimRank, and then return the k node pairs with highest within-ℓ-step SimRank (Line 20).

Correctnessandanalysis. Based on Theorem 1, Algorithm 1 correctly computes the at-step-ℓ SimRank, and hence, based on the input $\mathcal{S}_{\ell-1}(v, u)$, the within-$\ell$-step SimRank. For a node v, there are $d^{2\xi}$ pairable nodes. The time complexity is $O(|R|d^{2\xi})$.

Example 4. Considering calculating SimRank related to v_1 in the second iteration with $c = 0.360$. Table 2 shows part of intermediate parameters related to v_1. In the second iteration, IncrementalSimRank first finds out that $NP(v_1, 2) = \{(v_5, 0.360)\}$. After discovering $(v_4, 0.180)$ in $RNP(v_5, 2)$, it computes $P(\{(v_1, v_4) \overset{2}{\rightsquigarrow} (v_5, v_5)\}) = 0.065$. Then it scans $M(v_1, v_4, 1)$ and finds $(v_2, 0.090)$ which was already stored in the first iteration. Thus it calculates $\rho(v_2, 1) = 0.360$ and $P(\{(v_1, v_4) \overset{1}{\rightsquigarrow} (v_2, v_2)\}) \cdot \rho(v_2, 1) = 0.032$. Thus $S_2(v_1, v_4) = P(\{(v_1, v_4) \overset{2}{\rightsquigarrow} (v_5, v_5)\}) - P(\{(v_1, v_4) \overset{1}{\rightsquigarrow} (v_2, v_2)\}) \cdot \rho(v_2, 1) = 0.032$. Since we have already calculated that $\tilde{S}_1(v_1, v_4) = 0.090$ in the first iteration, thus $S_2(v_1, v_4) = \tilde{S}_1(v_1, v_4) + \tilde{S}_2(v_1, v_4) = 0.123$.

Table 2
Intermediate parameters related to v_1.

$NP(v_1, 2)$	$(v_5, 0.360)$
$RNP(v_5, 2)$	$(v_1, 0.360)$, $(v_4, 0.180)$, $(v_5, 0.270)$
$M(v_1, v_4, 1)$	$(v_2, 0.090)$

4. Iterative batch pruning framework

In this section, we present an iterative batch pruning framework for top-k similarity joins based on SimRank. It adopts the incremental method to derive SimRank, and excludes unpromising nodes after every iteration.

4.1. Join framework

In general, our framework comprises several iterations, in each of which the random walks for deriving SimRank advance one step further. Thus, there are at most ξ iterations in the algorithm. Particularly, in the ℓ-th iteration, we compute at-step-ℓ SimRank $\tilde{S}_\ell$ for candidate node pair (v, u), and also within-ℓ-step SimRank $S_\ell(v, u)$ by consolidating the results with previous iterations. Afterwards, unpromising nodes are pruned such that no node pair involving these nodes can be in the top-k results. That is, if the upper bound of SimRank involving a node is less than the current k-th largest S_ℓ, the node is determined to be *unpromising*. Proceeding in this way, we obtain the top-k node pairs as well as the corresponding SimRank values in the end of the ξ-th iteration. The idea above is expressed by Algorithm 2.

Algorithm 2: IterativePruneTopk (G, k, c, ξ).

Input: G is a network; k is an integer;c is decay factor; ξ is the maximum length of random walk.
Output: $\mathcal{K}$ is a set of k node pairs with largest SimRank .

```
1  build data structure NP and RNP;
2  ℓ ← 1, S ← ∅, R ← V;
3  while ℓ < ξ do
4      K ← IncrementalSimRank(R, S, ℓ);
5      σ ← the smallest partial SimRank in K;
6      foreach node v remaining in R do
7          U ← UpperBound(v, ℓ);
8          if U < σ then remove v from R;
9      ℓ ← ℓ + 1;
10 K ← IncrementalSimRank(R, S, ℓ);
11 return K
```

Algorithm 2 takes as input a network G, an integer k, decay factor c, and the maximum length of random walk ξ, and produces a set of k node pairs with the highest SimRank. Specifically, it first builds the indices of NP and RNP, and initializes step-control parameter ℓ, matrix $\mathcal{S}$ for storing current SimRank, and the set of remaining nodes R (Lines 1 and 2). Then, we start the iterations (Lines 3–9). As we discussed in Section 3, IncrementalSimRank incrementally computes k node pairs with the highest partial SimRank for the current step-ℓ. The intermediate results of SimRank are kept in $\mathcal{S}$ for the use of subsequent iterations. Recall that we employ a priority queue to maintain the top-k results. Thus, it is straightforward to obtain the current kth largest partial SimRank, which is put in σ as the threshold for pruning shortly (Line 5). Then, for each node v remained in R, we derive an upper bound U of its SimRank with any node in the network (Line 7). We will further discuss the upper-bound in the next section. If U is less than the current threshold σ, v is guaranteed to be not a part of the top-k results, and hence, can be safely excluded (Line 8). After $\xi - 1$ iterations, we exit to compute the final results by calling IncrementalSimRank again (Line 10).

Notice that the depth of iteration is synchronous with the length of the paths being calculated, i.e., paths of step-ℓ are calculated in ℓth iteration. For the computation of the algorithm, we use "ℓth iteration"; for the paths, we use "paths of step-ℓ" in the rest of the paper.

Correctness and complexity. The upper bound is the key to Algorithm 2. The validity of the upper bound guarantees the correctness of the algorithm. We proceed to complexity analysis. There are d^ξ one-way paths that start from one node, the complexity of pre-computation is $O(nd^\xi)$. Assuming $|R|$ nodes remained at last, then the complexity of pruning part is $\max\{O(|R|d^{2\xi}, O(UB)\}$ in which the complexity of UpperBound is denoted by $O(UB)$. Later we can see that $O(UB)$ is much less than $O(|R|d^{2\xi})$, thus the final complexity of the pruning process is $O(|R|d^{2\xi})$ with experimental results showing that $|R| \ll n$. The space complexity is $O(nd^\xi)$ for storing NP and RNP.

Algorithm 2 tries to exclude as many as possible unpromising nodes in every iteration before ℓ reaches the step limit ξ. It is immediate that the more nodes removed, the less candidates survive till the last, and hence, the faster the algorithm.

Table 3
Example of constructing super node.

	v_1	v_2	v_3	v_4	v_5	$\mathcal{V}_s$
v_1	0	0	0	0	0	{(0,0),(0,0)}
v_2	0	**0.054**	**0.054**	0.027	0	{(v_2, 0.054),(v_3, 0.054)}
v_3	**0.108**	0	0	0.054	**0.081**	{(v_1, 0.108),(v_5, 0.081)}
v_4	**0.108**	0	0	0	**0.081**	{(v_1, 0.108),(v_5, 0.081)}
v_5	0	**0.162**	**0.162**	0.081	0.054	{(v_2, 0.162),(v_3, 0.162)}

Thus, in the sequel, we will investigate how to derive a tight upper bound to prune irrelevant nodes while ensuring no false negatives.

4.2. Upper-bounding technique

Recall that SimRank is equivalent to the summation of probabilities of first-meeting two-way paths with lengths $1, 2, \ldots, \xi$. The maximum probability of all possible two-way paths at length ℓ is c^ℓ. If we consider the probability of all two-way paths as the maximum probability, we have a geometric progression with the first term and the ratio being both c, i.e. $c, c^1, c^2, \ldots, c^\xi$. Obviously, SimRank between any two nodes is always less than the summation of this geometric progression. Hence, this makes an immediate upper bound of SimRank, and we denote it by geoUB. geoUB is a commonly used upper bound in [18] for error bounding.

Nevertheless, we observe that geoUB ignores the different neighborhoods of different nodes, and this "little caution" leaves itself not a good hesitation of the exact SimRank . It is contended that a good upper bound strikes a balance between being close to the exact value and being easily calculated. For this propose, we design an upper bound snbUB that can be easily derived using existing information. The basic idea is to underestimate the multi-meeting probabilities, such that the resultant estimation makes an upper bound of the real SimRank. In the sequel, we start with the upper bound of a single node pair.

A straightforward way to approximate SimRank is to replace first-meeting probabilities with meeting probabilities [13].

Lemma 1. $S(v,u) \le \sum_{\ell=1}^{\xi} \sum_{x \in V} P(\{(v,u) \overset{\ell}{\rightsquigarrow} (x,x)\})$.

Lemma 1 provides an upper bound for single node pairs (v, u). To make it work for batch pruning under the aforementioned framework, we have to discriminate the neighborhoods of different pairable nodes. However, online enumeration is almost impractical. Consequently, we wish that there is a node coming with the following feature, i.e., for every node v, the SimRank between v and the node is not less than the maximum SimRank between v and any others. Thus the SimRank between v and the node makes up the upper bound for SimRank of node pairs involving v. To implement this, we define *super nodes*.

Definition 8 (Super node)**.** Consider a node v in a set of nodes R. The super node $\mathcal{V}_s$ of v is a virtual node such that its probability of reaching x at step-ℓ satisfies $P(\{\mathcal{V}_s \overset{\ell}{\rightsquigarrow} x\}) = \max\{P(\{u \overset{\ell}{\rightsquigarrow} x\}) | u, x \in R_v \wedge R_v = R \setminus \{v\}\}$.

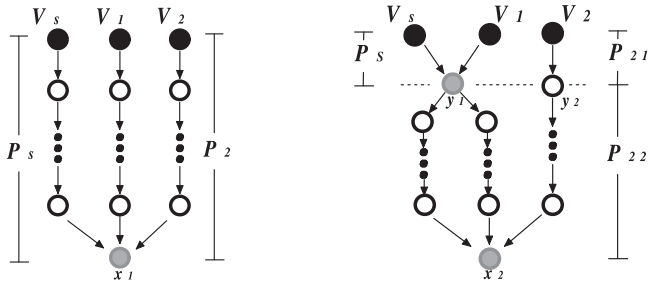
Example 5. Consider Fig. 2. Table 3 shows the probabilities of a node v_j reaches v_i at step-3, i.e., $p(\{v_j \overset{3}{\rightsquigarrow} v_i\})$, in the cell at ith row and jth column. Specifically, we record the top-2 largest probabilities of reaching a specific node in the column under $\mathcal{V}_s$, together with the starting nodes. For instance, as to v_2 (2nd row), we record {(v_2, 0.054),(v_3, 0.054)}. When we derive the upper bound for v_2, we need to calculates $P(\{(v_2, \mathcal{V}_s) \overset{3}{\rightsquigarrow} (v_2, v_2)\})$. It is easy to obtain that the largest probability of other nodes reaching v_2 at step-3 is 0.054, which is from v_3. Note that for v_2 at step-3, the largest probability of other nodes reaching v_2 equals the probability of itself reaching v_2. Although the two probabilities are identical, they imply two different sets of one-way paths.

Theorem 2. *Consider a node v in a set of nodes R and its super node $\mathcal{V}_s$. For any node $u \in R_v$, $S(v, \mathcal{V}_s) \ge S(v,u)$, where $R_v = R \setminus \{v\}$.*

Proof. Given a node v_1 and the super node $\mathcal{V}_s$, we need to prove that the SimRank between v_1 and $\mathcal{V}_s$ is not less than SimRank between v_1 with other nodes. Without loss of generality, we consider a node v_2 ($v_2 \neq v_1$). Recall that SimRank equals the summation of probabilities of all first-meeting two-way paths. Hence, we compare the probabilities of first-meeting two-way paths which start from $(v_1, \mathcal{V}_s)$ and (v_1, v_2) separately. We show that for each first-meeting two-way path that start at (v_1, v_2), there exists one first-meeting two-way path that starts at $(v_1, \mathcal{V}_s)$ of which the probability is not less than the one starts at (v_1, v_2).

Suppose (v_1, v_2) first meet at $\{x_1, x_2, \ldots\}$. As $\mathcal{V}_s$ can reach every node at every step, it certainly can reach $\{x_1, x_2, \ldots\}$. Fig. 3 shows two cases of three one-way paths.

In Fig. 3(a), (v_1, v_2) first meet at x_1, while $(v_1, \mathcal{V}_s)$ also first meet at x_1. v_2 and $\mathcal{V}_s$ reach x_1 with probabilities of P_2 and P_s, and $P_s \le P_2$. Thus, $P(\{(v_1, \mathcal{V}_s) \to \cdots \to (x_1, x_1)\}) \ge P(\{(v_1, v_2) \to \cdots \to (x_1, x_1)\})$.



(a) First-Meeting (b) Multi-Meeting

Fig. 3. Two cases of 3 one-way paths.

In Fig. 3(b), (v_1, v_2) still first meet at x_2, but $(v_1, \mathcal{V}_s)$ no longer first meet at x_1. Suppose $(v_1, \mathcal{V}_s)$ first meet at y_1 before x_1 at step-$i (1 \le i < \ell)$, also $\mathcal{V}_s, v_2$ reaches y_1, y_2 at step-i with probability P_s, P_{21} and y_2 reaches x_2 at step-$(\ell - i)$ with probability P_{22}. Because $\{((v_1, \mathcal{V}_s) \to \cdots \to (x_2, x_2)\}$ is not a first-meeting two-way path, its probability contributes zero to $S(v_1, \mathcal{V}_s)$. In the mean while, however, $\{((v_1, \mathcal{V}_s) \to \cdots \to (y_1, y_1)\}$ can be guaranteed a first-meeting two-way path. Thus, it contributes its probability to $S(v_1, \mathcal{V}_s)$. Because $y_1 \neq y_2$ and $P_s \ge P_{21} \ge P_{21} \cdot P_{22}$, $P(\{(v_1, \mathcal{V}_s) \to \cdots \to (y_1, 1_1)\}) \ge P(\{(v_1, v_2) \to \cdots \to (x_2, x_2)\})$. Therefore, the theorem holds. □

It is easy to see that $S(v, \mathcal{V}_s)$ is an upper bound of the SimRank of any node pair involving v. However, this upper bound is not easy to calculate, since computing $S(v, \mathcal{V}_s)$ still requires exact values of multi-meeting probabilities for all $\ell \in \{1, \ldots, \xi\}$. To resolve the issue, we propose to loosen the upper bound by underestimating the multi-meeting probabilities. Our strategy is to leverage existing second-meeting probabilities computed in previous iterations. Lemma 2 shows an upper bound of the at-step-ℓ SimRank.

Lemma 2. *Consider step-ℓ and an integer $t \in [1, \ell - 1)$,*

$$\tilde{S}_\ell(v, \mathcal{V}_s) \le \sum_{x \in V, i \in [1,\ell]} P(\{(v, \mathcal{V}_s) \overset{i}{\rightsquigarrow} (x, x)\}) - \sum_{y \in V, j \in [1,t]} P(\{(v, \mathcal{V}_s) \overset{\ell-j}{\rightsquigarrow} (y, y)\}) \cdot \rho(y, j).$$

Proof. By definition, $\tilde{S}_\ell(v, \mathcal{V}_s)$ equals

$$\sum_{x \in V, i \in [1,\ell]} P(\{(v, \mathcal{V}_s) \overset{i}{\rightsquigarrow} (x, x)\}) - \sum_{y \in V, j \in [1,\ell-1]} P(\{(v, \mathcal{V}_s) \overset{\ell-j}{\rightsquigarrow} (y, y)\}) \cdot \rho(y, j).$$

The latter summation is no smaller than $\sum_{y \in V, j \in [1,t]} P(\{(v, \mathcal{V}_s) \overset{\ell-j}{\rightsquigarrow} (y, y)\}) \cdot \rho(y, j)$. Substituting the two terms, hence, the lemma follows. □

Denote as $\hat{S}^t_\ell(v, \mathcal{V}_s)$ the right part of the inequation in Lemma 2. Then, we have the following estimation of $S(v, \mathcal{V}_s)$.

Definition 9 (Super node based estimation)**.** For step-ℓ, the super node based SimRank estimation of any node pair involving $v \in R$ is

$$\mathsf{snbUB}_\ell(v) = \sum_{m=1}^{\xi} \hat{S}_m^{\min\{\ell-1, m-1\}}(v, \mathcal{V}_s). \tag{3}$$

For all possible ℓ, $\mathsf{snbUB}_\ell(v)$ actually provides a series of upper bounds. As we have more second-meeting probabilities available when the iteration goes further, it gradually draw near to $\max\{S(v, u) | u \in R_v\}$.

Proposition 2. $\mathsf{snbUB}_\ell(v) \ge \mathsf{snbUB}_{\ell+1}(v)$, $\ell \in \{1, \xi - 1\}$.

Based on Lemma 2, the super node based estimation provides an gradually tight upper bound of $S_\ell(v, u)$, $u \in R_v$, as stated below.

Theorem 3. *Assume the top-k threshold of partial SimRank at step-ℓ is σ. If $\mathsf{snbUB}_\ell(v) < \sigma$, v cannot make an answer of the top-k node pairs.*

Proof. On one hand, as σ is the kth largest partial SimRank at step-ℓ, SimRank of these k pairs of nodes will never less than σ. On the other hand, $\mathsf{snbUB}_\ell(v)$ provides an upper bound of the SimRank of any node pairs involving v. That is to say, the SimRank of any pair involving v will never exceed $\mathsf{snbUB}_\ell(v)$. Therefore, v cannot compose a node pair in the top-k results. □

Algorithm 3: UpperBound (v, ℓ).

Input: v is a reference node from R, ℓ is an integer;
Output: U is the upper bound of SimRank of node pairs involving v.

```
1  U ← 0;
2  if ℓ = 1 then
3      foreach i = [1, ..., ξ] do
4          U_i ← 0;
5          foreach (x, P) in NP(v, i) do
6              P_x ← P · P_{V_s};
7              M(v, V_s, i) ← M(v, V_s, i) ∪ {(x, p_x)};
8              U_i ← U_i + p_x;
9          i ← i + 1;
10 else
11     foreach i = [ℓ, ..., ξ] do
12         foreach (x, P) in M(v, V_s, i − (ℓ − 1)) do
13             P_y ← P · ρ(x, ℓ − 1) ;        /* ρ(x, ℓ − 1) already calculated */
14             U_i ← U_i − P_y;
15         i ← i + 1;
16 foreach i = [1, ..., ξ] do U ← U + U_i ;
17 return U
```

Table 4
Paths of v_1 and $\mathcal{V}_s$.

Step	v_1		$\mathcal{V}_s$
Step-1	v_2	0.300	(v_1, 0.300),(**v_4, 0.300**)
	v_3	0.300	(v_1, 0.300),(**v_5, 0.300**)
Step-2	v_5	0.360	(v_1, 0.360),(**v_5, 0.270**)
Step-3	v_3	0.108	(v_1, 0.108),(**v_5, 0.081**)
	v_4	0.108	(v_1, 0.108),(**v_5, 0.081**)
Step-4	v_2	0.032	(v_1, 0.032),(**v_5, 0.024**)
	v_5	0.091	(v_1, 0.091),(**v_5, 0.073**)
Step-5	v_3	0.029	(v_1, 0.029),(**v_5, 0.022**)
	v_4	0.029	(v_1, 0.029),(**v_5, 0.022**)
	v_5	0.019	(**v_2, 0.044**),(v_3, 0.044)

Algorithm details. We describe how to compute the upper bound with pseudocode in Algorithm 3 . Algorithm 3 takes as input a node v, and an integer ℓ of step, and produces the upper bound of SimRank of node pairs involving v. Then, we compute the second part of the upper bound by judiciously derive the meeting and multi-meeting probabilities at step-ℓ of v and its super node $\mathcal{V}_s$. More specifically, Lines 2–9 takes care of the case when $\ell = 1$, while Lines 11–15 deals with the other scenarios. In the former case when $\ell = 1$, we calculate the meeting probability of $(v, \mathcal{V}_s)$. In the latter case when $\ell \neq 1$, we subtract part of multi-meeting probabilities which can be easily derived by calculated $\rho(x, \ell - 1)$. Eventually, Line 16 combines the probabilities together by Definition 9, which is returned afterwards as the upper bound of SimRank involving v.

Correctness and complexity.Based on Theorem 2, Algorithm 3 correctly produces an upper bound of SimRank involving node v. As to time complexity, since node v and the super node $\mathcal{V}_s$ can meet up to d^{ξ} times, the time complexity of UpperBound is in $O(|R|d^{\xi})$.

Example 6. Continue with Example 4 and calculate the upper-bound of node v_1 when $c = 0.360, \xi = 5$. Table 4 shows $NP(v_1, i)$ with $1 \le i \le \xi$ and the values of super node $\mathcal{V}_s$ takes for v_1 (highlighted). For simplicity, we use $\mathcal{P}^{\ell}_{\{X\}}$ to denote the probabilities of meeting two-way paths start at $(v_1, \mathcal{V}_s)$ end at a node in node set X at step-ℓ. We have already computed in the first iteration that $\text{snbUB}_1(v_1) = \mathcal{P}^1_{\{v_2,v_3\}} + \mathcal{P}^2_{\{v_5\}} + \mathcal{P}^3_{\{v_3,v_4\}} + \mathcal{P}^4_{\{v_2,v_5\}} + \mathcal{P}^5_{\{v_3,v_4,v_5\}} = 0.304$. In the second iteration after IncrementalSimRank, all $\rho(x, 1)$, $x \in V$ are calculated (shown in Table 5). We thus utilize the calculated $\rho(x, 1)$ to narrow $\text{snbUB}_2(v_1)$.

$$\begin{aligned}\text{snbUB}_2(v_1) = \text{snbUB}_1(v_1) &- \{\mathcal{P}^1_{\{v_2\}} \cdot \rho(v_2, 1) + \mathcal{P}^1_{\{v_3\}} \cdot \rho(v_3, 1)\} \\ &- \mathcal{P}^2_{\{v_5\}} \cdot \rho(v_5, 1) - \{\mathcal{P}^3_{\{v_3\}} \cdot \rho(v_3, 1) + \mathcal{P}^3_{\{v_4\}} \cdot \rho(v_4, 1)\} \\ &- \{\mathcal{P}^4_{\{v_5\}} \cdot \rho(v_5, 1) + \mathcal{P}^4_{\{v_2\}} \cdot \rho(v_2, 1)\} = 0.216.\end{aligned}$$

Table 5
$\rho(x, 1)$ in 2nd iteration.

Node	$\rho(x, 1)$
v_1	0.180
v_2	0.360
v_3	0.360
v_4	0.180
v_5	0.180

Table 6
Statistics of real datasets.

Dataset	$\|V\|$	$\|E\|$	d
Adolhealth	2,539	12,969	10.210
Wordnet3	67,595	88,006	1.300
Cora	225,026	714,266	3.170

Remark. An existing method [18] also uses an upper bound to assist in filtering, which is, however, conceived for a single node pair. Distinctively, our upper bound for a node ensures that once a node is disqualified, all node pairs involving it can safely be pruned. That is, rather than one pair a time, we exclude unpromising node pairs in a "batch mode".

5. Improving for large graphs

Recall in Algorithm 2 that data structures *NP* and *RNP* are built online in the first place. Nevertheless, the construction of *NP* and *RNP* may become infeasible on large graphs, due to the high complexity of space. In this case, large graphs desire specific attention. We thus propose a lazy strategy to incrementally build *NP* and *RNP* only one step forward for the remaining nodes R during each iteration.

On the other hand, however, UpperBound cannot proceed if $NP(v, j)(1 \le j \le \xi)$ is not completely known in the ith iteration. To this end, we propose to combine geoUB and snbUB, where the complete *NP* is not required. The improved upper bound of ith iteration is defined as

$$\mathsf{snbUB}_\ell(v)_{imp} = \sum_{m=1}^{\ell+1} \hat{S}_m^{\min\{\ell-1,m-1\}}(v, \mathcal{V}_s) + \sum_{m=\ell+2}^{\xi} c^m.$$

To implement this upper bound, we put it in a procedure ImprovedUpperBound, and replace Line 7 of Algorithm 2 with "$U \leftarrow \mathsf{ImprovedUpperBound}(v, \ell)$;".

Example 7. Recall the second iteration in Example 6. We have already computed in the first iteration that $\mathsf{snbUB}_1(v_1)_{imp} = \mathcal{P}^1_{\{v_2,v_3\}} + \mathcal{P}^2_{\{v_5\}} + c^3 + c^4 + c^5 = 0.347$. In the second iteration after IncrementalSimRank, $NP(v, 3)$, $RNP(v, 3)$ with $v \in R$ and all $\rho(x, 1)$ with $x \in V$ are calculated.

$$\begin{aligned}\mathsf{snbUB}_2(v_1)_{imp} = \mathsf{snbUB}_1(v_1)_{imp} - c^3 + \mathcal{P}^3_{\{v_3,v_4\}} - \{\mathcal{P}^1_{\{v_2\}} \cdot \rho(v_2, 1)\\ + \mathcal{P}^1_{\{v_3\}} \cdot \rho(v_3, 1)\} - \mathcal{P}^2_{\{v_5\}} \cdot \rho(v_5, 1) = 0.266.\end{aligned}$$

Complexity analysis. In the improved framework, only $|R|$ are remained till the end. Thus, the time complexity of the whole process then become $O(|R|d^\xi + |R|d^{2\xi}) = O(|R|d^{2\xi})$, and the space complexity is $O(|R|d^\xi)$ with $|R| \ll n$.

6. Performance evaluation

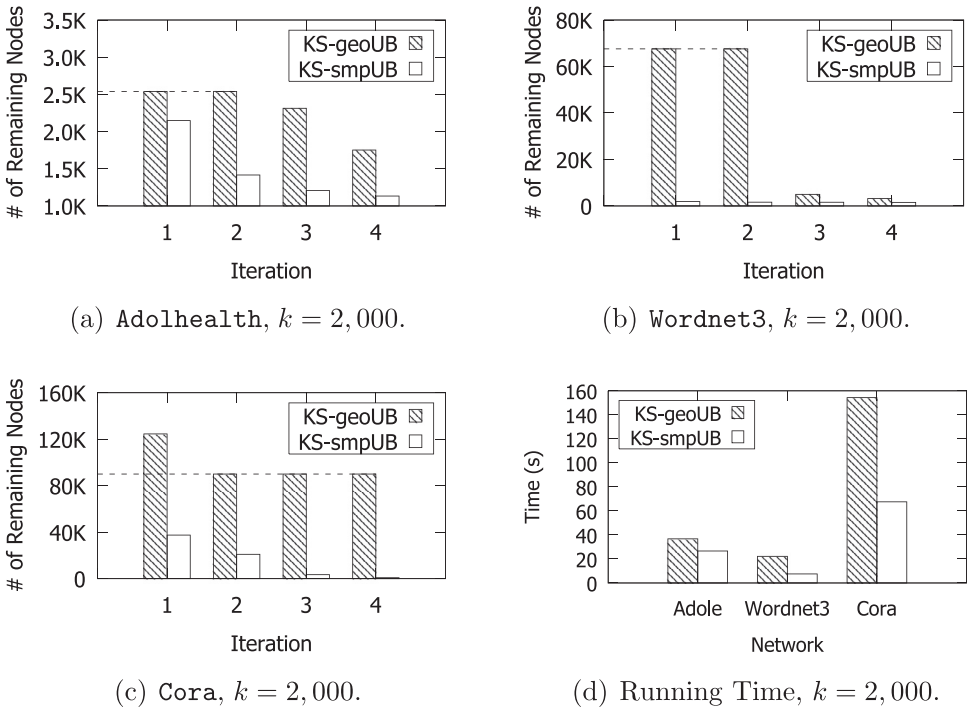
In this section, we evaluate our method in terms of pruning power, time efficiency and scalability on both real and synthetic networks.

Experiment setup. All algorithms were implemented in C++ and compiled using GCC with −O3 flag. The experiments were conducted on a Ubuntu server with 64GB RAM.

Real-life datasets. Adolhealth is a friendship network that happened in a scientific survey and an edge from node u to v represents a student u chose student v as friend in the survey. Wordnet3 is a directed network in which a node is a word and an edge from node u to v means word u is the hypernym of v. Cora is a citation network with a node denotes a scientific paper and an edge from u to v indicates that paper u cites paper v. Table 6 shows the statistics of the datasets, where $d \triangleq \frac{|E|}{|V|}$.

Synthetic datasets. We generated synthetic datasets with the maximum and minimum degree are 5 and 2, respectively. The size of these networks ranging from 0.5 M to 3.0 M.

Parameter setting. We set the decay factor $c = 0.360$, and the maximum step number $\xi = 5$ as existing work [4] if it is not otherwise specified.



(a) `Adolhealth`, $k = 2,000$. (b) `Wordnet3`, $k = 2,000$.

(c) `Cora`, $k = 2,000$. (d) Running Time, $k = 2,000$.

Fig. 4. Results for evaluating upper bounds.

Table 7
Tightness of upper bounds, $k = 2000$.

Percentage	`Adolhealth`		`Wordnet3`		`Cora`	
	geoUB	snbUB	geoUB	snbUB	geoUB	snbUB
Min	0.017	3.121×10^{-5}	0.017	5.623×10^{-9}	0.017	4.232×10^{-8}
Median	0.091	0.006	0.101	2.027×10^{-5}	0.017	1.337×10^{-5}
Max	5.000	4.889	6.112	5.999	0.017	0.001

6.1. Evaluating upper bounds

We first evaluate the pruning power of the proposed upper bound against the upper bound of geometric summation, i.e., snbUB versus geoUB. Specifically, we recorded the size of remaining nodes after each iteration, with respect to the original set of nodes in the networks. We plot the results when $k = 2000$ on the three networks in Fig. 4.

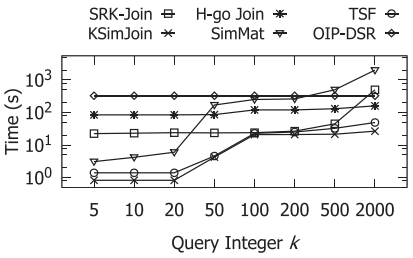
From the figure, we made the following observations. First, our upper bound works stably better then geoUB. Specifically, for geoUB on `Adolhealth` and `Wordnet3`, the remaining nodes of 1st and 2nd iterations are identical to the original set of network nodes. This indicates that geoUB does not work at these two iterations, and it only begins to play a role at 3rd iteration. On `Cora`, at first it works and pruns nodes at 1st and 2nd iterations. However, after 2nd iteration, it cannot work any further and the remaining nodes do not change. On the other hand, snbUB starts working even at the 1st iteration on all three networks; and as the iteration goes further, snbUB becomes tighter and more nodes are pruned.

Second, snbUB has greater pruning power than geoUB. This can be elaborated into two aspects. (1) snbUB works effectively even at 1st iteration. For instance, on `Wordnet3` and `Cora`, snbUB prunes more than 97.21% and 83.33% of the original nodes at 1st iteration, respectively; meanwhile, geoUB prunes 0% and 55.36%, respectively. (2) snbUB disqualifies more nodes than geoUB. The remaining nodes after 4th iteration are, respectively, of 69.00%, 4.71% and 40.01% of the original nodes for geoUB, and 44.60%, 2.27% and 9.93% for snbUB.

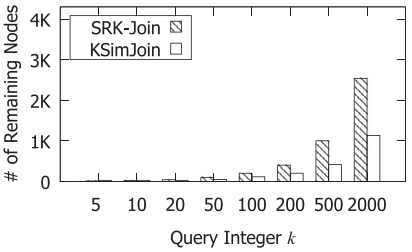
For better appreciation, Table 7 shows the difference between the upper bounds and the largest SimRank for nodes remained after 4th iteration in percentage, i.e., $\frac{UB(v)-\max\{s(v,u)|u\in R_v\}}{\max\{s(v,u)|u\in R_v\}}$, $v \in R$, and R is the set of nodes remained after 4th iteration. On `Adolhealth` and `Wordnet3`, snbUB is much closer to the SimRank, and most of the upper bounds are nearly equal to the exact values. Even for the one with largest gap, snbUB approaches closer than geoUB. Table 8 shows the percentage of nodes pruned by snbUB with respect to the original network nodes, when $k = 20$, 200, and 2000. On all the three networks, snbUB is demonstrated to be of greater pruning power.

Table 8
Percentage of nodes pruned by snbUB.

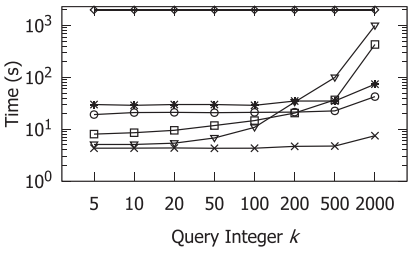
Dataset	$k = 20$ (%)	$k = 200$ (%)	$k = 2000$ (%)
Adolhealth	99.57	93.07	55.81
Wordnet3	99.97	99.77	98.33
Cora	99.88	99.88	99.88



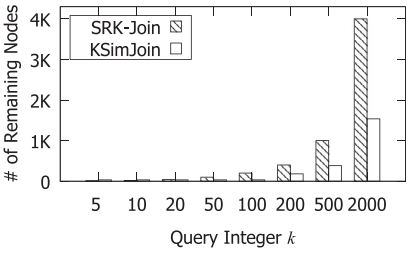
(a) Running Time, Adolhealth.



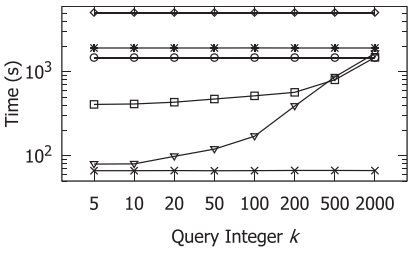
(b) Nodes after 4th iteration, Adolhealth.



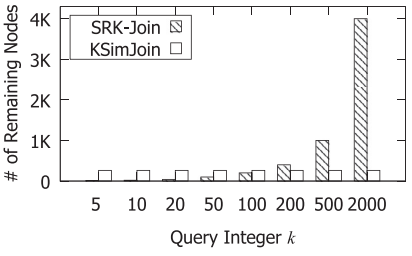
(c) Running Time, Wordnet3.



(d) Nodes after 4th iteration, Wordnet3.



(e) Running Time, Cora.



(f) Nodes after 4th iteration, Cora.

Fig. 5. Comparison with SRK-Join.

6.2. Evaluating efficiency

In this set of experiments, the following algorithms were involved in the comparison:

- KSimJoin, is the algorithm integrating the proposed framework and upper bound; and
- SRK-Join, is the state-of-the-art two-phase algorithm [14]; and
- H-go Join, is a threshold-based algorithm [18]; and
- SimMat, is a top-k search algorithm based on Sylvester equation [2]; and
- TSF, is a two-stage random-walk sampling algorithm based on one-way graphs [13]. The default parameters settings are $R_g = 100$ and $R_q = 20$ which make a trade off between the effectiveness and efficiency [13]; and
- OIP-DSR, is an fast algorithm for all-pair SimRank [17] in which we set error $\epsilon = 0.001$. It is used for a reference line to see how long it would cause if we calculated the SimRank of each pair and pick out the top-k ones.

We compared them on the 3 networks with different k, ranging from 5 to 2000. Please notice that since H-go Join is a threshold-based join method, we first perform KSimJoin and then set the kth largest SimRank as the threshold in H-go Join. Fig. 5 shows the results.

On all the three network, OIP-DSR is the slowest algorithm of all in most cases. TSF is the second fast one on small network Adolhealth, but when the network get larger, it slows down. SimMat and SRK-Join perform well for small k, as they use k to generate candidate nodes and conducts early termination; but for large k, e.g., 2000, both SimMat and SRK-Join become less efficient. H-go Join performs more efficient than SRK-Join for large k because H-go Join has to build indexes for the whole network and such indexes building activity is not cheap for small k. In comparison, our algorithm outperforms other algorithms for all query integers, by orders of magnitude in several cases. For example, on average our algorithm is 75.54% faster than SRK-Join, 97.10% faster than H-go Join, when $k = 5$; and 96.19% faster than SRK-Join, 90.06% faster than H-go Join, when $k = 2000$. The major reason is that (1) the index building in H-go Join heavily encumbers itself. Only for large k, H-go Join can benefit from its indexes and faster than SRK-Join. Besides, the filtration in H-go Join is also based on geoUB, which we have already shown to be less efficient than snbUB in Section 6.1. That is the reason of H-go Join being slower than KSimJoin for all query integers. (2) In the first phase of SRK-Join, each node needs to be encoded as a vector. During this transformation, SRK-Join has to enumerate all second meeting probabilities. For some nodes with high degree, this can be fairly expensive. Instead, our algorithm avoids much computation of second meeting probabilities by using a close upper bound to filter unpromising candidate nodes. (3) When candidate nodes are chosen, SRK-Join has to re-scan the one-way paths within 5 steps of all candidate nodes, while our algorithm only needs to derive the probability of step-5.

Since SRK-Join is also a method with node pruning strategy, we will take a closer look at SRK-Join and KSimJoin specifically. For each k on Adolhealth. SRK-Join runs slower than itself on Wordnet3, though Wordnet3 is much larger than Adolhealth. This is because the complexity of transforming nodes to vectors is $O(nd^{\xi})$, which is very sensitive to the density of networks. Comparatively, our algorithm is less sensitive to density. For instance, when $k < 100$, our algorithm on Adolhealth was very fast; when $k > 100$, it slows down but is still faster than SRK-Join.

We also recorded the number of nodes whose SimRank is precisely calculated. For SRK-Join, it is $2k$ since it generates $2k$ candidate nodes; for KSimJoin, it is the nodes that remained after 4th iteration. Generally, the number of nodes requiring precise calculation by KSimJoin is much smaller than that by SRK-Join, this benefit comes from the powerful snbUB. On the basis of that, there are still two aspects that worth noting: (1) on Adolhealth, the number of candidate nodes of SRK-Join is 2539 when $k = 2000$, i.e., the whole network. This is because SRK-Join has to generate $2k$ candidate nodes, and when $2k$ is larger than the network, the strategy of generating candidate nodes becomes useless and it is even slower than OIP-DSR. On the contrary, KSimJoin does not surfer from this situation. (2) On Cora, when k is smaller than 200, the number of nodes that need precise calculation by KSimJoin is larger than that of SRK-Join, however the running time of KSimJoin is still shorter than that of SRK-Join. This is because while pruning, KSimJoin removes many non-candidate nodes, and thus, shortens the overall time. On all the three networks, KSimJoin performs more efficient and independent from k, compared with SRK-Join and H-go Join.

In all, experimental result on six algorithms shows that KSimJoin outperforms the other five ones under all circumstances.

6.3. Evaluating effectiveness

In this experiment, we evaluate the effectiveness of our algorithms. Since for all methods based on exact SimRank, including our algorithm, the results are identical because they all compute SimRank accurately. Thus we compare our algorithm with two latest approximate techniques used in SimRank estimation [13,14], denoted as AP1 and AP2 respectively. AP1 [13] uses meeting probability instead of first meeting probability to estimate SimRank. AP2 [14] prunes some one-way paths and the error is guaranteed less than a user-defined error ϵ. In our experiment, we set $\epsilon = 0.0001$. Our evaluation adopts four widely-used measures, *precision* for overall accuracy evaluation, *NDCG* for ranking evaluation and *MAE* and *RMSE* for SimRank score error evaluation.

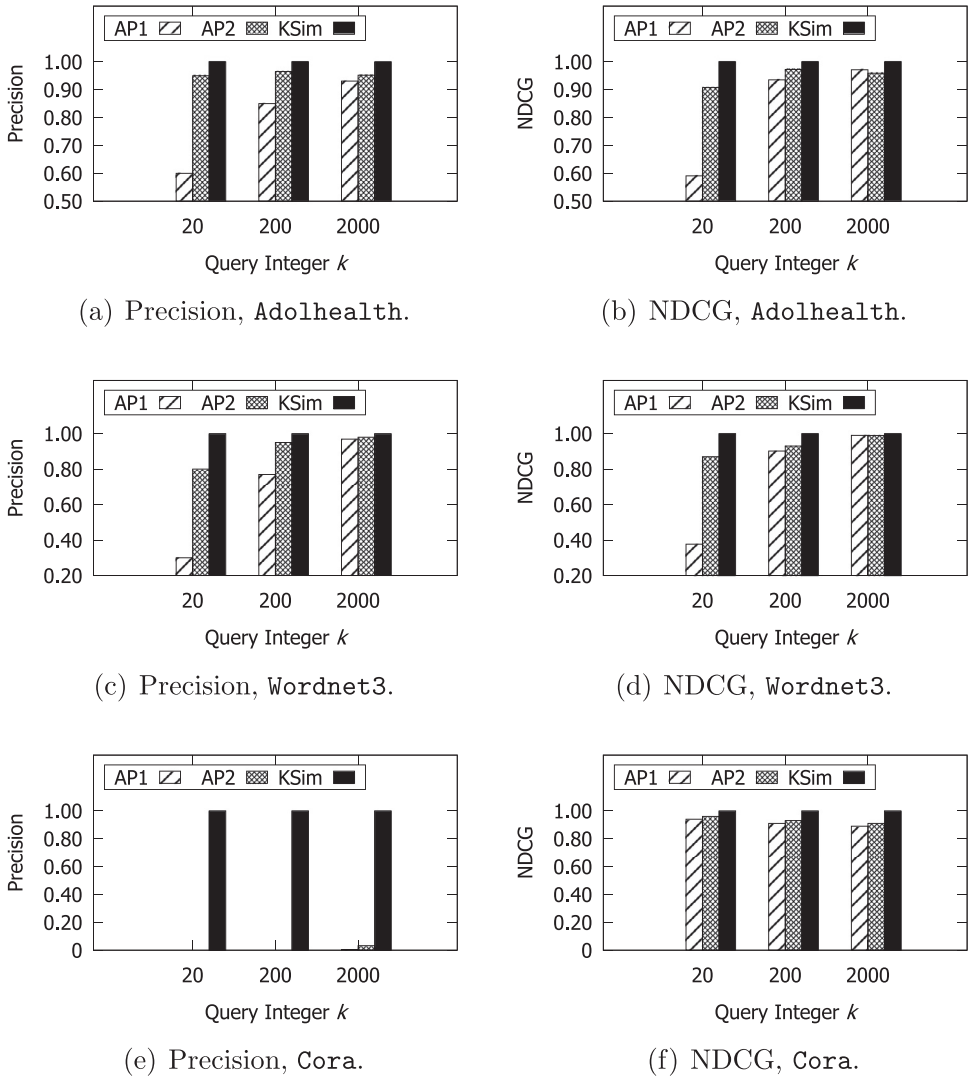
Precision is a measure to evaluate the accuracy of the algorithms. Given a query integer k, if we denote the real top-k answer pair set and the set returned by an algorithm as $\mathcal{P}_R$ and $\mathcal{P}_A$ respectively, then precision is defined as $\frac{|\mathcal{P}_R \cap \mathcal{P}_A|}{k}$.

Normalizing discounted cumulative gain(*NDCG*) is a measure to assess the ranking result. Given the top-k result, *NDCG@k* is define as

$$NDCG@k = \frac{DCG}{IDCG} = \frac{1}{IDCG} \sum_{i=1}^{k} \frac{2^{SR_i} - 1}{\log_2(i+1)},$$

where SR_i denotes the exact SimRank score of the node pair at rank i, IDCG is the normalizing factor that equals to the DCG of the real ranking.

Mean absolute error(*MAE*) *and root mean squared error*(*RMSE*) are two of the most common quantities used to comparing estimation with their exact scores. Specifically, they both measure the average magnitude of the errors in a set of estimations. The RMSE will always be larger or equal to the MAE; the greater difference between them, the greater the variance



(a) Precision, Adolhealth. (b) NDCG, Adolhealth.

(c) Precision, Wordnet3. (d) NDCG, Wordnet3.

(e) Precision, Cora. (f) NDCG, Cora.

Fig. 6. Precision and NDCG.

in the individual errors in the estimations. They are defined as follow equations:

$$MAE = \frac{1}{k}\sum_{i=1}^{k}|SR_i' - SR_i|, \qquad RMSE = \sqrt{\frac{\sum_{i=1}^{k}(SR_i' - SR_i)^2}{k}},$$

where SR_i' and SR_i denote the SimRank calculated by an algorithm and the exact SimRank.

Fig. 6 presents the precision and *NDCG@k* with $k = 20$ and 2000. We observed that AP1 is of lowest precision and NDCG in most cases and KSimJoin is of the highest precision and NDCG of the three algorithms. It is to be noticed that in Cora, both AP1 and AP2 have low precision and high NDCG because they return node pairs with high (not highest) SimRank. In the meanwhile, KSimJoin is achieves highest precision and NDCG of the three algorithms.

Table 9 shows MAE and RMSE of three algorithms when $k = 2000$. The MAE and RMSE of AP1 is the largest, which means the estimation of AP1 is of the greatest error. The RMSE of AP2 is 10 times larger than the MAE of it, which means the individual estimation error of AP2 is unstable. Among all three algorithms, KSimJoin is the most effective one with both the MAE and RMSE equal 0.

Table 10 presents the top-5 returned pairs of three algorithms in wordnet3. We observed that the word pairs given by KSimJoin are more similar. For instance, “Easter” and “Passover” are both festivals, “epicondylitis” and “tendinitis” are both inflammation. On the contrary, the words pairs returned by AP1 and AP2 are less similar, for example, as to the 5th result, while approximation methods return pairs like (“antispasmodic”,“mydriatic”) and (“adopt”,“follow_through”), which

Table 9
MAE and RMSE, $k = 2000$.

Percentage	Adolhealth		Wordnet3		Cora	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
AP1	0.014	0.021	0.006	0.015	0.199	0.199
AP2	6.817×10^{-6}	1.467×10^{-5}	1.151×10^{-8}	1.073×10^{-7}	0	0
KSimJoin	0	0	0	0	0	0

Table 10
Top-5 answer pairs, case study of Wordnet3.

AP1	AP2	KSimJoin
<adopt, follow_through>	<unyoke, unharness>	<unyoke, unharness>
<draw_in, sheathe>	<ferric_oxide, oxidize>	<disavow, contract>
<hearer, snoop>	<concur, coincide>	<ferric_oxide, oxidize>
<overprint, overcrowd>	<draw_in, sheathe>	<Easter, Passover>
<antispasmodic, mydriatic>	<adopt, follow_through>	<epicondylitis, tendinitis>

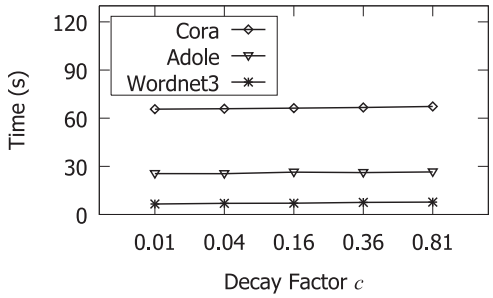


Fig. 7. Running time vs. c, $k = 2000$.

Table 11
Running time of KSimJoin and KSimJoin-imp, $k = 2000$.

Running time (s)	Adolhealth	Wordnet3	Cora
KSimJoin	21.023	4.575	66.074
KSimJoin-imp	20.282	1.214	35.464

are semantically less similar, in comparison with ("epicondylitis","tendinitis") by KSimJoin. The case study also indicates that our method is more effective than others.

6.4. Evaluating sensitivity

As discussed in Section 6.2 that KSimJoin is less sensitive to k compare to SRK-Join and H-go Join, we will evaluate the effect of decay factor c in this section. Fig. 7 shows the running time of SRK-Join on three networks with $k = 2000$ and c changing from 0.01 to 0.81. It is evident that the running time keeps almost the same when c changes, which means that KSimJoin is hardly affected by decay factor c. This is because snbUB is designed based on the SimRank between the object node and its super node. No matter how decay factor c changes, the relative numerical relationship between the threshold and the upper-bound will change little.

6.5. Evaluating improvement for large networks

In this set of experiments, we evaluate the proposed improvement for large networks, i.e., KSimJoin versus KSimJoin-imp. Table 11 lists the running time cost on three networks when $k = 2000$. Recall that main difference between KSimJoin and KSimJoin-imp is that KSimJoin-imp avoid building *NP* and *RNP* in advance. We see that on Wordnet3 and Cora, KSimJoin-imp costs about 30% less time than KSimJoin. This evidences the effectiveness of the improvement.

On Adolhealth, however, the running time of both algorithms are roughly the same. This is because Adolhealth is a network with only 2000 nodes, and hence, the cost of building *NP* and *RNP* is not remarkable. This also implies that the improvement is tailored for large networks.

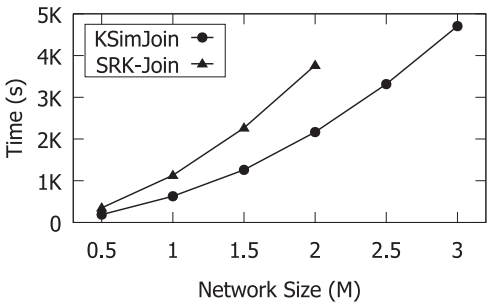


Fig. 8. Running time, $k = 2000$.

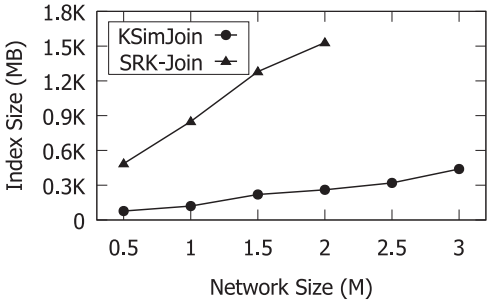


Fig. 9. Memory cost, $k = 2000$.

6.6. Evaluating scalability

In this set of experiments, we used synthetic datasets with 0.5M–3M nodes to test the scalability of the methods – KSimJoin and SRK-Join. We set $k = 2000$ and report running time and the memory usage of the *NP* and *RNP* in Figs. 8 and 9.

We can see from Fig. 8 that the running time of KSimJoin is constantly shorter than that of SRK-Join, which is in accordance with the results in Section 6.2. In particular, for SRK-Join, the overall running time on networks with 1 M, 1.5 M, and 2 M are, respectively, 1129 s, 2266 s, and 3766 s; while for KSimJoin, they are 627 s, 1260 s and 2165 s, respectively. Fig. 9 witness a similar trend for memory consumption, the memory usage of KSimJoin is much smaller than SRK-Join. Note that for networks with more than 2M nodes, SRK-Join did not finish after 5000 s, and the results are not reported in the figures.

7. Related work

Ever since SimRank was proposed [4] as a structural-based similarity measure towards nodes on networks, the problem of efficient SimRank computation has been extensively studied in a long line of research. The proposed solutions can be classified into three categories, i.e., fix-point iteration based solutions, low-rank based solutions and random-walk based solutions.

The first fix-point iteration based solution was proposed in [4], which computes all-pairs SimRank in $O(kd^2n^2)$ time and $O(n^2)$ space in k iterations. Lizorkin et al.improved it via partial sum memorization to $O(kdn^2)$ time [9]. Later, Yu et al. developed optimization techniques for minimizing the matrix bandwidth, and improved the I/O efficiency of SimRank iteration [15]. Recently, Yu et al. further devised an algorithm based on a minimum spanning tree to eliminate the partial sums redundancy [17]. Li et al. developed a single-pair approach by computing the position matrix in $O(kd \cdot \min\{d^k, n^2\})$ time and $O(n^2)$ space [8]. He et al. proposed an iterative aggregation technique to compute SimRank in parallel for large networks [3]. Due to the high time and space complexity, these methods are not considered to be efficient approaches to SimRank based similarity queries on large networks.

Low-rank based techniques are able to compute SimRank approximately. Yu et al. developed an efficient SimRank algorithm for undirected networks via eigenvalue decomposition technique [16]. In the meanwhile, Li et al. re-wrote the SimRank equation into a non-iterative form through Kronecker product and vectorization operators [7]. By pre-computing four auxiliary matrices, the similarity with respect to a given node can be computed in $O(r^4n)$ time, where r is the target rank of transpose matrix of the original network. Onizuka et al. used the matrix representation based on Sylvester equation, and improved the performance of similarity search to $O(rn)$ [2]. All these techniques are used for approximating SimRank and hence, are not feasible to *exact* similarity queries.

SimRank is also proved equivalent to the expected probability of two random surfers started at two queried nodes and first meet within ξ steps. Here ξ is equal to the time of iterations k in the iteration-based model. Under the random surfer model, there are two streams of research efforts. One is based on Monte-Carlo sampling, where SimRank is estimated by random sampling for enough times as per a user-defined error bound. Fogaras et al. proposed the first random-walk based algorithm for SimRank search based on Monte-Carlo sampling in $O(nr)$ time and space [1]. Recently, Kusumoto et al. formalized SimRank as linear recursive formula, and designed another Monte-Carlo based algorithm via Cauchy–Schwartz inequality [5], of which the preprocessing cost is $O(\xi nr)$, where r is the number of samples. The other is to enumerate possible random walks. Lee et al. proposed TopSim algorithm [6], which leverages heuristic rules to prune random walks during computation. The latest contribution of this line is from Tao et al. [14]. Utilizing the complementary relationship between first-meeting and multi-meeting probabilities, they encoded each node as a vector and transformed the calculation of SimRank between two nodes to dot product between two corresponding two vectors. The transformation from SimRank to dot product is $O(d^{\xi}n)$. In addition, early-termination rules were devised to prune unpromising candidate node pairs. Our work is inspired by the approach; however, instead of running a two-phase framework, we present a holistic framework with iterative batch pruning leveraging a novel upper-bounding technique.

Another related work to our paper is graph-based random walk querying technique. Nie et al. introduced a schema to enrich textual answers with image and video data for better QA experience [11]. He further studied the problem of automatically predicting searching performance [10] and querying with complex queries [12]. Zhang et al. studied the inbound top-k query based on random walk with restart.

8. Conclusion

In this paper, we have investigated the problem of top-k similarity join based on SimRank. Based on the method of two-way paths for SimRank computation, we first developed an incremental top-k algorithm for partial SimRank. On top of that, we designed an iterative batch pruning framework to speed up the overall join process, which is able to prune all the node pairs involving a node if the node is determined to be disqualified. The pruning strategy utilizes on a novel estimation based on the concept of super node, which is not only tight but also easy to obtain. Extensive experiments on real-life and synthetic networks confirm the effectiveness and efficiency of the proposed techniques.

Seeing the intrinsic uncertainty in various applications, e.g., protein–protein interactions and machine-augmented knowledge graphs, in the near future, we plan to look into SimRank based similarity joins over probabilistic networks. In particular, we will investigated how to map the random surfer model to probabilistic world, and how to carry out efficient path enumeration therein.

Acknowledgments

This work was partially supported by NSFC under Grant nos. 61402494 and 61402498, and NSF of Hunan under Grant no. 2015JJ4009.

References

[1] D. Fogaras, B. Rácz, Scaling link-based similarity search, in: Proceedings of the Fourteenth International Conference on World Wide Web, Chiba, Japan, 2005, pp. 641–650. May 10-14, 2005.
[2] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, M. Onizuka, Efficient search algorithm for SimRank, in: Proceedings of the Twenty Ninth IEEE International Conference on Data Engineering, Brisbane, Australia, 2013, pp. 589–600. April 8-12, 2013.
[3] G. He, H. Feng, C. Li, H. Chen, Parallel SimRank computation on large graphs with iterative aggregation, in: Proceedings of the Sixteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, 2010, pp. 543–552. July 25-28, 2010.
[4] G. Jeh, J. Widom, SimRank: a measure of structural-context similarity, in: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada, 2002, pp. 538–543. July 23–26, 2002.
[5] M. Kusumoto, T. Maehara, K. Kawarabayashi, Scalable similarity search for SimRank, in: Proceedings of the Thirty Third ACM SIGMOD International Conference on Management of Data, Snowbird, UT, USA, 2014, pp. 325–336. June 22–27, 2014.
[6] P. Lee, L.V.S. Lakshmanan, J.X. Yu, On top-k structural similarity search, in: Proceedings of the Twenty Eighth IEEE International Conference on Data Engineering, Washington, DC, USA (Arlington, Virginia), 2012, pp. 774–785. 1–5 April, 2012.
[7] C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, T. Wu, Fast computation of SimRank for static and dynamic information networks, in: Proceedings of the Thirteenth EDBT International Conference on Extending Database Technology, Lausanne, Switzerland, 2010, pp. 465–476. March 22–26, 2010..
[8] P. Li, H. Liu, J.X. Yu, J. He, X. Du, Fast single-pair SimRank computation, in: Proceedings of the Tenth SIAM International Conference on Data Mining, Columbus, Ohio, USA, 2010, pp. 571–582. April 29 - May 1, 2010.
[9] D. Lizorkin, P. Velikhov, M.N. Grinev, D. Turdakov, Accuracy estimate and optimization techniques for SimRank computation, VLDB J. 19 (1) (2010) 45–66.
[10] L. Nie, M. Wang, Z. Zha, T. Chua, Oracle in image search: a content-based approach to performance prediction, ACM Trans. Inf. Syst. 30 (2) (2012) 13.
[11] L. Nie, M. Wang, Z. Zha, G. Li, T. Chua, Multimedia answering: enriching text QA with media information, in: Proceeding of the Thirty Fourth International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, 2011, pp. 695–704. July 25–29, 2011.
[12] L. Nie, S. Yan, M. Wang, R. Hong, T. Chua, Harvesting visual concepts for image search with complex queries, in: Proceedings of the Twentieth ACM Multimedia Conference, MM '12, Nara, Japan, 2012, pp. 59–68. October 29 - November 02, 2012.
[13] Y. Shao, B. Cui, L. Chen, M. Liu, X. Xie, An efficient similarity search framework for SimRank over large dynamic graphs, in: Proceedings of the Forty First International Conference on Very Large Data Bases Endowment, 8, 2015, pp. 838–849.
[14] W. Tao, M. Yu, G. Li, Efficient top-k SimRank-based similarity join, in: Proceedings of the Fortieth International Conference on Very Large Data Bases Endowment, 8, 2014, pp. 317–328.

[15] W. Yu, X. Lin, J. Le, A space and time efficient algorithm for SimRank computation, in: Proceedings of the Twelfth Asia-Pacific Web Conference, Busan, Korea, 2010, pp. 164–170. 6–8 April 2010.
[16] W. Yu, X. Lin, J. Le, Taming computational complexity: efficient and parallel SimRank optimizations on undirected graphs, in: Proceedings of the Eleventh International Conference on Web-Age Information Management, Jiuzhaigou, China, 2010, pp. 280–296. July 15–17, 2010.
[17] W. Yu, X. Lin, W. Zhang, Towards efficient SimRank computation on large networks, in: Proceedings of Twenty Ninth IEEE International Conference on Data Engineering, Brisbane, Australia, 2013, pp. 601–612. April 8–12, 2013.
[18] W. Zheng, L. Zou, Y. Feng, L. Chen, D. Zhao, Efficient SimRank-based similarity join over large graphs, in: Proceedings of the Forty First International Conference on Very Large Data Bases Endowment, 6, 2013, pp. 493–504.