Scalable Similarity Search for SimRank

Mitsuru Kusumoto*
Preferred Infrastructure, Inc
mkusumoto@preferred.jp

Takanori Maehara
National Institute of Informatics
JST, ERATO, Kawarabayashi Project
maehara@nii.ac.jp

Ken-ichi Kawarabayashi
National Institute of Informatics
JST, ERATO, Kawarabayashi Project
k_keniti@nii.ac.jp

ABSTRACT

SimRank, proposed by Jeh and Widom, provides a good similarity score and has been successfully used in many of the above mentioned applications. While there are many algorithms proposed so far to compute SimRank, but unfortunately, none of them are scalable up to graphs of billions size. Motivated by this fact, we consider the following SimRank-based similarity search problem: given a query vertex u, find top-k vertices v with the k highest SimRank scores $s(u, v)$ with respect to u.

We propose a very fast and scalable algorithm for this similarity search problem. Our method consists of the following ingredients:

1. We first introduce a "linear" recursive formula for SimRank. This allows us to formulate a problem that we can propose a very fast algorithm.
2. We establish a Monte-Carlo based algorithm to compute a single pair SimRank score $s(u, v)$, which is based on the random-walk interpretation of our linear recursive formula.
3. We empirically show that SimRank score $s(u, v)$ decreases rapidly as distance $d(u, v)$ increases. Therefore, in order to compute SimRank scores for a query vertex u for our similarity search problem, we only need to look at very "local" area.
4. We can combine two upper bounds for SimRank score $s(u, v)$ (which can be obtained by Monte-Carlo simulation in our preprocess), together with some adaptive sample technique, to prune the similarity search procedure. This results in a much faster algorithm.

Once our preprocess is done (which only takes $O(n)$ time), our algorithm finds, given a query vertex u, top-20 similar vertices v with the 20 highest SimRank scores $s(u, v)$ in less than a few seconds even for graphs with billions edges.

*supported by JST, ERATO, Kawarabayashi Project

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.

http://dx.doi.org/10.1145/2588555.2610526.

To the best of our knowledge, this is the first time to scale for graphs with at least billions edges(for the single source case).

1. INTRODUCTION

1.1 SimRank

Recent advances in social and information science have shown that linked data pervades our society. Since graphs are an intuitive abstraction that can naturally capture objects as well as their relationships as links, they are becoming increasingly important to represent complicated structures and schema-less data. Much effort has been devoted to extracting useful information from large graphs, and many applications (e.g., link prediction [22], graph clustering [33,40], spam detection [4, 11], collaborative tagging analysis [14], and recommender systems [1]) show some success of vertex similarities and similarity search methods.

Several similarity measures have been proposed. For example, bibliographic coupling [16], co-citation [30], P-Rank [38], PageSim [23], Extended Nearest Neighborhood Structure [24], MatchSim [25], and so on. In this paper, we consider *SimRank* that was introduced by Jeh and Widom [13]. They observed that "two similar pages are linked from many similar pages." Due to this observation, SimRank is defined as the following, recursively.

$$s(u,v) = \begin{cases} 1 & (u = v), \\ \dfrac{c}{|\delta(u)||\delta(v)|} \displaystyle\sum_{u' \in \delta(u), v' \in \delta(v)} s(u', v') & (u \neq v), \end{cases} \tag{1}$$

where $\delta(u)$ denotes in-links of a vertex u and $c \in (0, 1)$ denotes a locality parameter, called a *decay factor*, which is usually set to $c = 0.8$ [13] or $c = 0.6$ [26]. SimRank and related similarity measures give high-quality results than other similarity measures, such as bibliographic coupling or co-citation. The reason is that SimRank exploits information on "multi-step neighborhoods" while other similarity measures, such as bibliographic coupling or co-citation, utilize only the one-step neighborhoods of pages.

1.2 SimRank computation

Although SimRank gives high-quality similarity measure, it is not so widely used in practice, due to high computational cost. While there are several algorithms proposed so far to compute SimRank scores, unfortunately, their computation complexities (in both time and space) are very expensive. The difficulty of computing SimRank may be viewed

as follows: to compute a SimRank score $s(u, v)$ for two vertices u, v, since (1) is defined recursively, we have to compute SimRank scores for all pairs of vertices. Therefore, since the number of pairs is $O(n^2)$, it requires $O(n^2)$ space and $O(n^2)$ time, where n is always the number of vertices in G[1].

In order to reduce this computation cost, several approaches have been proposed [9, 12, 19, 21, 26, 35, 36, 37]. Here, we briefly survey some existing computational techniques for SimRank. We summarize the existing results in Table 1. Let us point out that there are three fundamental problems for SimRank: (1) single-pair SimRank to compute $s(u, v)$ for given two vertices u and v, (2) single-source SimRank to compute $s(u, v)$ for a given vertex u and all other vertices v, and (3) all-pairs SimRank to compute $s(u, v)$ for all pair of vertices u and v.

In the original paper by Jeh and Widom [13], all-pairs SimRank scores are computed by recursively evaluating the equation (1) for all $u, v \in V$. This "naive" computation yields an $O(Td^2n^2)$ time algorithm, where T denotes the number of iterations and d denotes the average degree of a given network. Lizorkin et al. [26] proposed a "partial sum" technique, which memorizes partial calculations of Jeh and Widom's algorithm to reduce the time complexity of their algorithm. This leads to an $O(T \min\{nm, n^3/\log n\})$ algorithm. Yu et al. [37] applied the fast matrix multiplication [31, 32] and then obtained an $O(T \min\{nm, n^\omega\})$ algorithm to compute all pairs SimRank scores, where $\omega < 2.373$ is the exponent of matrix multiplication. Note that the space complexity of these algorithms is $O(n^2)$, since they have to maintain all SimRank scores for each pair of vertices to evaluate the equation (1). This results is, so far, the state-of-the-art algorithm to compute SimRank scores for all pairs of vertices.

There are some random-walk based algorithms. Jeh and Widom [13] proposed a random-walk interpretation of SimRank, which is called a "random surfer-pair model". Let us consider two random walks that start from vertices u and v, respectively, and follow the in-links. Let $u^{(t)}$ and $v^{(t)}$ be the t-th position of each random walk, respectively. The first meeting time $\tau_{u,v}$ is defined by

$$\tau_{uv} = \min_t \{u^{(t)} = v^{(t)}\}. \quad (2)$$

Then SimRank score is obtained by

$$s(u, v) = \mathbf{E}[c^{\tau_{u,v}}]. \quad (3)$$

Fogaras and Rácz [9] evaluate the right-hand side by Monte-Carlo simulation with a fingerprint tree data structure, and they obtained a faster algorithm to compute single pair SimRank score for given two vertices u, v. This work is relevant to our proposed algorithm because our algorithm is also based on Monte-Carlo simulation. Moreover, this is the state-of-the-art Monte-Carlo based algorithm for the single-pair and the single-source SimRank problems.

Some papers proposed spectral decomposition based algorithms (e.g., [10, 12, 19, 35, 36]), but there is a mistake in the formulation of SimRank. On the other hand, their algorithms may output reasonable results. We shall mention more details about these algorithms in Subsections 3.3.

To the best of our knowledge, there is no existing algorithm so far that can compute (single pair) SimRank for $m \geq 20,000,000$ size graphs. Indeed in Section 8, we shall clarify this fact, due to the space constraint. issues.

[1] m is always the number of edges in G

2. CONTRIBUTIONS

2.1 Problem setting

In this paper, we consider a somewhat different problem, called "The top-k similarity search problem", as follows:

PROBLEM 1 (TOP-k SIMILARITY SEARCH).

- *Given: A graph $G = (V, E)$.*
- *Query: A query vertex u and a positive integer k.*
- *Output: Top-k vertices with the k highest SimRank similarities with respect to a query vertex u.*

This problem is a special case of the single-source SimRank problem, since our problem only needs k highly-similar vertices. There are two advantages and reasons for this.

Firstly, this problem could be possibly solved more efficiently. More precisely, as pointed above, the difficulty of computing SimRank score $s(u, v)$ for two vertices u, v comes from (1) which is defined recursively. Thus we have to compute SimRank scores for all pairs of vertices, and hence it requires $O(n^2)$ space. On the other hand, the top-k similarity search problem has only k outputs for each vertex. Furthermore, since we do not have to compute the exact SimRank scores, we are allowed to use some "approximation" techniques. This could also help accelerating algorithms.

Secondly, in practice, there are usually only small number of vertices (e.g., 10 to 20 vertices) that have high SimRank scores for a query vertex u (e.g., greater than 10^{-2}). Since in many applications we are only interested in "very" similar vertices, it makes sense to consider only "top-k" vertices for a query vertex u, rather than to compute SimRank scores $s(u, v)$ for all vertices $v \in V$.

2.2 Our main contributions and overview

We propose a novel method that can efficiently find top-k similar vertices, based on SimRank. Our algorithm is scalable to graphs of billions edges.

Technically, our algorithm is based on the following four key ingredients:

1. We first establish a new framework of computing SimRank, which introduces a "linear" recursive formula for SimRank.
2. Based on this linear recursive formula, single-pair SimRank score $s(u, v)$ can be computed very efficiently by Monte-Carlo simulation. Indeed, the time complexity is independent of the size of networks (e.g., n, m).
3. We observe that SimRank score $s(u, v)$ decays very rapidly as distance of the pair u, v increases.
4. By the above observation, we establish upper bounds of SimRank score $s(u, v)$ that only depend on distance $d(u, v)$. The upper bounds can be efficiently computed by Monte-Carlo simulation (in our preprocess).

   These upper bounds, together with some adaptive sample technique, allow us to effectively prune the similarity search procedure.

Combining these ingredients, we can obtain the following algorithm for top-k similarity search problem (Problem 1).

Table 1: Complexity of SimRank algorithms. n denotes the number of vertices, m denotes the number of edges, d denotes the average degree, T denotes the number of iterations, R is the parameter for the algorithm [9], and r denotes the rank for low-rank approximation.

Algorithm	Type	Time	Space	Technique
Proposed (Section 7)	Top-k search	$<< O(n)$	$O(m)$	Linear recursive formulation & Monte Carlo
Proposed (Section 7)	Top-k for all	$<< O(n^2)$	$O(m)$	Linear recursive formulation & Monte Carlo
Li et al. [21]	Single-pair	$O(Td^2n^2)$	$O(n^2)$	Random surfer pair (Iterative)
Fogaras and Rácz [9]	Single-pair	$O(TR)$	$O(m + nR)$	Random surfer pair (Monte Carlo)
Jeh and Widom [13]	All-pairs	$O(Tn^2d^2)$	$O(n^2)$	Naive
Lizorkin et al. [26]	All-pairs	$O(T \min\{nm, n^3/\log n\})$	$O(n^2)$	Partial sum
Yu et al. [37]	All-pairs	$O(T \min\{nm, n^\omega\})$	$O(n^2)$	Fast matrix multiplication
Li et al. [20]	All-pairs	$O(Tn^{4/3})$	$O(n^{4/3})$	Block partition
Li et al. [19]	All-pairs	$O(r^4n^2)$	$O(n^2)$	Singular value decomposition
Fujiwara et al. [10]	All-pairs	$O(r^4n)$	$O(r^2n^2)$	Singular value decomposition
Yu et al. [35]	All-pairs	$O(n^3 + Tn^2)$	$O(n^2)$	Eigenvalue decomposition

There are two phases: (1) the preprocess phase (for the fourth ingredients) and (2) the query phase with respect to a query vertex u (to obtain top-k vertices for u).

We first perform preprocess to compute the auxiliary values for upper bounds of SimRank $s(u, v)$ for all $v \in V$ (see Section 6). In addition, we construct an auxiliary bipartite graph H as in Section 7.1. This graph H allows us to enumerate "candidates" of highly similar vertices v more accurate. This is our preprocess phase. The time complexity is $O(n)$.

We now perform our query phase. We compute SimRank scores $s(u, v)$ for each vertex u in the ascending order of distance from a given vertex u, and at the same time, we perform "pruning" by the upper bounds obtained in the fourth ingredient. Each $s(u, v)$ computation is very efficient by the second ingredient. By the third ingredient, all top-k vertices (with respect to u) are close to u (in the sense of graph distance). In addition, we use the adaptive sample technique. Specifically, for a query vertex u, we first estimate SimRank scores roughly for each candidate v (i.e, we only perform small number of random walks for v). Then, we re-compute more accurate SimRank scores for each candidate v that has high estimated SimRank scores (i.e., we perform more random walks for v. For more details, please see Section 7.2).

In summary, our search procedure finds all top-k vertices efficiently by checking the upper bounds and performing the adaptive sampling, and moreover it terminates very quickly (since it only looks at local area, i.e, vertices that are close to u). The time complexity is $O(n)$ in the worst case, but our experiments in Section 8 shows that it is indeed much faster, see below.

We can actually perform the similarity search for all vertices. We simply apply the above similarity search procedure for each vertex u. The time complexity is $O(n^2)$ in the worst case but it is much faster in practice. The space complexity is $O(m + kn)$, where kn comes from the number of outputs. It is also worth noting that our algorithm is distributed computing friendly. The bottleneck of the algorithm is computing single-source SimRank scores locally, as described above, but this part can be performed independently. Therefore if there are M machines, the running time of our algorithm is $O(n^2/M)$, i.e., reduced by a factor M^2.

In summary, our algorithm has the following attractive characteristics:

1. **Small space:** It requires $O(n)$ space for storing the preprocess results. In addition, for both the single-source case and all vertices case, the algorithm uses only $O(m)$ space to store a network. By comparison, almost all previously known algorithms need $O(n^2)$ space.

2. **Very fast:** For the preprocess phase, the time complexity is $O(n)$ (see Section 7).

   Concerning the query phase: for the single-source case (Problem 1), the time complexity is $O(n)$ in the worst case but much faster in practice. For all vertices case, the time complexity is $O(n^2)$ in the worst case but much faster in practice.

3. **Accurate:** Our algorithm computes only "approximate SimRank". However, we empirically show that our algorithm gives almost exact top-k vertices. See Section 8.2 (and Figure 1 in Section 3.3).

Let us observe that for the space complexity, $O(m)$ is optimal, because we have to read all edges of an input graph.

We conducted our experiments to implement our proposed algorithm. Our experiments show that the proposed algorithm can scale up to billion-size real networks for the single-source case. Furthermore, since our all-pairs algorithm can easily be parallelized to multiple machines, if there are 100 machines, even for graphs of billions size, our all-pairs algorithm can output all top-20 vertices in less than 5 days. To the best of our knowledge, this is the first time to scale up to such large networks.

Comparison with other existing algorithms. We compare our algorithm with the existing ones in Subsection 8.3. Our algorithms have the following advantages:

1. The state-of-the-art all-pairs algorithm, proposed by Yu et al. [37], requires much more memory, and hence works only for graphs with at most a million edges. On the other hand, our all-pairs algorithm requires much less memory, therefore it works for a very large networks.

2. The state-of-the-art Monte Carlo based single-pair and single-source algorithm by Fogaras and Racz [9] requires much more space than our algorithm(even for the case $R' = 100$ samples for their algorithms). Even though their algorithm can be faster both in preprocess and in query time, their algorithm scales up to graphs with at most $70,000,000$ edges. In addition, the accuracy of their algorithm is less than that of our algorithm, see Section 8.2.

The above two comparisons explain why our algorithm is the first to scale up to graphs of billions edges. Namely,

[2]indeed, it would be much faster, since in practice our proposed algorithm for Problem 1 would be much faster

some state-of-the-art algorithm (i.e., Fogaras and Racz [9] Monte-Carlo type algorithm) may be faster than ours in query time, but it does not scale up to large graphs, due to the space constraint. By comparison, our algorithm requires much less memory space than any other existing algorithms, which allows our algorithm to scale for much larger graphs.

Organization of the paper

This paper is organized as follows. In Section 3, we first introduce a "linear" recursive formula for SimRank. Based on this linear recursive formula, single-pair SimRank score $s(u, v)$ can be computed very efficiently by Monte-Carlo simulation. This will be presented in Section 4. SimRank score $s(u, v)$ decays very rapidly as distance of the pair u, v increases. This is shown in Section 5. In Section 6, the upper bound of SimRank score $s(u, v)$ will be given via our theoretical analysis. Indeed, we give two such upper bounds, one is more effective for a low degree query vertex, and the other is more effective for a high degree vertex. We also present efficiently Monte-Carlo based algorithms to obtain such two upper bounds in Section 6. Note that this is done in our preprocess phase. In Section 7 we propose an algorithm for the similarity search problem by combining the above ingredients. Finally, in Section 8, we present many experiments to justify our proposed algorithm, and compare with other state-of-the-arts algorithms. We conclude our paper in Section 9.

3. NEW COMPUTATIONAL FRAMEWORK OF SIMRANK

In this section, we introduce a new computational framework of SimRank. Our framework introduces the *linear recursive formulation* of SimRank (5) (see below) to "decompose" the computational difficulty of SimRank. The framework immediately leads to an efficient deterministic algorithm for SimRank (see Subsection 3.2) and a very efficient randomized algorithm for SimRank (see Section 4). Furthermore, since the framework is easy to analyze, we can establish some upper bounds for SimRank (see Section 6), which are very useful to prune the similarity search. This pruning results in a much faster algorithm.

3.1 Linear recursive formulation

We first introduce a matrix notation of SimRank. Let P be a transition matrix of the transposed graph $G^\top$, i.e.,

$$P_{ij} = \begin{cases} 1/|\delta_-(j)| & (i,j) \in E, \\ 0 & (i,j) \notin E. \end{cases}$$

Let $S := (s(i,j))_{ij}$ be a matrix whose (i,j) entry is the SimRank score of i and j (such a matrix S is called a *SimRank matrix*). Then the SimRank equation (1) is simply represented by

$$S = (cP^\top SP) \vee I, \tag{4}$$

where $\vee$ denotes the entry-wise maximum, i.e., $(A \vee B)_{ij} := \max\{A_{ij}, B_{ij}\}$.

The difficulty of computing SimRank stems from the recursion (4). This is hard to compute because we have to figure out the entry-wise maximum each time (i.e., non-linear recursion)[3].

In order to overcome this obstacle, the first main idea of this paper is to establish the *linear recursive formulation* of SimRank, which consists of only linear operations as follows. Let us introduce the diagonal matrix D such that

$$S = cP^\top SP + D. \tag{5}$$

Such a matrix always exists because we can take $D := S - cP^\top SP$ for the SimRank matrix defined by (4). We call this matrix D the *diagonal correction matrix*. Using the notion of the diagonal correction matrix, we can define SimRank as follows:

> SimRank matrix is a matrix that satisfies (5) with some diagonal matrix D and $S_{ii} = 1$ for all $i \in V$.

This important point of this formulation is the following proposition, whose proof will be given in Appendix.

PROPOSITION 1. *If a matrix S' satisfies (5) for some diagonal matrix D, and $S'_{ii} = 1$ for $i \in V$ then S' is equal to the SimRank matrix S. The converse is also true.*

We can also bound the range of D as follows.

PROPOSITION 2.

$$(1 - c) \leq D_{ii} \leq 1. \tag{6}$$

For the proof of Propositions 2, see Appendix.

We here show an example to illustrate the linear formulation of SimRank (5).

EXAMPLE 1. *Consider the star graph of order 4 (i.e., a claw). The transition matrix (of the transposed graph) is*

$$P = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 \end{bmatrix},$$

and SimRank for $c = 0.8$ is

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 4/5 & 4/5 \\ 0 & 4/5 & 1 & 4/5 \\ 0 & 4/5 & 4/5 & 1 \end{bmatrix}.$$

Since

$$S - cP^\top SP = \mathrm{diag}(23/75, 1/5, 1/5, 1/5),$$

we can take $D = \mathrm{diag}(23/75, 1/5, 1/5, 1/5)$ for (5). Let us emphasis that $D \neq (1 - c)I$.

3.2 Computing SimRank

Having defined our linear recursive formulation (5), we are ready to mention our way to compute SimRank.

For the equation (5), by substituting the left-hand side to the right-hand side recursively, we obtain the (converging) series:

$$S = D + cP^\top DP + c^2 P^{\top 2} DP^2 + \cdots. \tag{7}$$

[3]This is the exactly the place where the papers (e.g., [10,12, 19,35,36]) made mistakes. See Subsection 3.3

Therefore the SimRank score $s(u,v)$ is obtained from (u,v) component of (7) by

$$\begin{aligned} s(u,v) &= e_u^\top S e_v \\ &= e_u^\top D e_v + c(Pe_u)^\top DPe_v + c^2(P^2e_u)^\top DP^2e_v + \cdots . \end{aligned} \tag{8}$$

Let $s^{(T)}(u,v)$ be the sum of T-th terms of the above equation:

$$\begin{aligned} s^{(T)}(u,v) :=& e_u^\top D e_v + c(Pe_u)^\top DPe_v \\ &+ \cdots + c^{T-1}(P^{T-1}e_u)^\top DP^{T-1}e_v. \end{aligned} \tag{9}$$

Then $s^{(T)}(u,v)$ converges to $s(u,v)$ with rate c^T. More precisely, it is straightforward to see that

$$0 \leq s(u,v) - s^{(T)}(u,v) \leq \frac{c^T}{1-c}. \tag{10}$$

This implies that if we want to compute $s(u,v)$ with accuracy less than $\epsilon > 0$, we just need to compute $s^{(T)}(u,v)$ by taking $T = \lceil \log(\epsilon(1-c))/\log c \rceil$. This yields an algorithm whose time and space complexity is $O(Tm)$ and $O(n)$ space, respectively.

As claimed in Section 1, we would like to make this time complexity faster. For this purpose, in the rest of the paper we focus on computing $s^{(T)}(u,v)$, instead of $s(u,v)$.

3.3 Estimating diagonal correction matrix

Estimating the diagonal correction matrix D is a hard task. Indeed it is as difficult as computing the SimRank matrix S itself. Hence we need an easily computable approximation of D.

Recall that some papers use the following SimRank "definition", which is indeed incorrect:

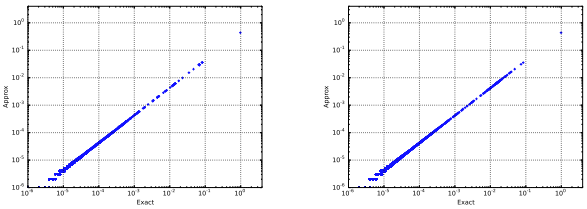
$$S' = cP^\top S' P + (1-c)I. \tag{11}$$

(e.g., equation (2) in [19], equation (2) in [12], equation (3) in [36], and equation (2) in [10]). This recursion does not provide the original SimRank because S' does not necessary satisfy $S'_{ii} = 1$ $(i = 1, \ldots, n)$; see Example 1 for a counterexample.

However, if we regard (11) as an "approximation" of D, i.e., $D \simeq (1-c)I$, the situation is much better for the similarity search problem. Because, in practice, this approximation does not much affect top-k similarity rankings ("approximate" SimRank scores could be different from the "exact" SimRank scores, though), in this paper, we simply approximate $D \simeq (1-c)I$, as other papers did (e.g., [10, 12, 19, 35, 36]). We now give practical justifications for approximating $D \simeq (1-c)I$. Figure 1 shows the scatter plot of exact SimRank scores and approximated SimRank scores (based on (5)) for highly similar vertices. We can see that the points are in a straight line of slope one (in log-log plot). This means that the approximation $D \simeq (1-c)I$ only changes the scale of each SimRank score and hence it does not affect the similarity ranking.

Note that our proposed method does not depend on the approximation $D \simeq (1-c)I$. Hence if we estimate D more accurately, the result also becomes more precisely. Therefore, in the rest of the paper we keep using the symbol "D" to denote $(1-c)I$, to emphasize this point.

REMARK 1. *Since our definition* (5) *is linear in D, the top-k similarity ranking does not change under the scaling of D. Therefore if the "exact" diagonal correction matrix D is proportional to the identity matrix I, we can exactly finds the top-k similarity ranking by approximating $D \simeq (1-c)I$.*



(a) ca-GrQc (b) cit-HepTh

Figure 1: Correlation of exact SimRank scores and approximated SimRank scores

4. MONTE CARLO ALGORITHM FOR SINGLE-PAIR SIMRANK

The computational method based on (9) (described in Subsection 3.2) is more efficient than any previously proposed algorithm (indeed this is the first linear time algorithm $(O(Tm))$, and the first linear space algorithm to compute SimRank score for a single pair of vertices).

However, since we are interested in much faster computation for the top-k similarity search problem for a single vertex, this is not enough, because we consider that the computation time $O(Tm)$ is still expensive. To be more precise, we would like to solve the top-k similarity search problem for all vertices. In this case, the above algorithm would only yield an $O(Tmn)$ algorithm, which is definitely too expensive.

Here we propose a Monte-Carlo algorithm for the Single source SimRank, which is based on the random-walk interpretation of formula (9). The algorithm requires only a small number of samples. This algorithm yields a much faster algorithm for the top-k similarity search problem for a single vertex.

Let us consider a random walk that starts from $u \in V$ and that follows its in-links, and let $u^{(t)}$ be a random variable for the t-th position of this random walk. Then we observe that

$$P^t e_u = \mathbf{E}[e_{u^{(t)}}]. \tag{12}$$

Therefore, by plugging (12) to (9), we obtain

$$\begin{aligned} s^{(T)}(u,v) =& D_{uv} + c\mathbf{E}[e_{u^{(1)}}]^\top D\mathbf{E}[e_{v^{(1)}}] \\ &+ \cdots + c^{T-1}\mathbf{E}[e_{u^{(T-1)}}]^\top D\mathbf{E}[e_{v^{(T-1)}}]. \end{aligned} \tag{13}$$

Our algorithm computes the expectations in the right hand side of (13) by Monte-Carlo simulation as follows: Consider R independent random walks $u_1^{(t)}, \ldots, u_R^{(t)}$ that start from $u \in V$, and R independent random walks $v_1^{(t)}, \ldots, v_R^{(t)}$ that start from $v \in V$ with $u \neq v$. Then each t-th term of (13) can be estimated as

$$c^t \mathbf{E}[e_{u^{(t)}}]^\top D\mathbf{E}[e_{v^{(t)}}] \simeq \frac{c^t}{R^2} \sum_{r=1}^{R} \sum_{r'=1}^{R} D_{u_r^{(t)} v_{r'}^{(t)}}. \tag{14}$$

We compute the right hand side of (14). Specifically by maintaining the positions of $u_1^{(t)}, \ldots, u_R^{(t)}$ and $v_1^{(t)}, \ldots, v_R^{(t)}$ by hash tables, it can be evaluated in $O(R)$ time. Therefore

the total time complexity to evaluate (13) is $O(TR)$. The algorithm is shown in Algorithm 1. We emphasize that this time complexity is independent of the size of networks (i.e, n, m) Hence this algorithm can scale to very large networks.

Algorithm 1 Monte-Carlo Single-pair SimRank

```
1: procedure SinglePair(u,v)
2:     u_1 ← u, ..., u_R ← u, v_1 ← v, ..., v_R ← v
3:     σ = 0
4:     for t = 0, 1, ..., T − 1 do
5:         for w ∈ {u_1, ..., u_R} ∩ {v_1 ..., v_R} do
6:             α ← #{r : u_r = w, r = 1, ..., R}
7:             β ← #{r : v_r = w, r = 1, ..., R}
8:             σ ← σ + c^t D_ww αβ/R^2
9:         end for
10:        for r = 1, ..., R do
11:            u_r ← δ_−(u_r), v_r ← δ_−(v_r), randomly
12:        end for
13:    end for
14:    return σ
15: end procedure
```

We give estimation of the number of samples to compute (13) accurately, with high probability. We use the Hoeffding inequality to show the following.

Proposition 3. *Let $\tilde{s}^{(T)}(u,v)$ be the output of the algorithm. Then*

$$\mathbf{P}\left\{|\tilde{s}^{(T)}(u,v) - s^{(T)}(u,v)| \geq \epsilon\right\} \leq 4nT\exp\left(-\epsilon^2 R/2(1-c)^2\right). \quad (15)$$

The proof of of Proposition 3 will be given in Appendix. By Proposition 3, we have the following.

Corollary 1. *Algorithm 1 computes the SimRank score $s^{(T)}(u,v)$ with accuracy $0 < \epsilon < 1$ with probability $0 < \delta < 1$ by setting $R = 2(1-c)^2\log(4nT/\delta)/\epsilon^2$.*

Remark 2. *Fogaras and Balázs [9] also proposed a Monte-Carlo based single-pair SimRank algorithm but their algorithm and our algorithm are conceptually different. Their algorithm is based on the random surfer-pair model of SimRank* (3)*. The issue to evaluate* (3) *is: how to maintain the random walks, i.e., it requires some path-maintaining data structure. Fogaras and Balázs used the fingerprint tree data structure for this purpose. However, the data structure is much more complicated and hence it becomes a computational bottleneck.*

On the other hand, our algorithm is based on (13)*. To evaluate* (13)*, we only have to store the t-th position of random walks and hence* (13) *can be computed very efficient.*

5. DISTANCE CORRELATION OF SIMRANK

Our top k similarity search algorithm performs single-pair SimRank computations for a given source vertex u and for other vertices v, but we save the time complexity by pruning. In order to perform this pruning, we need some upper bounds. This section and the next section are devoted to establish the upper bounds.

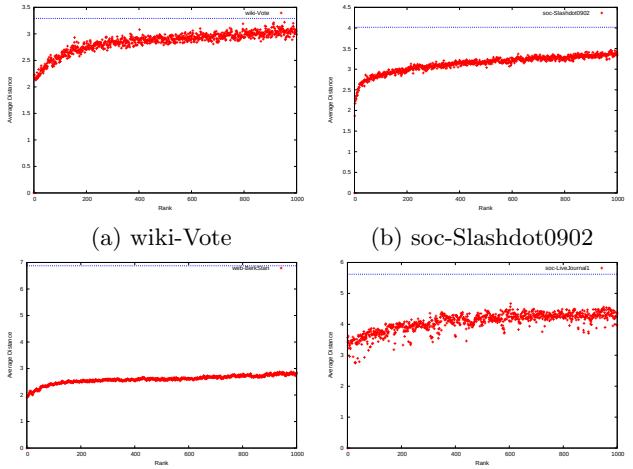
The important observation of SimRank is

> SimRank score $s(u,v)$ decays very fast as the pair u, v goes away.

In this section, we empirically verify this fact in some real networks, and in the next section, we develop the upper bounds that only depend on distance.

Let us look at Figure 2. We randomly chose 100 vertices u and enumerate top-1000 similar vertices with respect to to a query vertex u (note that these top-1000 vertices are "exact", not 'approximate"). Each point denotes the average distance of the k-th similar vertex. To convince the reader, we also give the average distance between two vertices for each network by the blue line.

Figure 2 clearly shows much intuitive information. If we only need to compute top-10 vertices, all of them are within distance two, three, or four. In real applications, it is unlikely that we need to compute top-1000 vertices, but even for this case, most of them are within distance four or five. We emphasize that these distances are smaller than the average distance of two vertices in each network. Thus we can conclude that the "candidates" of highly similar vertices are screened by distances very well.



(a) wiki-Vote (b) soc-Slashdot0902

(c) web-BerkStan (d) soc-LiveJournal1

Figure 2: Distance correlation of similarity ranking. The red points are for distance of top-1000 similar vertices and the blue line is for average distance of two vertices in each network

There is one remark we would like to make from Figure 2. The top-10 highest SimRank vertices in Web graphs are much closer to a query vertex than social networks. Thus we can also claim that our algorithm would work better for Web graphs than for social networks, because we only look at subgraphs induced by vertices of distance within three (or even two) from a query vertex. This claim is verified in Section 8.

6. TIGHT UPPER BOUNDS

In the previous section, we observe that highly similar vertices with respect to a query vertex are within small distance from u. This observation allows us to propose our efficient algorithm for the top-k similarity search problem for a single vertex (i.e., Problem 1). In order to obtain this algorithm, we need to establish the upper bounds of SimRank that depend only on distance, which will be done in this section.

Let us observe that by definition, SimRank score is bounded by the decay factor to the power of the distance:

$$s(u,v) \leq c^{d(u,v)}$$

Since almost all high SimRank score vertices with respect to a query vertex u are located within distance three from u (see Figure 2), we obtain $s(u,v) \leq c^3 = 0.216$. But this is too large for our purpose (indeed, our further experiments to compare actual SimRank scores with this bound confirm that it is too large).

Here we propose two upper bounds, called "L1 bound" and "L2 bound". Our algorithm, described in a later section, combines these two bounds to perform "pruning", which results in a much faster algorithm.

6.1 L1 bound

The first bound is based on the following inequality: for a vector x and a stochastic vector y,

$$x^\top y \leq \max_{w \in \mathrm{supp}(y)} x^\top e_w, \tag{16}$$

where $\mathrm{supp}(y) := \{w \in V : y(w) > 0\}$ is a positive support of y. We bound $(P^t e_u)^\top D(P^t e_v)$ by this inequality.

Fix a query vertex u. Let us define

$$\alpha(u,d,t) := \max_{w \in V, d(u,w)=d} (P^t e_u)^\top D e_w \tag{17}$$

for $d = 1, \ldots, d_{\max}$ and $t = 1, \ldots, T$, and

$$\beta(u,d) := \sum_{t=0}^{T-1} c^t \max_{d-t \leq d' \leq d+t} \alpha(u,d',t) \tag{18}$$

for $t = 1, \ldots, T$. Here $d_{\max}$ is distance such that if $d(u,v) > d_{\max}$ then $s(u,v)$ is too small to take into account. (We usually set $d_{\max} = T$).

PROPOSITION 4. *For a vertex v with $d(u,v) = d$, we have*

$$s^{(T)}(u,v) \leq \beta(u,d) \tag{19}$$

The proof will be given in Appendix.

REMARK 3. *$\alpha(u,d,t)$ has the following probabilistic representation:*

$$\alpha(u,d,t) = \max_{d(u,w)=d} D_{ww} \mathbf{P}\{u^{(t)} = w\}$$

where $u^{(t)}$ denotes the position of a random walk that starts from u and follows its in-links.

To compute $\alpha(u,d,t)$ and $\beta(u,d)$, we can use Monte-Carlo simulation for $P^t e_u$ as shown in Algorithm 2.

Similar to Proposition 3, we obtain the following proposition, whose proof will be given in Appendix. This proposition shows that Algorithm 2 can compute $\alpha(u,d,t)$ and $\beta(u,d)$.

PROPOSITION 5. *Let $\tilde{\beta}(u,d)$ be computed by Algorithm 2. Then*

$$\mathbf{P}\left\{|\tilde{\beta}(d,t) - \beta(d,t)| \geq \epsilon\right\} \leq 2nd_{\max}T \exp(-2\epsilon^2 R)$$

By Proposition 5, we have the following.

COROLLARY 2. *Algorithm 2 computes $\beta(u,d)$ with accuracy less than $0 < \epsilon < 1$ with probability at least $0 < \delta < 1$ by setting $R = \log(2nd_{\max}T/\delta)/(2\epsilon^2)$.*

Algorithm 2 Monte-Carlo $\alpha(u,d,t)$, $\beta(u,d)$ computation

```
1: procedure ComputeAlphaBeta(u)
2:     u_1 ← u, ..., u_R ← u
3:     for t = 0, 1, ..., T − 1 do
4:         for w ∈ {u_1, ..., u_R} do
5:             μ ← D_ww #{r : u_r = w, r ∈ [1, R]}/R
6:             α(u, d(u, w), t) ← max{α(u, d(u, w), t), μ}
7:         end for
8:         for r = 1, ..., R do
9:             u_r ← δ(u_r) randomly
10:        end for
11:    end for
12:    for d = 1, ..., T do
13:        β(u, d) = Σ_{t=0}^{T−1} c^t max_{d−t≤d'≤d+t} α(u, d', t)
14:    end for
15: end procedure
```

6.2 L2 bound

The second bound is based on the Cauchy–Schwartz inequality: for nonnegative vectors x and y,

$$x^\top y \leq \|x\| \|y\|. \tag{20}$$

We also bound $(P^t e_u) D (P^t e_v)$ by this inequality.

Let

$$\gamma(u,t) := \|\sqrt{D} P^t e_u\|, \tag{21}$$

where $\sqrt{D} = \mathrm{diag}(\sqrt{D_{11}}, \ldots, \sqrt{D_{nn}})$. Note that, since D is a nonnegative diagonal matrix, $\sqrt{D}$ is well-defined.

PROPOSITION 6. *For two vertices u and v, we have*

$$s^{(T)}(u,v) \leq \sum_{t=0}^{T} c^t \gamma(u,t) \gamma(v,t) \tag{22}$$

The proof of Proposition 6 will be given in Appendix.

To compute $\gamma(u,d)$ for each u, we can use Monte-Carlo simulation. Let us emphasize that we can compute $\gamma(u,d)$ for each u and $d \leq d_{\max}$ in preprocess.

The following proposition, whose proof will be given in Appendix, shows that Algorithm 3 can compute $\gamma(u,d)$.

PROPOSITION 7. *Let $\tilde{\gamma}(u,t)$ be computed by Algorithm 3. Then*

$$\mathbf{P}\left\{|\tilde{\gamma}(u,t) - \gamma(u,t)| \geq \epsilon\right\} \leq 4n \exp\left(-\epsilon^2 R/8\right).$$

The proof of of Proposition 7 will be given in Appendix. By Proposition 7, we have the following.

COROLLARY 3. *Algorithm 3 computes $\gamma(u,t)$ with accuracy less than $0 < \epsilon < 1$ with probability at least $0 < \delta < 1$ by setting $R = 8\log(4n/\delta)/\epsilon^2$.*

6.3 Comparison of two bounds

The reason why we need both L1 and L2 bounds is the following: The L1 bound is more effective for a low degree query vertex u. This is because if u has low degree then $P^t e_u$ is sparse. Therefore the bound (25) becomes tighter.

On the other hand, the L2 bound is more effective for high degree vertex u. This is because if u has high degree then $P^t e_u$ spreads widely, and hence each entry is small. Therefore $\|\sqrt{D} P^t e_u\|$ decrease rapidly.

Algorithm 3 Monte-Carlo $\gamma(u,t)$ computation

procedure ComputeGamma(u)
  $u_1 \leftarrow u, \ldots, u_R \leftarrow u$
  for $t = 0, 1, \ldots, T-1$ **do**
    $\mu = 0$
    for $w \in \{u_1, \ldots, u_R\}$ **do**
      $\mu \leftarrow \mu + D_{ww} \#\{r : u_r = w, r \in [1, R]\}^2 / R^2$
    end for
    $\gamma(u,t) \leftarrow \sqrt{\mu}$
    for $r = 1, \ldots, R$ **do**
      $u_r \leftarrow \delta(u_r)$ randomly
    end for
  end for
end procedure

7. ALGORITHM

We are now ready to provide our whole algorithm for top-k similarity search. Our algorithm consists of two phases: preprocess phase and query phase. In the query phase, for a given vertex u, we compute single-pair SimRanks $s(u,v)$ for some vertices v that may have high SimRank value (we call such vertices *candidates* that are computed in the preprocess phase), and output k highly similar vertices.

In order to obtain similar vertices accurately, we have to perform many Monte Carlo simulations in single-pair SimRank computation (Algorithm 1). Thus, the key of our algorithm is the way to reduce the number of candidates that are computed in the preprocess phase in Section 7.1.

7.1 Preprocess phase

In the preprocess phase, we precompute γ in (21) for the L2 bound as described in Algorithm 3. Note that we compute α, β in (17), (18) for the L1 bound in query phase.

After that, for each vertex u, we enumerate "candidates" of highly similar vertices v. For this purpose, we consider the following auxiliary bipartite graph H. The left and right vertices of H are copy of V (i.e., H has $2n$ vertices). Let u_{left} be the copy of $u \in V$ in the left vertices and let v_{right} be the copy of $v \in V$ in the right vertices. There is an edge $(u_{\text{left}}, v_{\text{right}})$ if a random walk that starts from u frequently reaches v. By this construction, a pair of vertices u and v has high SimRank score if u_{left} and v_{left} share many neighbors. We construct this bipartite graph H by performing Monte-Carlo simulations in the original graph G as follows. For each vertex u, we iterate the following procedure P times to construct an index for u. We perform a random walk W_0 of length T from u in G. We further perform Q random walks $W_1, \ldots, W_Q$ from u. Let v be t-th vertex on W_0. Then we put an edge $(u_{\text{left}}, v_{\text{right}})$ in H if there are at least two random walks in $W_1, \ldots W_Q$ that contain v at t-th step. The whole procedure is described in Algorithm 3 below. Here, for a random walk W_j and $t \geq 0$, we denote the vertex at the t-th step of W_j by W_{jt}.

In our experiment, we set $P = 10$, $T = 11$ and $Q = 5$. The time complexity of this preprocess phase is $O(n(R+PQ)T)$, where R comes from Algorithm 3 and we set $R = 100$ in our experiment. The space complexity is $O(nP)$, but in practice, since the number of candidates are usually small, the space is less smaller than this bound.

Algorithm 4 Proposed algorithm (preprocess)

procedure Indexing
  for $u \in V$ **do**
    for $i = 1, \ldots, P$ **do**
      Perform a random walk W_0 from u
      Perform random walks $W_1, \ldots, W_Q$ from u
      for $t = 1, \ldots, T$ **do**
        if $W_{jt} = W_{kt}$ for some $j \neq k$ **then**
          Add W_{0t} for index of u
        end if
      end for
    end for
  end for
end procedure

7.2 Query phase

We now describe our query phase. For a given vertex u, we first traverse the auxiliary bipartite graphs H and enumerate all the vertices v that share the neighbor in H. We then prune some vertices v by L1 and L2 bounds. After that, for each candidate v, we compute SimRank scores $s(u,v)$ by Algorithm 1. Finally we output top k similar vertices as the solution of similarity search.

To accelerate the above procedure, we use the adaptive sample technique. For a query vertex u, we first set $R = 10$ (in Algorithm 1) and estimate SimRank scores roughly for each candidate v by Monte Carlo simulation (i.e, we only perform 10 random walks for v by Algorithm 1). Then, we change $R = 100$ and re-compute more accurate SimRank scores for each candidate v that has high estimated SimRank scores by Monte Carlo simulation (i.e, we perform 100 random walks for v by Algorithm 1). The whole procedure is described in Algorithm 5.

The overall time complexity of the query phase is $O(RT|S|)$ where $|S|$ is the number of candidates, since computing SimRank scores $s(u,v)$ by Algorithm 1 for two vertices u, v takes $O(RT)$. The space complexity is $O(m + nP)$.

Algorithm 5 Proposed algorithm (query)

procedure Query(u)
  Enumerate $S := \{v | \delta_H(u_{\text{left}}) \cap \delta_H(v_{\text{left}}) \neq \emptyset\}$
  Prune S by L1 and L2 bound
  for $v \in S$ **do**
    Perform Algorithm 1 $R = 10$ times to roughly estimate $s(u,v)$.
    if The estimated score $s(u,v)$ is not small **then**
      Perform Algorithm 1 $R = 100$ times to estimate $s(u,v)$ more accurately
    end if
  end for
  Output top k similar vertices
end procedure

8. EXPERIMENT

We perform our proposed algorithm for several real networks and evaluate performance of our algorithm. We also compare our algorithm with some state-of-the-art algorithms.

All experiments are conducted on an Intel Xeon E5-2690 2.90GHz CPU with 256GB memory and running Ubuntu 12.04. Our algorithm is implemented in C++ and compiled with g++v4.6 with -O3 option.

Table 2: Dataset information[5]

Dataset	n	m
ca-GrQc	5,242	14,496
ca-HepTh	9,877	25,998
Wiki-Vote	7,155	103,689
soc-Epinions1	75,879	508,837
soc-SlashDot0811	77,360	905,468
soc-SlashDot0902	82,168	948,464
Cora-direct	225,026	714,266
web-Stanford	281,903	2,312,497
web-NotreDame	325,728	1,497,134
web-Google	875,713	5,105,049
web-BerkStan	685,230	7,600,505
dblp-2011	933,258	6,707,236
in-2004	1,382,908	17,917,053
flickr	1,715,255	22,613,981
soc-LiveJournal1	4,847,571	68,993,773
indochina-2004	7,414,866	194,109,311
it-2004	41,291,549	1,150,725,436
twitter-2010	41,652,230	1,468,365,182

According to discussion in the previous section, we set the parameters as follows: decay factor $c = 0.6$, $T = 11$, $R = 100$ for γ (Algorithm 3) and for $s(\cdot, \cdot)$ (Algorithm 1), and $R = 10000$ for α and β (Algorithm 2) that is optimized by pre-experiment[4]. We also set $P = 10$, $T = 11$ and $Q = 5$ in our preprocess phase as in Section 7.1.

In addition, we set $k = 20$ since we are only interested in small number of similar vertices. To avoid searching vertices of very small SimRank scores, we set a threshold $\theta = 0.01$ to terminate the procedure when upper bounds become smaller than θ.

We use the datasets shown in Table 2.

8.1 Results

We evaluate our proposed algorithm for several real networks. The results are summarized in Tables 4.

We first observe that our proposed algorithm can find top-20 similar vertices in less than a few seconds for graphs of billions edges (i.e., "it-2004") and in less than a second for graphs of one hundred millions edges, respectively.

We can also observe that the query time for our algorithm does not much depend on the size of networks. For example, "indochina-2004" has 8 times more edges than "flickr" but the query time is twice faster than that. Hence the computational time of our algorithm depends on the *network structure* rather than the network size. Specifically, our algorithm works better for web graphs than for social networks.

8.2 Analysis of Accuracy

In this subsection, we shall investigate performance of our algorithm in terms of accuracy. In many applications, we are only interested in very similar vertices. Hence we only look at vertices that have high SimRank scores.

Specifically, what we do is the following. We first compute, for a query vertex u, the single source SimRank scores $s(u, v)$ for all the vertices v (for the whole graph) by the exact method. Then we pick up all "high score" vertices v with score at least t from this computation (for $t = 0.04$, 0.05, 0.06, 0.07). Finally, we compute "high score" vertices v with respect to the query vertex u by our proposed algorithm. Let us point out that our algorithm can be easily modified so that we only output high SimRank score vertices(because we just need to set up the threshold to prune the similarity search). We then compute the following value:

$$\frac{\#\text{ of our high score vertices}}{\#\text{ of the optimal high score vertices}}.$$

We also do the same thing for high score vertices computed by Fogaras and Rácz [9](we used the same parameter $R' = 100$ presented in [9]). We perform this operation 100 times, and take the average. The result is in Table 3. We can see that our algorithm actually gives very accurate results. In addition, our algorithm gives better accuracy than Fogaras and Rácz [9].

8.3 Comparison with existing results

In this subsection, we compare our algorithm with two state-of-the-art algorithms for computing SimRank, and show that our algorithm outperforms significantly in terms of scalability.

Comparison with the state-of-the-art all-pairs algorithm.

Yu et al. [37] proposed an efficient all-pairs algorithm; the time complexity of their algorithm is $O(Tnm)$, and the space complexity is $O(n^2)$, where T is the number of the iterations. We implemented their algorithm and evaluated it in comparison with ours. We used the same parameters presented in [37].

Results are shown in Table 4; the omitted results (—) mean that their algorithm failed to allocate memory. From the results, we observe that their algorithm is a little faster than ours in query time, but our algorithm uses much less space(15–30 times). In fact, their algorithm failed for graphs with a million edges, because of memory allocation. More importantly, their algorithm cannot estimate the memory usage before running the algorithm. Moreover, since our all-pairs algorithm can easily be parallelized to multiple machines, if there are 100 machines, even for graphs of billions size, our all-pairs algorithm can output all top-20 vertices in less than 5 days. Thus, our algorithm significantly outperforms their algorithm in terms of scalability.

Comparison with the state-of-the-art single-pair and single-source algorithm.

Fogaras and Rácz [9] proposed an efficient single-pair algorithm that estimates SimRank scores with Monte Carlo simulation. Like our approach, their algorithm also consists of two phases, a preprocessing phase and a query phase. In the preprocessing phase, their algorithm generates R' random walks and stores the walks efficiently; this phase requires $O(nR')$ time and $O(nR')$ space. The query phase phase requires $O(TnR')$ time, where T is the number of iterations. We implemented their algorithm and evaluated

[4]These values are much smaller than our theoretical estimations. The reason is that Hoeffding bound is not tight in this case.

[5]ca-GrQc, ca-HepTh, Wiki-Vote, soc-SlashDot0811, soc-SlashDot0902, soc-Epinions1, soc-LiveJournal1 web-BerkStan web-Google, web-NotreDame, and web-Stanford data sets are available at `http://snap.stanford.edu/data/index.html` [2, 18]. in-2004, indochina-2004, it-2004, uk-2007-05, twitter-2010, and dblp-2011 data sets are available at `http://law.di.unimi.it/datasets.php` [5, 6]. flickr data set is available at `http://socialnetworks.mpi-sws.org/datasets.html` [28]. Cora [27] set is available at `http://people.cs.umass.edu/mccallum/data.html`. See the web pages for detailed information of these datasets.

Table 3: Accuracy

Dataset	Threshold	Proposed	Fogaras and Rácz [9]
ca-GrQc	0.04	0.98665	0.92329
	0.05	0.98854	0.92467
	0.06	0.99461	0.95225
	0.07	0.99554	0.92881
as20000102	0.04	0.97831	0.94643
	0.05	0.98727	0.94783
	0.06	0.99177	0.94713
	0.07	0.99550	0.94760
wiki-Vote	0.04	0.81862	0.93491
	0.05	0.88629	0.93760
	0.06	0.90801	0.94215
	0.07	0.94785	0.97916
ca-HepTh	0.04	0.97142	0.88964
	0.05	0.98782	0.94354
	0.06	0.99673	0.91848
	0.07	0.99746	0.93647

it in comparison with ours. We used the same parameter $R' = 100$ presented in [9].

We can see that their algorithm is faster in query time. But we suspect that this is due to relaxing accuracy, as in the previous subsection. In order to obtain the same accuracy as our algorithm, we suspect that R' should be 500–1000, which implies that their algorithm would be at least 5–10 times slower, and require at leats 5–10 times more space.

In this case, their algorithm would fail for graphs with more than ten millions edges because of memory allocation. Even for the case $R' = 100$, our algorithm uses much less space(10–20 times), and their algorithm failed for graphs with more than 70 millions edges because of memory allocation. Therefore we can conclude that our algorithm significantly outperforms their algorithm in terms of scalability.

9. CONCLUSION

In this paper, we consider the similarity search problem that finds top-k similar vertices from a given vertex u, and propose a very efficient and scalable algorithm.

Our algorithm is based on the new formulation of SimRank, called linear recurrence formulation. Due to this linear recursive formula, single-pair SimRank score $s(u, v)$ can be computed very efficiently by Monte-Carlo simulation (indeed the time complexity is independent of size of graphs). We also observe that SimRank score $s(u, v)$ decays very rapidly as distance of the pair u, v increases. By this observation, we establish upper bounds of SimRank score $s(u, v)$ that only depend on distance $d(u, v)$. The upper bounds can be efficiently computed by Monte-Carlo simulation (in our preprocess). Moreover these upper bounds, together with some adaptive sample technique, allow us to effectively prune the similarity search procedure.

Our experiments show that the proposed algorithm can compute top-20 similar vertices in less than a few seconds for graphs of billions edges and in less than a second for graphs of one hundred millions edges, respectively. Our algorithm requires smaller memory space than any other existing algorithms, which allows our algorithm to scale for much larger graphs (i.e., graphs with at least a billion edge for the first time).

10. REFERENCES

[1] Z. Abbassi and V. S. Mirrokni. A recommender system based on local random walks and spectral methods. In *WebKDD/SNA-KDD*, volume 5439 of *Lecture Notes in Computer Science*, pages 139–153. Springer, 2007.

[2] R. Albert, H. Jeong, and A.-L. Barabasi. Internet: Diameter of the World-Wide Web. *Nature*, 401(6749):130–131, 1999.

[3] I. Antonellis, H. Garcia-Molina, and C.-C. Chang. Simrank++: query rewriting through link analysis of the click graph. *PVLDB*, 1(1):408–421, 2008.

[4] A. A. Benczúr, K. Csalogány, and T. Sarlós. Link-based similarity search to fight web spam. In *AIRWeb*, pages 9–16, 2006.

[5] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In *WWW*, pages 587–596. ACM, 2011.

[6] P. Boldi and S. Vigna. The webgraph framework i: compression techniques. In *WWW*, pages 595–602. ACM, 2004.

[7] Y. Cai, G. Cong, X. Jia, H. Liu, J. He, J. Lu, and X. Du. Efficient algorithm for computing link-based similarity in real world networks. In *ICDM*, pages 734–739. IEEE Computer Society, 2009.

[8] L. Cao, B. Cho, H. D. Kim, Z. Li, M.-H. Tsai, and I. Gupta. Delta-SimRank computing on MapReduce. In *BigMine*, pages 28–35. ACM, 2012.

[9] D. Fogaras and B. Rácz. Scaling link-based similarity search. In *WWW*, pages 641–650. ACM, 2005.

[10] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, and M. Onizuka. Efficient search algorithm for simrank. In *ICDE*, pages 589–600. IEEE Computer Society, 2013.

[11] Z. Gyöngyi, H. Garcia-Molina, and J. O. Pedersen. Combating web spam with TrustRank. In *VLDB*, pages 576–587. Morgan Kaufmann, 2004.

[12] G. He, H. Feng, C. Li, and H. Chen. Parallel SimRank computation on large graphs with iterative aggregation. In *KDD*, pages 543–552. ACM, 2010.

[13] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *KDD*, pages 538–543. ACM, 2002.

[14] X. Jia, Y. Cai, H. Liu, J. He, and X. Du. Calculating similarity efficiently in a small world. In *ADMA*, volume 5678 of *Lecture Notes in Computer Science*, pages 175–187. Springer, 2009.

[15] X. Jia, H. Liu, L. Zou, J. He, and X. Du. A fast two-stage algorithm for computing SimRank and its extensions. In *WAIM Workshops*, volume 6185 of *Lecture Notes in Computer Science*, pages 61–73. Springer, 2010.

[16] M. M. Kessler. Bibliographic coupling extended in time: Ten case histories. *Information Storage and Retrieval*, 1(4):169–187, 1963.

[17] A. N. Langville and C. D. Meyer. *Google's PageRank and beyond: The science of search engine rankings*. Princeton University Press, 2006.

[18] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[19] C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, and T. Wu. Fast computation of simrank for static and dynamic information networks. In *EDBT*, volume 426 of *ACM International Conference Proceeding Series*, pages 465–476. ACM, 2010.

[20] P. Li, Y. Cai, H. Liu, J. He, and X. Du. Exploiting the block structure of link graph for efficient similarity computation. In *PAKDD*, volume 5476 of *Lecture Notes in Computer Science*, pages 389–400. Springer, 2009.

[21] P. Li, H. Liu, J. X. Yu, J. He, and X. Du. Fast single-pair simrank computation. In *SDM*, pages 571–582. SIAM, 2010.

[22] D. Liben-Nowell and J. M. Kleinberg. The link-prediction problem for social networks. *JASIST*, 58(7):1019–1031, 2007.

Table 4: Preprocess time, query time and space for our proposed algorithm, Fogaras and Rácz [9], and Yu et al. [37]. The single-pair and single-source results are the average of 10 trials; we omitted results of the all-pairs computation of our proposed algorithm for large networks; other omitted results (—) mean that the algorithms failed to allocate memory

Dataset	Proposed				Fogaras and Rácz [9]			Yu et al. [37]	
	Preproc.	Query	AllPairs	Index	Preproc.	Query	Index	AllPairs	Memory
ca-GrQc	1.5 s	2.6 ms	13.5 s	2.4 MB	110 ms	0.11 ms	22.7 MB	2.97 s	66 MB
as20000102	1.6 s	18 ms	115 s	3.3 MB	81 ms	1.3 ms	28.0 MB	0.13 s	51 MB
Wiki-Vote	1.9 s	3.8 ms	26.9 s	5.3 MB	110 ms	0.41 ms	31.1 MB	8.74 s	138 MB
ca-HepTh	1.8 s	3.3 ms	32.3 s	4.5 MB	253 ms	0.44 ms	43.3 MB	23.3 s	312 MB
email-Enron	7.8 s	24 ms	864 s	21.6 MB	949 ms	1.1 ms	158 MB	302 s	3.45 GB
soc-Epinions1	19.5 s	44 ms	3335 s	18.9 MB	1.8 s	1.4 ms	332 MB	777 s	6.91 GB
soc-Slashdot0811	20.2 s	53 ms	4081 s	48.6 MB	1.9 s	1.2 ms	341 MB	747 s	7,34 GB
soc-Slashdot0902	21.3 s	55 ms	4515 s	51.2 MB	2.0 s	1.3 ms	363 MB	694 s	7.21 GB
email-EuAll	55.6 s	226 ms	—	103 MB	3.6 s	14 ms	1.1 GB	—	—
web-Stanford	69.6 s	103 ms	—	141 MB	10.4 s	10 ms	1.2 GB	—	—
web-NotreDame	60.6 s	73 ms	—	163 MB	7.6 s	2.8 ms	1.4 GB	—	—
web-BerkStan	240.2 s	93 ms	—	288 MB	47.4 s	4.3 ms	3.8 GB	—	—
web-Google	156.7 s	199 ms	—	211 MB	24.0 s	13 ms	3.0 GB	—	—
dblp-2011	82.2 s	16 ms	—	144 MB	16.4 s	1.3 ms	1.4 GB	—	—
in-2004	292.9 s	95 ms	—	451 MB	46.4 s	6.8 ms	6.0 GB	—	—
flickr	622.7 s	1.5 s	—	513 MB	90.1 s	7.6 ms	7.4 GB	—	—
soc-LiveJournal	2335.9 s	431 ms	—	1.2 GB	397.9 s	27 ms	21.6 GB	—	—
indochina-2004	1585.2 s	714 ms	—	2.1 GB	—	—	—	—	—
it-2004	3.1 h	2.3 s	—	11.2 GB	—	—	—	—	—
twitter-2010	7.7 h	17.4 s	—	8.4 GB	—	—	—	—	— —

[23] Z. Lin, I. King, and M. R. Lyu. PageSim: A novel link-based similarity measure for the World Wide Web. In *Web Intelligence*, pages 687–693. IEEE Computer Society, 2006.

[24] Z. Lin, M. R. Lyu, and I. King. Extending link-based algorithms for similar web pages with neighborhood structure. In *Web Intelligence*, pages 263–266. IEEE Computer Society, 2007.

[25] Z. Lin, M. R. Lyu, and I. King. MatchSim: a novel similarity measure based on maximum neighborhood matching. *Knowledge and information systems*, 32(1):141–166, 2012.

[26] D. Lizorkin, P. Velikhov, M. N. Grinev, and D. Turdakov. Accuracy estimate and optimization techniques for simrank computation. *The VLDB Journal*, 19(1):45–66, 2010.

[27] A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval Journal*, 3(2):127–163, 2000.

[28] A. Mislove, M. Marcon, P. K. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Internet Measurement Conference*, pages 29–42. ACM, 2007.

[29] C. Scheible. Sentiment translation through lexicon induction. In *ACL (Student Research Workshop)*, pages 25–30. The Association for Computer Linguistics, 2010.

[30] H. Small. Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for Information Science*, 24(4):265–269, 1973.

[31] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[32] V. V. Williams. Multiplying matrices faster than Coppersmith-Winograd. In *STOC*, pages 887–898. ACM, 2012.

[33] X. Yin, J. Han, and P. S. Yu. LinkClus: Efficient clustering via heterogeneous semantic links. In *VLDB*, pages 427–438. ACM, 2006.

[34] L. Yu, Z. Shu, and X. Yang. SimRate: Improve collaborative recommendation based on rating graph for sparsity. In *ADMA (2)*, volume 6441 of *Lecture Notes in Computer Science*, pages 167–174. Springer, 2010.

[35] W. Yu, X. Lin, and J. Le. Taming computational complexity: Efficient and parallel SimRank optimizations on undirected graphs. In *WAIM*, volume 6184 of *Lecture Notes in Computer Science*, pages 280–296. Springer, 2010.

[36] W. Yu, X. Lin, W. Zhang, L. Chang, and J. Pei. More is simpler: Effectively and efficiently assessing node-pair similarities based on hyperlinks. *PVLDB*, 7(1):13–24, 2013.

[37] W. Yu, W. Zhang, X. Lin, Q. Zhang, and J. Le. A space and time efficient algorithm for SimRank computation. *World Wide Web*, 15(3):327–353, 2012.

[38] P. Zhao, J. Han, and Y. Sun. P-Rank: a comprehensive structural similarity measure over information networks. In *CIKM*, pages 553–562. ACM, 2009.

[39] W. Zheng, L. Zou, Y. Feng, L. Chen, and D. Zhao. Efficient SimRank-based similarity join over large graphs. *PVLDB*, 6(7):493–504, 2013.

[40] Y. Zhou, H. Cheng, and J. X. Yu. Graph clustering based on structural/attribute similarities. *PVLDB*, 2(1):718–729, 2009.

APPENDIX

In Appendix, we give omitted proofs of propositions in the paper. We first prove the non probablistic results (i.e, Propositions 1, 2, 4 and 6) and then prove other probablistic results (concentration bounds).

Proof of Proposition 1. Let $S(D)$ be a matrix that satisfies

$$S(D) = cP^\top S(D)P + D. \tag{23}$$

We first prove that $S(D)$ is well-defined. Let $\mathrm{vec} : \mathbb{R}^{n\times n} \to \mathbb{R}^{n^2}$ be the vectorization operation defined by $\mathrm{vec}(A)_{i+nj} = A_{ij}$. Using vec, (23) is written as the linear equation

$$\left(I - cP^\top \otimes P^\top\right)\mathrm{vec}(S(D)) = \mathrm{vec}(D). \tag{24}$$

Since the coefficient matrix of (24) is nonsingular, it a unique solution. This proves the well-definedness of $S(D)$.

Next, we prove that D is uniquely determined under the condition $S(D)_{ii} = 1$ for all $i \in V$. Write this condition by using (24) with a partitioned system:

$$\begin{bmatrix} I - cP_{DD} & -P_{DO} \\ -P_{OD} & I - cP_{OO} \end{bmatrix}\begin{bmatrix} \vec{1} \\ u \end{bmatrix} = \begin{bmatrix} d \\ 0 \end{bmatrix},$$

where d denotes diagonal entries of D and u be an arbitrary vector. We observe that we can eliminate u from this

equation. The eliminated system is

$$d = \left((I - cP_{DD}) - c^2 P_{DD}(I - cP_{OO})^{-1} P_{OD}\right)\vec{1}.$$

Since this formula does not depend on u, D is uniquely determined by the diagonal condition of $S(D)$. Note that the above gives an explicit formula of D.

Proof of Proposition 2. Since D satisfies $D = S - cP^\top SP$, we have

$$D_{uu} = 1 - c(Pe_u)^\top SPe_u.$$

Since Pe_u is a stochastic vector, we have

$$1 - c\max_{ij} S_{ij} \leq D_{uu} \leq 1 - c\min_{ij} S_{ij}$$

By the random surfer pair interpretation of SimRank, we have $0 \leq S_{ij} \leq 1$. Therefore we obtain $1 - c \leq D_{uu} \leq 1$.

Proof of Proposition 4. Consider $(P^t e_u)^\top D(P^t e_v)$. Since $P^t e_v$ is a stochastic vector, by (16), we have

$$(P^t e_u)^\top D(P^t e_v) \leq \max_{w \in \mathrm{supp}(P^t e_v)} (P^t e_u)^\top De_w, \qquad (25)$$

Since $P^t e_v$ corresponds to a t-step random walk, the support is contained by a ball of radius t centered at v. Therefore we have

$$(P^t e_u)^\top D(P^t e_v) \leq \max_{w \in V : d-t \leq d(u,w) \leq d+t} (P^t e_u)^\top De_w$$

By plugging (17), we have

$$(P^t e_u)^\top D(P^t e_v) \leq \max_{d-t \leq d' \leq d+t} \alpha(u, d', t).$$

Substitute the above to (9), we obtain (19).

Proof of Proposition 6. Consider $c^t (P^t e_u)^\top D(P^t e_v)$. By (20), we have

$$(P^t e_u)^\top D(P^t e_v) = (\sqrt{D}P^t e_u)^\top (\sqrt{D}P^t e_v) \leq \gamma(u,t)\gamma(v,t).$$

Substitute the above to (9), we obtain (22).

Let us start the proof of concentration bounds. We first prepare some basic probablistic inequalities.

Lemma 1 (Hoeffding's inequality). *Let $X_1, \ldots, X_R$ be independent random variables with $X_r \in [0,1]$ for all $r = 1, \ldots, R$. Let $S := (X_1 + \cdots + X_R)/R$. Then*

$$\mathbf{P}\left\{|S - \mathbf{E}[S]| \geq \epsilon\right\} \leq 2\exp(-2\epsilon^2 R).$$

Lemma 2 (Max-Hoeffding's inequality). *For each $f = 1, \ldots, F$, let $X_r(f)$ $(r = 1, \ldots, R)$ be independent random variables with $X_r(f) \in [0,1]$. Let $S(f) := (X_1(f) + \cdots + X_R(f))/R$. Then*

$$\mathbf{P}\left\{\max_f |S(f) - \mathbf{E}[S(f)]| \geq \epsilon\right\} \leq 2F\exp(-2\epsilon^2 R).$$

Proof.

$$\mathbf{P}\left\{\max_f |S(f) - \mathbf{E}[S(f)]| \geq \epsilon\right\} \leq \sum_f \mathbf{P}\left\{|S(f) - \mathbf{E}[S(f)]| \geq \epsilon\right\} \leq 2F\exp(-2\epsilon^2 R).$$

We write $u_r^{(t)}$ and $v_r^{(t)}$ $(r = 1, \ldots, R)$ for the t-th positions of independent random walks start from u and v and follow the in-links, respectively, and $X_u^{(t)} := (1/R)\sum_{r=1}^R e_{u_r^{(t)}}$, $X_v^{(t)} := (1/R)\sum_{r=1}^R e_{v_r^{(t)}}$.

Lemma 3. *For each $w \in V$,*

$$\mathbf{P}\left\{\left|X_u^{(t)\top} De_w - (P^t e_u)^\top De_w\right| \geq \epsilon\right\} \leq 2\exp(-2\epsilon^2 R).$$

Proof.

$$\mathbf{P}\left\{\left|X_u^{(t)\top} De_w - (P^t e_u)^\top De_w\right| \geq \epsilon\right\} \leq \mathbf{P}\left\{\left|X_u^{(t)^\top} e_w - (P^t e_u)^\top e_w\right| \geq \epsilon\right\} \leq 2\exp(-2\epsilon^2 R).$$

Lemma 4.

$$\mathbf{P}\left\{\left|X_u^{(t)\top} DX_v^{(t)} - (P^t e_u)^\top DP^t e_v\right| \geq \epsilon\right\} \leq 4n\exp(-\epsilon^2 R/2).$$

Proof.

$$\begin{aligned}
&\mathbf{P}\left\{\left|X_u^{(t)\top} DX_v^{(t)} - (P^t e_u)^\top DP^t e_v\right| \geq \epsilon\right\} \\
&\leq \mathbf{P}\left\{\left|X_u^{(t)\top} D\left(X_v^{(t)} - P^t e_v\right)\right| \geq \epsilon/2\right\} \\
&+ \mathbf{P}\left\{\left|\left(X_u^{(t)} - P^t e_u\right)^\top DP^t e_v\right| \geq \epsilon/2\right\} \\
&\leq 4n\exp(-\epsilon^2 R/2).
\end{aligned}$$

Proof of Proposition 3. By Lemma 4, we have

$$\begin{aligned}
&\mathbf{P}\left\{\left|\left(\sum_{t=0}^{T-1} c^t X_u^{(t)\top} DX_v^{(t)}\right) - s^{(T)}(u,v)\right| \geq \epsilon\right\} \\
&\leq \sum_{t=0}^{T-1} \mathbf{P}\left\{\left|c^t X_u^{(t)\top} DX_v^{(t)} - c^t (P^t e_u)^\top DP^t e_v\right| \geq c^t\epsilon/(1-c)\right\} \\
&\leq 4nT\exp\left(-\epsilon^2 R/2(1-c)^2\right).
\end{aligned}$$

Proof of Proposition 5. We first prove the bound of $\alpha(u,d,t)$. Note that $\alpha(u,d,t) = \max_w\{P^t e_u De_w\}$ and the algorithm computes $\tilde{\alpha}(u,d,t) = \max_w\{X_u^{(t)^\top} De_w\}$. By Lemmas 3 and 2, we have

$$\begin{aligned}
&\mathbf{P}\left\{\left|\max_w X_u^{(t)\top} De_w - \max_w (P^t e_u)^\top De_w\right| \geq \epsilon\right\} \\
&\leq \mathbf{P}\left\{\max_w \left|X_u^{(t)\top} De_w - (P^t e_u)^\top De_w\right| \geq \epsilon\right\} \\
&\leq 2n\exp(-2\epsilon^2 R).
\end{aligned}$$

Using the above estimation, we bound β as

$$\begin{aligned}
&\mathbf{P}\left\{\left|\tilde{\beta}(u,d) - \beta(u,d)\right| \geq \epsilon\right\} \\
&\leq \sum_{d,t} \mathbf{P}\left\{|\tilde{\alpha}(u,d,t) - \alpha(u,d,t)| \geq \epsilon\right\} \\
&\leq 2nd_{\max}T\exp(-2\epsilon^2 R).
\end{aligned}$$

Proof of Proposition 7. We first observe that $\gamma(u,t)^2 = (P^t e_u)^\top D(P^t e_u)$ and the algorithm estimates this value by

$$(\tilde{\gamma}(u,t))^2 = \frac{1}{R^2} D_{u_r^{(t)} u_r^{(t)}}.$$

Hence, by the same proof as Lemma 4, we have

$$\mathbf{P}\left\{|\tilde{\gamma}(u,t)^2 - \gamma(u,t)^2| \geq \epsilon\right\} \leq 4n\exp(-\epsilon^2 R/2).$$

Therefore

$$\begin{aligned}
&\mathbf{P}\left\{|\tilde{\gamma}(u,t) - \gamma(u,t)| \geq \epsilon\right\} \\
&\leq \mathbf{P}\left\{|\tilde{\gamma}(u,t)^2 - \gamma(u,t)^2| \geq \frac{\epsilon}{\tilde{\gamma}(u,t) + \gamma(u,t)}\right\} \\
&\leq \mathbf{P}\left\{|\tilde{\gamma}(u,t)^2 - \gamma(u,t)^2| \geq \epsilon/2\right\} \leq 4n\exp(-\epsilon^2 R/8).
\end{aligned}$$

Here we use the fact that both $\tilde{\gamma}(u,t)$ and $\gamma(u,t)$ are smaller than $\sqrt{\max_w D_{ww}} = 1$.