Efficient SimRank Computation via Linearization

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

Mitsuru Kusumoto*
Preferred Infrastructure, Inc
mkusumoto@preferred.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 measure that has been successfully used in numerous applications. While there are many algorithms proposed for computing SimRank, their computational costs are very high.

In this paper, we propose a framework for computing SimRank accurately. Our framework is based on the novel technique, *linearized SimRank*, which provides efficient algorithms for the single-pair, single-source, and all-pairs SimRank problems, respectively. More specifically, our framework consists of two phases, a preprocessing phase and a query phase. In the preprocessing phase, we estimate a parameter for a given graph and a desired accuracy by Monte Carlo simulation, and then in the query phase, we efficiently solve SimRank problems using this preprocessed parameter. Our algorithms are efficient in both time and space. The preprocessing phase is performed in nearly linear time and the query phase requires only linear time for single-pair and single-source problems; furthermore, the space complexity is linear for both preprocessing and query phase.

We conducted experiments to evaluate our algorithms. For small networks ($n \leq 1,000,000$), our algorithm required only a few minutes for preprocessing, and then answered single-pair and single-source queries in 100 and 300 milliseconds, respectively. All-pairs computation was performed in a few days. For large networks ($n \geq 40,000,000$), our algorithm required a few hours for preprocessing, and then answered single-pair and single-source queries approximately in 10 seconds and in a half minutes, respectively.

In comparison to previous studies, for the all-pairs problem, our algorithm uses significantly less memory than any existing algorithm; thus it scales for much larger networks. For the single-pair and the single-source problems, our algorithm requires much fewer random samples than any existing Monte Carlo based algorithm for the same accuracy solution.

*supported by JST, ERATO, Kawarabayashi Project

KDD '14, August 24 - 27 2014, New York, NY, USA

http://dx.doi.org/10.1145/2623330.2623696.

Keywords

Similarity measure; SimRank; linearized SimRank

1. INTRODUCTION

1.1 SimRank

In recent years, much effort has been devoted to extracting useful information from large graphs. With the rapidly increasing amount of graph data, the *similarity search problem*, which identifies similar vertices in a graph, has become an important problem with many applications, including web analysis [16, 27], graph clustering [38, 45], spam detection [14], computational advertisement [4], recommender systems [2, 39], and natural language processing [34].

Several similarity measures have been proposed [20, 28, 29, 30, 35]. In this paper, we focus on *SimRank* [16]. SimRank is a link-based similarity measure proposed for similarity searches on the World Wide Web. The fundamental idea behind SimRank is that

> two pages are similar if they are referenced by similar pages.

This intuition is formulated as follows. Let $G = (V, E)$ be a directed graph with vertex set $V = \{1, \ldots, n\}$. *SimRank* $s(i, j)$ of vertex i and j is recursively defined by

$$s(i,j) = \begin{cases} 1, & (i = j), \\ \dfrac{c}{|\mathrm{In}(i)||\mathrm{In}(j)|} \displaystyle\sum_{i' \in \mathrm{In}(i), j' \in \mathrm{In}(j)} s(i', j'), & (i \neq j), \end{cases} \tag{1}$$

where $\mathrm{In}(i) = \{j \in V : (j, i) \in E\}$ is the set of in-neighbors of i, and $c \in (0, 1)$ is a decay factor usually set to $c = 0.8$ [16] or $c = 0.6$ [31]. SimRank and its related measures [4,9,41,43] give high-quality scores in many application areas, because it takes multi-step neighborhoods into account, whereas other similarity measures, such as the Jaccard coefficient, utilize only one-step neighborhoods.

While there are several algorithms proposed to compute SimRank scores, its computation cost is very high. More precisely, there are three fundamental SimRank problems:

Single-pair SimRank. Given two vertices $i, j \in V$, compute SimRank score $s(i, j)$.

Single-source SimRank. Given a vertex $i \in V$, compute SimRank scores $s(i, j)$ for all $j \in V$.

All-pairs SimRank. Compute SimRank scores $s(i, j)$ for all $i, j \in V$.

Table 1: Summary of SimRank algorithms on sparse graphs, where n is the number of vertices, m is the number of edges, T is the number of SimRank iterations, R and L are parameters that depend on the desired accuracy for our algorithm, and R' is the parameter for the algorithm [11].

Algorithm	Preprocessing	Query Type	Time	Memory	Technique
Proposed	$O(TRLn)$	Single-pair	$O(Tm)$	$O(m)$	Linearization
		Single-source	$O(T^2m)$		
		All-pairs	$O(T^2nm)$		
Fogaras and Rácz [11]	$O(TR'n)$	Single-pair	$O(TR')$	$O(m+nR')$	Random surfer pair (Monte Carlo)
		Single-source	$O(TR'n)$		
Li et al. [26]	—	Single-pair	$O(Tm^2)$	$O(n^2)$	Random surfer pair (Iterative)
Jeh and Widom [16]	—	All-pairs	$O(Tm^2)$	$O(n^2)$	Naive
Lizorkin et al. [31]	—	All-pairs	$O(Tnm)$	$O(n^2)$	Partial sum
Yu et al. [42]	—	All-pairs	$O(Tnm)$	$O(n^2)$	Fast matrix multiplication

Most existing algorithms solve the all-pairs SimRank problem using $O(n^2)$ space, where n is the number of vertices; thus they are impractical for large graphs. A few algorithms solve the single-pair and single-source problems, but they also have accuracy and complexity issues, which we describe in Section 5.3.

Therefore, the purpose of this paper is to propose efficient algorithms, in both time and space, for these SimRank problems.

1.2 Related work

In this subsection, we review existing algorithms for SimRank computation.

All-pairs SimRank.

In the original paper by Jeh and Widom [16], SimRank is computed by recursively evaluating equation (1) for all $i, j \in V$. This naive computation yields an $O(Tm^2)$ time algorithm, where m denotes the number of edges, and T denotes the number of iterations. Lizorkin et al. [31] proposed a *partial sum* technique to reduce the time complexity to $O(T\min\{nm, n^3/\log n\})$. Jia et al. [19] removed dangling vertices, which are vertices with no in-links, since the SimRank scores of these vertices are always zero. Yu et al. [42] introduced a matrix-based iteration approach and applied a fast matrix multiplication algorithm [36, 37] that yields an $O(T\min\{nm, n^\omega\})$ time algorithm, where $\omega < 2.373$ is the exponent of matrix multiplication. Cai et al. [8] proposed an algorithm that discards SimRank scores less than a predefined threshold, a technique already employed in previous algorithms [31, 42]. They showed that this technique works effectively on scale-free networks. Jia et al. [18] proposed an algorithm to maintain only vertex pairs of small distances and showed that this technique works effectively on small-world networks.

We emphasize that for all existing algorithms for the all-pairs problem, the space complexity of these algorithms is $O(n^2)$, since they must maintain all SimRank scores for each pair of nodes to evaluate equation (1).

Single-pair SimRank.

Jeh and Widom [16] provided a random-walk interpretation of SimRank, called a *random surfer pair model*. Consider two random walks that start from vertices i and j respectively, and follow their in-links. Let $\tau(i, j)$ be the random variable that denotes the first meeting time of i and j. Then the SimRank score is obtained by

$$s(i, j) = \mathbf{E}[c^{\tau(i,j)}], \tag{2}$$

where $\mathbf{E}$ denotes the expectation. Fogaras and Rácz [11] evaluated the right hand side via Monte Carlo simulation with a fingerprint tree data structure. This requires $O(nR')$ space to maintain R' random walks for each vertex, and computes a SimRank score $s(i, j)$ in $O(TR')$ time. Furthermore, they extended their method for single-source query that simultaneously computes single-pair queries for (possibly) similar vertices by traversing the fingerprint tree; this yields an $O(TR'n)$ time algorithm. Li et al. [26] also used the random surfer pair model, but their algorithm is deterministic; more specifically, their algorithm is an iterative algorithm for the first meeting time, and requires $O(n^2)$ space.

Note that some papers [12,25,40,41] proposed spectral decomposition (or low-rank decomposition) based algorithms; however, unfortunately, their algorithms do not actually compute SimRank, because their algorithms are based on an incorrect formula; see Remark 1 in Section 2 below.

1.3 Contribution

In this paper, we propose a framework for efficiently computing SimRank with an arbitrary accuracy ϵ (i.e., the error is less than ϵ). Our framework consists of two phases, a *preprocessing phase* and a *query phase*. In the preprocessing phase, we estimate a diagonal matrix D, which is defined in Section 2 and which plays a crucial role in our framework, for a given graph and an accuracy ϵ, and in the query phase, by using this estimated matrix D, we solve single-pair, single-source, and all-pairs SimRank problems. The obtained algorithms are efficient in both time and space, as described below.

Efficient in time. The preprocessing phase to estimate a matrix D is an iterative algorithm with Monte Carlo simulation (Subsections 4.1 and 4.2). The time complexity of this phase is $O(TLRn)$, where T is the number of iterations to compute SimRank scores, R is the number of Monte Carlo samples, and L is the number of iterations for the estimation algorithm. It can be shown that $R = O((\log n)/\epsilon^2)$ and $L = O(\log(n/\epsilon))$ (Subsection 4.3). Thus, the time complexity is expected to be nearly linear. Note that the preprocessing phase is the most computationally intensive part of our framework.

Once this matrix D is estimated, SimRank scores can be computed by deterministic algorithms (Section 3). The single-pair problem can be solved in $O(Tm)$ time, the single-

source problem can be solved in $O(T^2m)$ time, and the all-pairs problem can be solved in $O(T^2nm)$ time by solving single-source problems for all vertices. Note that our all-pairs algorithm can easily be parallelized to multiple machines, i.e., if we have M machines, it can run in $O(T^2nm/M)$ time.

Efficient in space. In both the preprocessing and query phases, the algorithms only require $O(n)$ extra space to store a diagonal matrix D and vectors of length n. thus the total space complexity is $O(m)$. Note that for the all-pairs case, we do not have to *store* the similarity of all pairs of vertices. We emphasize that this is the first linear space algorithm to compute all-pairs SimRank scores.

We evaluated our algorithms using real networks in Section 5. For small networks ($n \leq 1,000,000$), our algorithm took only a few minutes for preprocessing, and then answered single-pair and single-source queries in 100 and 300 milliseconds (on average), respectively. All-pairs computation was performed in a few days. For large networks ($n \geq 40,000,000$), our algorithm required a few hours for preprocessing, and then answered single-pair and single-source queries approximately in 10 seconds and in a half minutes (on average), respectively. To the best of our knowledge, our experiment is performed on the largest datasets in the literature.

Comparison with other existing algorithms. We compare our algorithm with the existing ones in Subsection 5.3. Our algorithms have the following advantages:

1. The state-of-the-art all-pairs algorithm, proposed by Yu et al. [42], requires much more memory, and hence works only for small networks which has $n \leq$ 1,000,000 vertices. On the other hand, our 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 [11] also requires much more space.

Our framework is based on a novel technique, named *linearized SimRank*. In Section 2, we observe that the computational difficulty of SimRank comes from its *non-linearity*. We introduced linearized SimRank to overcome this difficulty, which enabled us to compute SimRank scores via a *linear recurrence equation*.

1.4 Organization

In Section 2, we introduce the linearized SimRank, which is the main idea of our paper. In Section 3, we propose algorithms for three fundamental problems for SimRank, which are based on the linearized SimRank. In Section 4, we propose a parameter estimation algorithm, which is required for preprocessing. In Section 5, we evaluate our algorithm on several real networks in terms of accuracy and efficiency. We also give comparisons with existing algorithms. Finally in Section 6, we conclude our paper and describe future work.

2. LINEARIZED SIMRANK

Let us first observe the difficulty in computing SimRank. Let $G = (V, E)$ be a directed graph, and let $P = (P_{ij})$ be a transition matrix of transpose graph $G^\top$ defined by

$$P_{ij} := \begin{cases} 1/|\mathrm{In}(j)|, & (i,j) \in E, \\ 0, & (i,j) \notin E, \end{cases}$$

where $\mathrm{In}(i) = \{j \in V : (j,i) \in E\}$ denotes the in-neighbors of $i \in V$. Let $S = (s(i,j))$ be the *SimRank matrix*, whose (i,j) entry is the SimRank score of i and j. Then the SimRank equation (1) is represented [42] by:

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

where I is the identity matrix, and $\vee$ denotes the element-wise maximum, i.e., (i,j) entry of the matrix $A \vee B$ is given by $\max\{A_{ij}, B_{ij}\}$.

In our view, the difficulty in computing SimRank via equation (3) comes from the element-wise maximum, which is a *non-linear* operation. To avoid the element-wise maximum, we introduce a new formulation of SimRank as follows. By observing (3), since S and $cP^\top SP$ only differ in their diagonal elements, there exists a diagonal matrix D such that

$$S = cP^\top SP + D. \tag{4}$$

We call such a matrix D the *diagonal correction matrix*. The main idea of our approach here is to split the SimRank computation problem into the following two subproblems:

1. Estimate diagonal correction matrix D, and

2. compute SimRank using D and the linear recurrence equation (4).

The second step is straightforward and can be efficiently performed (Section 3). The first step is the difficult portion of our framework. For efficient computation, we must estimate D without computing the whole part of S.

For the remainder of this paper, to simplify the discussion, we introduce the notion of *linearized SimRank*. Let Θ be an $n \times n$ matrix. A linearized SimRank $S^L(\Theta)$ is a matrix that satisfies the following *linear* recurrence equation:

$$S^L(\Theta) = cP^\top S^L(\Theta)P + \Theta. \tag{5}$$

Below, we provide an example that illustrates what linearized SimRank is.

EXAMPLE 1 (STAR GRAPH OF ORDER 4). *Let G be a star graph of order 4 (i.e., G has one vertex of degree three and three vertices of degree one). 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

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

Thus, the diagonal correction matrix D is obtained by

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

Algorithm 1 Single-pair SimRank

procedure SINGLEPAIRSIMRANK(i,j)
  $\alpha \leftarrow 0$, $x \leftarrow e_i$, $y \leftarrow e_j$
  for $t = 0, 1, \ldots, T-1$ **do**
    $\alpha \leftarrow \alpha + c^t x^\top D y$, $x \leftarrow Px$, $y \leftarrow Py$
  end for
  Report $S_{ij} = \alpha$
end procedure

REMARK 1. *Some papers have used the following formula for SimRank (e.g., equation (2) in [12], equation (2) in [15], equation (2) in [25], and equation (3) in [41]):*

$$S = cP^\top SP + (1-c)I. \tag{6}$$

However, this formula does not hold; (6) *requires diagonal correction matrix D to have the same diagonal entries, but Example 1 is a counterexample. In fact, matrix S defined by* (6) *is a linearized SimRank $S^L(\Theta)$ for a matrix $\Theta = (1-c)I$.*

We provide some basic properties of linearized SimRank in Appendix.

3. LINEARIZED SIMRANK COMPUTATION

In this section, we present our proposed algorithms for SimRank by assuming that the diagonal correction matrix D has already been obtained. All algorithms are based on the same fundamental idea; i.e., in (4), by recursively substituting the left hand side into the right hand side, we obtain the following series expansion:

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

Our algorithms compute SimRank by evaluating the first T terms of the above series. The time complexity of the algorithms are $O(Tm)$ for the single-pair problem, $O(T^2 m)$ for the single-source problem, and $O(T^2 nm)$ for the all-pairs problem. For all problems, the space complexity is $O(m)$.

3.1 Single-pair SimRank

Let e_i be the i-th unit vector ($i = 1, \ldots, n$); then SimRank score $s(i,j)$ is obtained via the (i,j) component of SimRank matrix S, i.e., $s(i,j) = e_i^\top S e_j$. Thus, by applying $e_i^\top$ and e_j to both sides of (7), we obtain

$$e_i^\top S e_j = e_i^\top D e_j + c(Pe_i)^\top DPe_j + c^2 (P^2 e_i)^\top DP^2 e_j + \cdots. \tag{8}$$

Our single-pair algorithm (Algorithm 1) evaluates the right-hand side of (8) by maintaining $P^t e_i$ and $P^t e_j$. The time complexity is $O(Tm)$ since the algorithm performs $O(T)$ matrix vector products for $P^t e_i$ and $P^t e_j$ ($t = 1, \ldots, T-1$).

3.2 Single-source SimRank

For the single-source problem, to obtain $s(i,j)$ for all $j \in V$, we need only compute vector Se_i, because its j-th component is $s(i,j)$. By applying e_i to (7), we obtain

$$Se_i = De_i + cP^\top DPe_i + c^2 P^{\top 2} DP^2 e_i + \cdots. \tag{9}$$

Our single-source algorithm (Algorithm 2) evaluates the right hand side of (9) by maintaining $P^t e_i$ and $P^t e_j$. The time complexity is $O(T^2 m)$ since it performs $O(T^2)$ matrix vector products for $P^{\top t} DP^t e_i$ ($t = 1, \ldots, T-1$).

Algorithm 2 Single-source SimRank

procedure SINGLESOURCESIMRANK(i)
  $\gamma \leftarrow \vec{0}$, $x \leftarrow e_i$
  for $t = 0, 1, \ldots, T-1$ **do**
    $\gamma \leftarrow \gamma + c^t P^{\top t} Dx$, $x \leftarrow Px$
  end for
  Report $S_{ij} = \gamma_j$ for $j = 1, \ldots, n$
end procedure

Algorithm 3 All-pairs SimRank

procedure ALLPAIRSSIMRANK
  for $i = 1, \ldots, n$ **do**
    Compute SingleSourceSimRank(i)
  end for
end procedure

3.3 All-pairs SimRank

Computing all-pairs SimRank is an expensive task for a large network, because it requires $O(n^2)$ time since the number of pairs is n^2. To compute all-pairs SimRank, it is best to avoid using $O(n^2)$ space.

Our all-pairs SimRank algorithm applies the single-source SimRank algorithm (Algorithm 2) for all initial vertices, as shown in Algorithm 3. The complexity is $O(T^2 nm)$ time and requires only $O(m)$ space. Since the best-known all-pairs SimRank algorithm [31] requires $O(Tnm)$ time and $O(n^2)$ space, our algorithm significantly improves the space complexity and has almost the same time complexity (since the cost of factor T is much smaller than n or m).

It is worth noting that this algorithm is distributed computing friendly. If we have M machines, we assign initial vertices to each machine and independently compute the single-source SimRank. Then the computational time is reduced to $O(T^2 nm/M)$. This shows the scalability of our all-pairs algorithm.

4. DIAGONAL CORRECTION MATRIX ESTIMATION

As seen in the previous section, once diagonal correction matrix D is obtained, SimRank computation is a straightforward task. In this section, we show how to estimate diagonal correction matrix D.

We first observe that the diagonal correction matrix is uniquely determined from the diagonal condition.

PROPOSITION 1. *A diagonal matrix D is the diagonal correction matrix, i.e., $S^L(D) = S$ if and only if D satisfies*

$$S^L(D)_{kk} = 1, \quad (k = 1, \ldots, n), \tag{10}$$

where $S^L(D)_{kk}$ denotes (k,k) entry of the linearized SimRank matrix $S^L(D)$.

PROOF. See Appendix. □

This proposition shows that the diagonal correction matrix can be estimated by solving equation (10). Furthermore, we observe that, since S^L is a linear operator, (10) is a *linear equation* with n real variables $D_{11}, \ldots, D_{nn}$ where $D = \mathrm{diag}(D_{11}, \ldots, D_{nn})$. Therefore, we can apply a numerical linear algebraic method to estimate matrix D.

Algorithm 4 Diagonal estimation algorithm.

procedure DIAGONALESTIMATION
  Set initial guess of D.
  for $\ell = 1, \ldots, L$ **do**
    for $k = 1, \ldots, n$ **do**
      $\delta \leftarrow (1 - S^L(D)_{kk})/S^L(E^{(k,k)})_{kk}$
      $D_{kk} \leftarrow D_{kk} + \delta$
    end for
  end for
  return D
end procedure

The problem for solving (10) lies in the complexity; a naive method (Algorithm 1) requires $O(Tm)$ time to evaluate $S^L(D)_{kk}$ for each k, but this is very expensive. To reduce the complexity, we combine an *alternating method* (a.k.a. the *Gauss-Seidel method*) with *Monte Carlo simulation*. The complexity of the obtained algorithm is $O(TLRn)$ time, where L is the number of iterations for the alternating method, and R is the number of Monte Carlo samples. We analyze the upper bound of parameters L and R for sufficient accuracy in Subsection 4.3 below.

4.1 Alternating method for diagonal estimation

Our algorithm is motivated by the following intuition:

$$\text{A } (k,k) \text{ diagonal entry } S^L(D)_{kk} \text{ is the most affected by the } (k,k) \text{ diagonal entry } D_{kk} \text{ of } D. \tag{11}$$

This intuition leads to the following iterative algorithm. Let D be an initial guess[1]; for each $k = 1, \ldots, n$, the algorithm iteratively updates D_{kk} to satisfy $S^L(D)_{kk} = 1$. The update is performed as follows. Let $E^{(k,k)}$ be the matrix whose (k,k) entry is one, with the other entries being zero. To update D_{kk}, we must find $\delta \in \mathbb{R}$ such that

$$S^L(D + \delta E^{(k,k)})_{kk} = 1.$$

Since S^L is linear, the above equation is solved as follows:

$$\delta = \frac{1 - S^L(D)_{kk}}{S^L(E^{(k,k)})_{kk}}. \tag{12}$$

This algorithm is shown in Algorithm 4.

Mathematically, the intuition (11) shows the *diagonally dominant* property of operator S^L. Furthermore, the obtained algorithm (i.e., Algorithm 4) is the *Gauss-Seidel* method for a linear equation. Since the Gauss-Seidel method converges for a diagonally dominant operator [13], Algorithm 4 converges to the diagonal correction matrix[2] .

4.2 Monte Carlo based evaluation

For an efficient implementation of our diagonal estimation algorithm (Algorithm 4), we must establish an efficient method to estimate $S^L(D)_{kk}$ and $S^L(E^{(k,k)})_{kk}$.

Consider a random walk that starts at vertex k and follows its in-links. Let $k^{(t)}$ denote the location of the random walk after t steps. Then we have

$$\mathbf{E}[e_{k^{(t)}}] = P^t e_k.$$

We substitute this representation into (8) and evaluate the expectation via Monte Carlo simulation. Let $k_1^{(t)}, \ldots, k_R^{(t)}$ be R independent random walks. Then for each step t, we have estimation

$$(P^t e_k)_i \approx \#\{r = 1, \ldots, R : k_r^{(t)} = i\}/R =: p_{ki}^{(t)}. \tag{13}$$

Thus the t-th term of (8) for $i = j = k$ is estimated as

$$(P^t e_k)^\top D P^t e_k \approx \sum_{i=1}^{n} p_{ki}^{(t)2} D_{ii}. \tag{14}$$

We therefore obtain Algorithm 5 for estimating $S^L(D)_{kk}$ and $S^L(E^{(k,k)})_{kk}$.

Algorithm 5 Estimate $S^L(D)_{kk}$ and $S^L(E^{(k,k)})_{kk}$.

$\alpha \leftarrow 0$, $\beta \leftarrow 0$, $k_1 \leftarrow k, k_2 \leftarrow k, \ldots, k_R \leftarrow k$
for $t = 0, 1, \ldots, T-1$ **do**
  for $i \in \{k_1, k_2, \ldots, k_R\}$ **do**
    $p_{ki}^{(t)} \leftarrow \#\{r = 1, \ldots, R : k_r = i\}/R$
    if $i = k$ **then**
      $\alpha \leftarrow \alpha + c^t p_{ki}^{(t)2}$
    end if
    $\beta \leftarrow \beta + c^t p_{ki}^{(t)2} D_{ii}$
  end for
  for $r = 1, \ldots, R$ **do**
    $k_r \leftarrow \delta_-(k_r)$ randomly
  end for
end for
return $S^L(D)_{kk} \approx \alpha$, $S^L(E^{(k,k)})_{kk} \approx \beta$.

Using a hash table, we can implement Algorithm 5 in $O(TR)$ time, where R denotes the number of samples and T denotes the maximum steps of random walks that are exactly the number of SimRank iterations. Therefore Algorithm 4 is performed in $O(TLRn)$ time, where L denotes the number of iterations required for Algorithm 4.

4.3 Correctness: Accuracy of the algorithm

To complete the algorithm, we provide a theoretical estimation of parameters L and R that are determined in relation to the desired accuracy. In Section 5, we experimentally evaluate the accuracy.

Estimation of the number of iterations L. The convergence rate of the Gauss-Seidel method is linear; i.e., the squared error $\sqrt{\sum_{k=1}^{n}(S^L(D)_{kk} - 1)^2}$ at l-th iteration of Algorithm 4 is estimated as $O(\rho^l)$, where $0 \leq \rho < 1$ is a constant (i.e., the spectral radius of the iteration matrix). Therefore, since the error of an initial solution is $O(n)$, the number of iterations L of Algorithm 4 is estimated as $O(\log(n/\epsilon))$ for desired accuracy ϵ.

Estimation of the number of samples R. Since the algorithm is a Monte Carlo simulation, there is a trade-off between accuracy and the number of samples R. The dependency is estimated by the Hoeffding inequality, which is described below.

[1]We discuss an initial solution in Remark 2 in Appendix.
[2]Strictly speaking, we need some conditions for the diagonally dominant property of operator S^L. In practice, we can expect the estimation algorithm converges; see Lemma 1 in Appendix.

PROPOSITION 2. *Let* $p_{ki}^{(t)}$ $(i = 1, \ldots, n)$ *be defined by* (13), *and let* $p_k^{(t)} := (p_{k1}^{(t)}, \ldots, p_{kn}^{(t)})$. *Then*

$$\mathbf{P}\left\{\|P^t e_k - p_k^{(t)}\| > \epsilon\right\} \leq 2n \exp\left(-\frac{(1-c)R\epsilon^2}{2}\right).$$

where $\mathbf{P}$ *denotes the probability.*

PROOF. See Appendix. □

This shows that we need $R = O((\log n)/\epsilon^2)$ samples to accurately estimate $P^t e_k$ via Monte Carlo simulation.

By combining all estimations, we conclude that diagonal correction matrix D is estimated in $O(Tn \log(n/\epsilon)(\log n)/\epsilon^2)$ time. Since this is nearly linear time, the algorithm scales well.

Note that the accuracy of our framework only depends on the accuracy of the diagonal estimation, i.e., if D is accurately estimated, the SimRank matrix S are accurately estimated by $S^L(D)$; see Proposition 4 in Appendix. Therefore, if we want accurate SimRank scores, we only need to spend more time in the preprocessing phase and fortunately do not need to increase the time required in the query phase.

5. EXPERIMENTS

In this section, we evaluate our algorithm via experiments using real networks. The datasets we used are shown in Table 2[3]. We first evaluate the accuracy in Subsection 5.1, then evaluate the efficiency in Subsection 5.2; and finally, we compare our algorithm with some existing ones in Subsection 5.3.

For all experiments, we used decay factor $c = 0.6$, as suggested by Lizorkin et al. [31], and the number of SimRank iterations $T = 11$, which is the same as Fogaras and Rácz [11].

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

5.1 Accuracy

The accuracy of our framework depends on the accuracy of the estimated diagonal correction matrix, computed via Algorithm 4. As discussed in Subsection 4.3, our algorithm has two parameters, L and R, the number of iterations for the Gauss-Seidel method, and the number of samples for Monte Carlo simulation, respectively. We evaluate the accuracy by changing these parameters.

To evaluate the accuracy, we first compute the *exact* SimRank matrix S by Jeh and Widom's original algorithm [16], and then compute the *mean error* [42] defined as follows:

$$\mathrm{ME} = \frac{1}{n^2} \sum_{i,j} \left| S^L(D)_{ij} - s(i,j) \right|. \qquad (15)$$

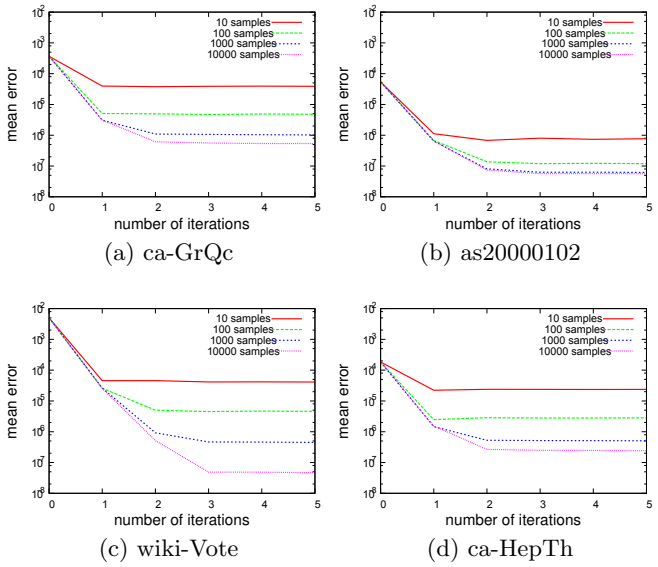
[3]ca-GrQc, as20000102, Wiki-Vote, ca-HepTh, email-Enron, soc-Slashdot0811, soc-Slashdot0902, soc-Epinions1, email-EuAll, web-BerkStan web-Google, web-NotreDame, web-Stanford, and soc-LiveJournal datasets are available at `http://snap.stanford.edu/data/index.html` [3, 21, 22, 23, 24]. dblp-2011, in-2004, indochina-2004, it-2004, twitter-2010, and uk-2007-05 datasets are available at `http://law.di.unimi.it/datasets.php` [5, 6]. flickr dataset is available at `http://socialnetworks.mpi-sws.org/datasets.html` [33]. See the corresponding web pages noted above for detailed information about each of these datasets.

Table 2: Dataset information.

Dataset	n	m
ca-GrQc	5,242	14,496
as20000102	6,474	13,895
Wiki-Vote	7,155	103,689
ca-HepTh	9,877	25,998
email-Enron	36,692	183,831
soc-Epinions1	75,879	508,837
soc-Slashdot0811	77,360	905,468
soc-Slashdot0902	82,168	948,464
email-EuAll	265,214	400,045
web-Stanford	281,903	2,312,497
web-NotreDame	325,728	1,497,134
web-BerkStan	685,230	7,600,505
web-Google	875,713	5,105,049
dblp-2011	933,258	6,707,236
in-2004	1,382,908	17,917,053
flickr	1,715,255	22,613,981
soc-LiveJournal	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
uk-2007-05	105,896,555	3,738,733,648

Since this evaluation is expensive (i.e., it requires SimRank scores for $O(n^2)$ pairs), we used the following smaller datasets: ca-GrQc, as20000102, wiki-Vote, and ca-HepTh. Results are shown in Figure 1, and we summarize our results below.

- For mean error ME $\leq 10^{-5} \sim 10^{-6}$, we need only $R = 100$ samples with $L = 3$ iterations. Note that this is the same accuracy level as [42].
- If we want more accurate SimRank scores, we need much more samples R and little more iterations L. This coincides with the analysis shown in Subsection 4.3 in which we estimated that $L = O(\log(n/\epsilon))$ and $R = O((\log n)/\epsilon^2)$.



(a) ca-GrQc (b) as20000102

(c) wiki-Vote (d) ca-HepTh

Figure 1: Number of iterations L vs. mean error of computed SimRank.

5.2 Efficiency

We next evaluate the efficiency of our algorithm. We first performed preprocessing with parameters $R = 100$ and $L = 3$, respectively. We then performed single-pair, single-source and all-pairs queries for real networks. Results are shown in Table 3; we omitted results of the all-pairs computation for a network larger than in-2004 since runtimes exceeded three days. We summarize our results below.

- For small networks ($n \leq 1,000,000$), only a few minutes of preprocessing time were required; furthermore, answers to single-pair queries were obtained in 100 milliseconds, while answers to single-source queries were obtained in 300 milliseconds. This efficiency is certainly acceptable for online services. We were also able to solve all-pairs query in a few days.
- For large networks, $n \geq 40,000,000$, a few hours of preprocessing time were required; furthermore, answers to single-pair queries were obtained approximately in 10 seconds, while answers to single-source queries were obtained in a half minutes. To the best of our knowledge, this is the first time that such an algorithm is successfully scaled up to such large networks.
- The space complexity is proportional to the number of edges, which enables us to compute SimRank values for large networks.

5.3 Comparisons with existing algorithms

In this subsection, we compare our algorithm with two state-of-the-art algorithms for computing SimRank. We used the same parameters ($R = 100$, $L = 3$) as the above.

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

Yu et al. [42] proposed an efficient all-pairs algorithm; the time complexity of their algorithm is $O(Tnm)$, and the space complexity is $O(n^2)$. They computed SimRank via matrix-based iteration (3) and reduced the space complexity by discarding entries in SimRank matrix that are smaller than a given threshold. We implemented their algorithm and evaluated it in comparison with ours. We used the same parameters presented in [42] that attain the same accuracy level as our algorithm.

Results are shown in Table 3; the omitted results (—) mean that their algorithm failed to allocate memory. From the results, we observe that their algorithm performs a little faster than ours, because the time complexity of their algorithm is $O(Tnm)$, whereas the time complexity of our algorithm is $O(T^2nm)$; however, our algorithm uses much less space. In fact, their algorithm failed for a network with $n \geq 300{,}000$ vertices because of memory allocation. More importantly, their algorithm cannot estimate the memory usage before running the algorithm. 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 [11] proposed an efficient single-pair algorithm that estimates SimRank scores by using first meeting time formula (2) 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. In the query phase, their algorithm computes scores via formula (2); this phase requires $O(TnR')$ time. We implemented their algorithm and evaluated it in comparison with ours.

Table 4: Accuracy of the single-pair algorithm proposed by Fogaras and Rácz [11]; accuracy is shown as mean error.

Dataset	Samples	Accuracy
ca-GrQc	100	1.59×10^{-4}
	1,000	5.87×10^{-5}
	10,000	1.32×10^{-5}
	100,000	6.43×10^{-6}
	(Proposed	4.77×10^{-6})
as20000102	100	2.51×10^{-3}
	1,000	7.87×10^{-4}
	10,000	2.54×10^{-4}
	100,000	8.69×10^{-5}
	(Proposed	1.19×10^{-7})
wiki-Vote	100	1.03×10^{-3}
	1,000	3.57×10^{-4}
	10,000	1.13×10^{-4}
	100,000	3.63×10^{-5}
	(Proposed	2.81×10^{-6})
ca-HepTh	100	1.36×10^{-4}
	1,000	5.58×10^{-5}
	10,000	1.18×10^{-5}
	100,000	6.04×10^{-6}
	(Proposed	4.56×10^{-6})

We first checked the accuracy of their algorithm by computing all-pairs SimRank for the smaller datasets used in Subsection 5.1; results are shown in Table 4. From the table, we observe that in order to obtain the same accuracy as our algorithm, their algorithm requires $R' \geq 100{,}000$ samples, which are much larger than our random samples $R = 100$. This is because their algorithm estimates all $O(n^2)$ entries by Monte Carlo simulation, but our algorithm only estimates $O(n)$ diagonal entries by Monte Carlo simulation.

We then evaluated the efficiency of their algorithm with $R' = 100{,}000$ samples. These results are shown in Table 3. This shows that their algorithm needs much more memory, thus it only works for small networks. This concludes that in order to obtain accurate scores, our algorithm is much more efficient than their algorithm.

6. CONCLUSION

In this paper, we propose a framework for computing SimRank that is based on a linearized SimRank. Our framework consists of two phases, a preprocessing phase and a query phase. Once the preprocessing completes, we can efficiently solve single-pair, single-source, and all-pairs SimRank problems.

Our experiments show that our algorithm is fast and uses relatively low memory; thus it can scale up to very large networks.

For future work, it would be interesting to extend our method to dynamic networks [25]. When a network is modified by adding or removing vertices or edges, we must efficiently update matrix D. Establishing such a method will be noteworthy both in theory and practice.

Table 3: Computational results of our proposed algorithm and existing algorithms; 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 a network larger than in-2004 since runtimes exceeded three days; other omitted results (—) mean that the algorithms failed to allocate memory.

Dataset	Proposed					Yu et al. [42]		Fogaras and Rácz [11]			
	Preproc.	SinglePair	SingleSrc.	AllPairs	Memory	AllPairs	Memory	Preproc.	SinglePair	SingleSrc.	Memory
ca-GrQc	842 ms	0.236 ms	2.080 ms	10.90 s	3 MB	2.97 s	69 MB	64.7 s	87.0 ms	288 ms	23.3 GB
as20000102	96 ms	0.518 ms	0.812 ms	5.26 s	2 MB	0.13 s	8 MB	106 s	112 ms	262 ms	28.1 GB
Wiki-Vote	187 ms	0.613 ms	5.26 ms	37.4 s	6 MB	8.74 s	143 MB	43.4 s	30.4 ms	42.5 ms	24.5 GB
ca-HepTh	698 ms	0.493 ms	3.24 ms	39.0 s	4 MB	23.3 s	316 MB	205 s	61.2 ms	262 ms	44.2 GB
email-Enron	2.75 s	2.56 ms	24.13 ms	885 s	20 MB	302 s	3.47 GB	8055 s	86.9 ms	1.19 ms	162 GB
soc-Epinions1	4.12 s	5.90 ms	74.4 ms	5647 s	31 MB	777 s	6.94 GB	—	—	—	—
soc-Slashdot0811	6.14 s	4.16 ms	20.4 ms	1581 s	47 MB	747 s	7,37 GB	—	—	—	—
soc-Slashdot0902	5.87 s	4.63 ms	21.0 ms	1725 s	49 MB	694 s	7.24 GB	—	—	—	—
email-EuAll	11.3 s	14.5 ms	61.7 ms	4.54 h	57 MB	2.00 h	59.1 GB	—	—	—	—
web-Stanford	21.0 s	9.76 ms	288 ms	22.5 h	132 MB	—	—	—	—	—	—
web-NotreDame	8.07 s	14.7 ms	47.3 ms	4.28 h	107 MB	1.50 h	45.5 GB	—	—	—	—
web-BerkStan	49.6 s	35.7 ms	272 ms	51.7 h	392 MB	—	—	—	—	—	—
web-Google	52.2 s	64.2 ms	234 ms	57.0 h	325 MB	11.1 h	203 GB	—	—	—	—
dblp-2011	104 s	53.6 ms	207 ms	53.7 h	395 MB	3140 s	24.1 GB	—	—	—	—
in-2004	71.7 s	91.1 ms	335 ms	—	843 MB	—	—	—	—	—	—
flickr	160 s	137 ms	424 ms	—	1.11 GB	—	—	—	—	—	—
soc-LiveJournal	819 s	394 ms	1.19 s	—	3.74 GB	—	—	—	—	—	—
indochina-2004	391 s	487 ms	1.73 s	—	8.15 GB	—	—	—	—	—	—
it-2004	2822 s	3.51 s	12.0 s	—	49.2 GB	—	—	—	—	—	—
twitter-2010	14376 s	3.17 s	11.9 s	—	59.4 GB	—	—	—	—	—	—
uk-2007-05	8291 s	9.42 s	32.7 s	—	153 GB	—	—	—	—	—	—

In general, our linearization technique can be applied to recursively defined values with a maximum. For example, consider P-Rank [7, 43], which generalizes SimRank as follows. Let P be a transition matrix of $G^\top$ and let Q be a transition matrix of G. Then P-Rank is defined by

$$S = \Big((1-\lambda)cP^\top SP + \lambda cQ^\top SQ\Big) \vee I.$$

SimRank is the case in which $\lambda = 0$, and rvs-SimRank [43] is the case in which $\lambda = 1$; i.e., SimRank only uses the in-link information, whereas rvs-SimRank only uses out-link information, but P-Rank uses both types of links. We can establish a linear formulation of P-Rank as

$$S = \Big((1-\lambda)cP^\top SP + \lambda cQ^\top SQ\Big) + D$$

with a suitable diagonal matrix D. Therefore we can possibly extend our algorithm to P-Rank by the above formula.

References

[1] K. M. Abadir and J. R. Magnus. *Matrix Algebra*. Cambridge University Press, 2005.

[2] Z. Abbassi and V. S. Mirrokni. A recommender system based on local random walks and spectral methods. In *WebKDD/SNA-KDD*, pages 139–153. Springer, 2007.

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

[4] 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.

[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, 2011.

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

[7] Y. Cai and S. Chakravarthy. Pairwise similarity calculation of information networks. In *DaWaK*, pages 316–329, 2011.

[8] 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, 2009.

[9] Y. Cai, P. Li, H. Liu, J. He, and X. Du. S-SimRank: Combining content and link information to cluster papers effectively and efficiently. In *ADMA*, pages 317–329, 2008.

[10] 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, 2012.

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

[12] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, and M. Onizuka. Efficient search algorithm for SimRank. In *ICDE*, pages 589–600, 2013.

[13] G. H. Golub and C. F. Van Loan. *Matrix Computations*. 3rd eds., Johns Hopkins Studies in the Mathematical Sciences, 2012.

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

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

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

[17] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, pages 271–279, 2003.

[18] X. Jia, Y. Cai, H. Liu, J. He, and X. Du. Calculating similarity efficiently in a small world. In *ADMA*, pages 175–187, 2009.

[19] 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*, pages 61–73, 2010.

[20] M. M. Kessler. Bibliographic coupling extended in time: Ten case histories. *ISR*, 1(4):169–187, 1963.

[21] J. Leskovec, D. P. Huttenlocher, and J. M. Kleinberg. Predicting positive and negative links in online social networks. In *WWW*, pages 641–650, 2010.

[22] J. Leskovec, D. P. Huttenlocher, and J. M. Kleinberg. Signed networks in social media. In *CHI*, pages 1361–1370, 2010.

[23] J. Leskovec, J. M. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*, pages 177–187, 2005.

[24] 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 Math.*, 6(1):29–123, 2009.

[25] 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*, pages 465–476, 2010.

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

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

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

[29] Z. Lin, M. R. Lyu, and I. King. Extending link-based algorithms for similar web pages with neighborhood structure. In *WI*, pages 263–266, 2007.

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

[31] D. Lizorkin, P. Velikhov, M. N. Grinev, and D. Turdakov. Accuracy estimate and optimization techniques for SimRank computation. *VLDBJ*, 19(1):45–66, 2010.

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

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

[34] C. Scheible. Sentiment translation through lexicon induction. In *ACL (Student Research Workshop)*, pages 25–30. 2010.

[35] H. Small. Co-citation in the scientific literature: A new measure of the relationship between two documents. *JASIS*, 24(4):265–269, 1973.

[36] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13(4):354–356, 1969.

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

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

[39] L. Yu, Z. Shu, and X. Yang. SimRate: Improve collaborative recommendation based on rating graph for sparsity. In *ADMA*, pages 167–174, 2010.

[40] W. Yu, X. Lin, and J. Le. Taming computational complexity: Efficient and parallel SimRank optimizations on undirected graphs. In *WAIM*, pages 280–296, 2010.

[41] 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.

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

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

[44] 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.

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

APPENDIX

In this appendix, we provide details of propositions and proofs mentioned in the main body of our paper.

We first introduce a vectorized form of SimRank, which is convenient for analysis. Let $\otimes$ be the Kronecker product of matrices, i.e., for $n \times n$ matrices $A = (A_{ij})$ and B,

$$A \otimes B = \begin{bmatrix} A_{11}B & \cdots & A_{1n}B \\ \vdots & \ddots & \vdots \\ A_{n1}B & \cdots & A_{nn}B \end{bmatrix}.$$

Let "vec" be the vectorization operator, which reshapes an $n \times n$ matrix to an n^2 vector, i.e., $\mathrm{vec}(A)_{n\times i+j} = a_{ij}$. Then we have the following relation, which is well known in linear algebra [1]:

$$\mathrm{vec}(ABC) = (C^\top \otimes A)\mathrm{vec}(B). \tag{16}$$

PROPOSITION 3. *Linearized SimRank operator S^L is a non-singular linear operator.*

PROOF. A linearized SimRank $S^L(\Theta)$ for a matrix Θ is a matrix satisfies relation

$$S^L(\Theta) = cP^\top S^L(\Theta)P + \Theta.$$

By applying the vectorization operator and using (16), we obtain

$$\left(I - cP^\top \otimes P^\top\right)\mathrm{vec}(S^L(\Theta)) = \mathrm{vec}(\Theta). \tag{17}$$

Thus, to prove Proposition 3, we only have to prove that the coefficient matrix $I - cP^\top \otimes P^\top$ is non-singular.

Since $P^\top \otimes P^\top$ is a (left) stochastic matrix, its spectral radius is equal to one. Hence all eigenvalues of $I - cP^\top \otimes P^\top$

are contained in the disk with center 1 and radius c in the complex plane. Therefore $I - cP^\top \otimes P^\top$ does not have a zero eigenvalue, and hence $I - cP^\top \otimes P^\top$ is nonsingular. □

We now prove Proposition 1, which is the basis of our diagonal estimation algorithm.

Proof of Proposition 1. Let us consider linear system (17) and let $Q := P^\top \otimes P^\top$. We partite the system (17) into 2×2 blocks that correspond to the diagonal entries and the others:

$$\begin{bmatrix} I - cQ_{DD} & -cQ_{DO} \\ -cQ_{OD} & I - cQ_{OO} \end{bmatrix} \begin{bmatrix} \vec{1} \\ X \end{bmatrix} = \begin{bmatrix} \mathrm{diag}(D) \\ 0 \end{bmatrix}, \tag{18}$$

where Q_{DD}, Q_{DO}, Q_{OD}, and Q_{OO} are submatrices of Q that denote contributions of diagonals to diagonals, diagonals to off-diagonals, off-diagonals to diagonals, and off-diagonals to off-diagonals, respectively. X is the off-diagonal entries of $S^L(D)$. To prove Proposition 1, we only have to prove that there is a unique diagonal matrix D that satisfies (18).

We observe that the block-diagonal component $I - cQ_{OO}$ of (18) is non-singular, which can be proved similarly as in Proposition 3. Thus, the equation (18) is uniquely solved as

$$(I - cQ_{DD} - c^2 Q_{DO}(I - cQ_{OO})^{-1} Q_{OD})\vec{1} = \mathrm{diag}(D), \tag{19}$$

which is a closed form solution of diagonal correction matrix D. □

Remark 2. *It is hard to compute D via the closed form formula* (19) *because the evaluation of the third term of the left hand side of* (19) *is too expensive.*

On the other hand, we can use (19) *to obtain a reasonable initial solution for Algorithm 4. By using first two terms of* (19)*, we have*

$$\mathrm{diag}(D) \approx \vec{1} - cQ_{DD}\vec{1}, \tag{20}$$

which can be computed in $O(m)$ time.

Our additional experiment shows that the initial solution (20) *gives a slightly better (at most twice) results than the trivial guesses $D = I$ and $D = 1 - c$.*

We give a convergence proof of our diagonal estimation algorithm (Algorithm 4). As mentioned in Subsection 4.1, Algorithm 4 is the Gauss-Seidel method [13] for the linear system

$$\begin{bmatrix} S^L(E^{(1,1)})_{11} & \cdots & S^L(E^{(n,n)})_{11} \\ \vdots & \ddots & \vdots \\ S^L(E^{(1,1)})_{nn} & \cdots & S^L(E^{(n,n)})_{nn} \end{bmatrix} \begin{bmatrix} D_{11} \\ \vdots \\ D_{nn} \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}. \tag{21}$$

Lemma 1. *Consider two independent random walks start from the same vertex i and follow their in-links. Let $p_i(t)$ be the probability that two random walks meet t-th step (at some vertex). Let $\Delta := \max_i\{\sum_{t=1}^\infty c^t p_i(t)\}$. If $\Delta < 1$ then the coefficient matrix of* (21) *is diagonally dominant.*

Proof. By definition, each diagonal entry $S^L(E^{(j,j)})_{jj}$ is greater than or equal to one. For the off diagonals, we have

$$\sum_{i:i\neq j} S^L(E^{(i,i)})_{jj} = \sum_{i:i\neq j}\sum_{t=1}^\infty c^t (P^t e_j)^\top E^{(i,i)}(P^t e_j) \le \sum_{t=1}^\infty c^t (P^t e_j)^\top (P^t e_j) = \sum_{t=1}^\infty c^t p_j(t) \le \Delta.$$

This shows that if $\Delta < 1$ then the matrix is diagonally dominant. □

Corollary 1. *If a graph G satisfies the condition of Lemma 1, Algorithm 4 converges with convergence rate $O(\Delta^l)$.*

Proof. This follows the standard theory of the Gauss-Seidel method [13]. □

Remark 3. *Let us observe that, in practice, the assumption $\Delta < 1$ is not an issue. For a network of average degree d, the probability $p_i(t)$ is expected to $1/d^t$. Therefore $\Delta = \sum_t c^t p_i(t) \simeq (c/d)/(1 - (c/d)) \le 1/(d-1) < 1$. This implies that Algorithm 4 converges quickly when the average degree is large.*

We now prove Proposition 2. We use the following lemma.

Lemma 2. *Let $k_1^{(t)}, \ldots, k_R^{(t)}$ be positions of t-th step of independent random walks that start from a vertex k and follow ln-links. Let $X_k^{(t)} := (1/R)\sum_{r=1}^R e_{k_r^{(t)}}$. Then for all $l = 1, \ldots, n$,*

$$P\left\{\left|e_l^\top \left(X_k^{(t)} - P^t e_k\right)\right| \ge \epsilon\right\} \le 2\exp\left(-2R\epsilon^2\right).$$

Proof. Since $\mathbf{E}[e_{k_r^{(t)}}] = P^t e_k$, this is a direct application of the Hoeffding's inequality. □

Proof of Proposition 2. Since $p_{ki}^{(t)}$ defined by (13) is represented by $p_{ki}^{(t)} = e_i^\top X_k^{(t)}$. Thus we have

$$P\left\{\|P^t e_k - p_k^{(t)}\| > \epsilon\right\} \le nP\left\{\left|e_i^\top P^t e_k - p_{ki}^{(t)}\right| > \epsilon\right\} \le 2n\exp\left(-2R\epsilon^2\right).$$

□

Proposition 4. *Let $D = \mathrm{diag}(D_{11}, \ldots, D_{nn})$ and $\tilde{D} = \mathrm{diag}(\tilde{D}_{11}, \ldots, \tilde{D}_{nn})$ be diagonal matrices. If they satisfy*

$$\sup_k |D_{kk} - \tilde{D}_{kk}| \le \epsilon$$

then

$$\sup_{i,j} |S^L(D)_{ij} - S^L(\tilde{D})_{ij}| \le \frac{\epsilon}{1-c}.$$

Proof. Let $\Delta := D - \tilde{D}$. Since S^L is linear, we have $S^L(\Delta) = S^L(D) - S^L(\tilde{D})$. Consider

$$S^L(\Delta) = cP^\top S^L(\Delta)P + \Delta.$$

By applying e_i and e_j, we have

$$S^L(\Delta)_{ij} = c(Pe_i)^\top S^L(\Delta)(Pe_j) + \Delta_{ij} \le c\sup_{i',j'} S^L(\Delta)_{i'j'} + \epsilon.$$

Here, we used $p^\top A q \le \sup_{ij} A_{ij}$ for any stochastic vectors p and q. Therefore

$$(1-c)\sup_{i,j} S^L(\Delta)_{ij} \le \epsilon.$$

By the same argument, we have

$$(1-c)\inf_{i,j} S^L(\Delta)_{ij} \ge -\epsilon.$$

By combining them, we obtain the proposition. □

The above proposition shows that if diagonal correction matrix D is accurately estimated, all entries of SimRank matrix S is accurately computed.