Fast Computation of SimRank for Static and Dynamic Information Networks

Cuiping Li
Renmin University of China
licuiping@ruc.edu.cn

Jiawei Han
UIUC, IL, USA
hanj@cs.uiuc.edu

Guoming He
Renmin University of China
Guoming@ruc.edu.cn

Xin Jin
UIUC, IL, USA
xinjin3@illinois.edu

Yizhou Sun, Yintao Yu
UIUC, IL, USA
sun22@illinois.edu,
yintao@uiuc.edu

Tianyi Wu
UIUC, IL, USA
twu5@uiuc.edu

ABSTRACT

Information networks are ubiquitous in many applications and analysis on such networks has attracted significant attention in the academic communities. One of the most important aspects of information network analysis is to measure similarity between nodes in a network. SimRank is a simple and influential measure of this kind, based on a solid theoretical "random surfer" model. Existing work computes SimRank similarity scores in an iterative mode. We argue that the iterative method can be infeasible and inefficient when, as in many real-world scenarios, the networks change dynamically and frequently. We envision non-iterative method to bridge the gap. It allows users not only to update the similarity scores incrementally, but also to derive similarity scores for an arbitrary subset of nodes. To enable the non-iterative computation, we propose to rewrite the SimRank equation into a non-iterative form by using the Kronecker product and vectorization operators. Based on this, we develop a family of novel approximate SimRank computation algorithms for static and dynamic information networks, and give their corresponding theoretical justification and analysis. The non-iterative method supports efficient processing of various node analysis including *similarity tracking* and *centrality tracking* on evolving information networks. The effectiveness and efficiency of our proposed methods are evaluated on synthetic and real data sets.

Keywords

Similarity Measure, SimRank, Information Network, Graph

1. INTRODUCTION

In many applications, there exist a large number of individual agents or components interacting with a specific set of components, forming large, interconnected, and sophisticated networks. We call such interconnected networks as *information networks*, with examples including the Internet, research collaboration networks, public health systems, biological networks, and so on. Clearly, information networks are ubiquitous and form a critical part of modern information infrastructures.

Information network analysis has attracted a lot of attention from epidemiologists, sociologists, biologists, and more recently also computer scientists. In recent years, various approaches have been proposed to deal with a variety of information network related research problems, including power laws discovery [1], frequent pattern mining [2, 3], clustering and community identification [4, 5], and node ranking [6, 7].

One of the most important aspects of information network analysis is to measure similarity between nodes in a network. There are many situations in which it would be useful to be able to answer questions such as "How similar are these two nodes?" or "Which other nodes are most similar to this one?". Motivated by this need, a great number of similarity measures are reported in the literature [8, 9, 10, 11, 12]. Most of them fall into one of the following two categories:

1. text- or content-based similarity measures: treat each object as a bag of items or as a vector of word weights [8];
2. link- or structure-based similarity measures: consider object-to-object relationships expressed in terms of links [9, 10, 11, 12];

Based on the evaluation of [13], link-based measures produce better correlation with human judgements compared with text-based measures. From this perspective, it is reasonable to assume that analyzing information network based on structure similarity is essential for many applications and worth thoroughly exploring.

Among almost all existing link-based similarity measures, SimRank [9] is an influential one. Informally speaking, SimRank similarity score relates to the expected distance for two random surfers to first meet at the same node. In contrast with other measures, SimRank does not suffer from any field restrictions and can be applied to any domain with object-to-object relationships. Furthermore, SimRank takes into account not only direct connections among nodes but also indirect connections.

SimRank similarity score plays a significant role in the analysis of information networks and a variety of other applications such as neighborhood search, centrality analysis, link prediction, graph clustering and multimedia (image, video clip, or audio song) captioning. For example, a general problem in image captioning is to automatically assign keywords to an image. In this case, a graph is generated from extracted images regions and terms according to structural characteristics. Then, the graph is used to estimate the affinity of each term to the uncaptioned image, and the top-k affinitive terms are selected as the caption of the image. In this context, SimRank provides a good way for measuring node similarities.

EDBT 2010, March 22–26, 2010, Lausanne, Switzerland.

Unfortunately, the main drawback of SimRank is its computation complexity. In the spirit of PageRank [6], SimRank computes similarity of two objects through an iterative mode. Despite the importance of the theoretical guarantee on the convergence, the cost for iteratively computing SimRank similarity scores can be very high in practice. In [14], Dmitry, Pavel, Maxim and Denis run the original iterative SimRank on a 2.1GHz Intel Pentium processor with 1Gb RAM for a scale-free generated graph which consists of 10000 nodes. It took 46 hours and 5 minutes for the algorithm to iterate 5 times to compute all node similarities.

In order to optimize the computation of SimRank, a few techniques have been proposed [15, 14]. However, these approaches are all under the same iterative computation framework, which suffers from the following limitations. First, the iterative algorithm cannot deal effectively with the dynamic behavior of the network; when the network is changed, all existing similarity measures will have to be recomputed, i.e., they cannot be updated incrementally. Second, the iterative algorithm has the global nature: the whole similarity scores will be computed even only a portion of them is required, which wastes a lot of time and space. Therefore, these optimized solutions are particularly inefficient in practice.

Accordingly, in this paper we propose a rather different optimization approach for SimRank to address the above challenges. Our key observation is that the iterative computation formula of SimRank resembles the well-known *Sylvester equation* [16]. Based on this, we propose a novel technique that re-writes the SimRank equation into a non-iterative form by using the Kronecker product and vectorization operators. Equipped with the powerful low-rank approximation technology, the non-iterative computation framework enables us: (1) to derive similarity scores for an arbitrary subset of nodes in a network on-the-fly; for instance, if only the similarity score between two nodes i and j is needed, it can be computed individually in linear time without having to compute the whole similarity matrix; and (2) to update SimRank scores incrementally; when the network changs over time, we can provide any-time query answer by updating SimRank scores incrementally. Specifically, this paper has made the following contributions.

1. We propose a novel technique that re-writes the SimRank equation into a non-iterative form by using the Kronecker product and vectorization operators, which lays the foundation for SimRank's optimization as well as incremental update.
2. We develop a family of novel approximate SimRank computation algorithms for static and dynamic information networks, and give formal proofs, complexity analysis, and error bounds, showing our methods are provably efficient, with small loss of accuracy.
3. Based on this efficient computation methods, we develop two algorithms S_Track and C_Track for performing node similarity and centrality tracking analysis on evolving information networks respectively.
4. Extensive experimental studies on synthetic and real data sets to verify the effectiveness and efficiency of the proposed methods.

The rest of this paper is organized as follows. Section 2 gives the background information of our study. Section 3 introduces our techniques for non-iterative SimRank computation. Section 4 presents two approximate SimRank computation algorithms for static information network while Section 5 gives one incremental update algorithm for dynamic network. Section 6 investigates the applications of these algorithms in performing node similarity and centrality tracking analysis on evolving information networks. A performance analysis of our methods is presented in Section 7. We discuss related work in Section 8 and conclude the study in Section 9.2.

2. PRELIMINARIES

In this section, we provide the necessary background for the subsequent discussions. We first present some notations and assumptions that are adopted in this paper in Section 2.1, and then give a brief review of SimRank in Section 2.2.

2.1 Notations and Assumptions

Symbol	Definition and Description
$\mathbf{A}$,$\mathbf{B}$, ...	matrices (bold upper case)
$\mathbf{A}(i, j)$	the element at i^{th} row and j^{th} column of matrix $\mathbf{A}$
$\mathbf{A}(i, :)$	the i^{th} row of matrix $\mathbf{A}$
$\mathbf{A}(:, j)$	the j^{th} column of matrix $\mathbf{A}$
$\tilde{\mathbf{A}}$	transpose of matrix $\mathbf{A}$
$\mathcal{G}$,$\mathcal{V}$, ...	sets (calligraphic)
n	the number of nodes in the network
k	the rank of a matrix
c	the decay factor for SimRank
m	the number of changed node in the network
N	the number of top objects that have high similarity or centrality scores

Table 1: Symbols

Table 1 lists the main symbols we use throughout the paper. Without loss of generality, we model objects and relationships in an information network as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where nodes in $\mathcal{V}$ represent objects of the domain and edges in $\mathcal{E}$ represent relationships between objects. For a node v in a graph, $\mathcal{I}(v)$ and $\mathcal{O}(v)$ denote the set of in-neighbors and out-neighbors of v, respectively.

Given a graph $\mathcal{G}$, $\mathbf{M}$ denotes the adjacency matrix of $\mathcal{G}$ and $\tilde{\mathbf{M}}$ the transpose of $\mathbf{M}$. Similar to Matlab, we use $\mathbf{M}(i, j)$ to represent the element at the i^{th} row and j^{th} column of the matrix $\mathbf{M}$, $\mathbf{M}(i, :)$ the i^{th} row of $\mathbf{M}$, and so on. Given two nodes i and j, we use $\mathbf{S}(i, j)$ to denote the similarity between nodes i and j. The whole similarity matrix of $\mathcal{G}$ is denoted by $\mathbf{S}$.

In rapidly changing environments such as World Wide Web, the graph is frequently updated. At each time step t, we use $\mathbf{M}^t$ to denote the adjacency matrix at time t. We will not use a t subscript on these variables except where it is needed for clarity. We assume that the number of network nodes is fixed; if not, we can reserve rows/columns with zero elements as necessary. In the following discussions, we focus on the undirected graphs. Our approach can be easily applied to directed graphs.

2.2 SimRank Overview

In this section, we will give a brief review of SimRank. Let $\mathbf{S}(a, b) \in [0, 1]$ denote the similarity between two objects a and b, the iterative similarity computation equation of SimRank is as follows:

$$\mathbf{S}(a,b) = \begin{cases} \frac{c}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{i=1}^{|\mathcal{I}(a)|} \sum_{j=1}^{|\mathcal{I}(b)|} \mathbf{S}(\mathcal{I}_i(a), \mathcal{I}_j(b)), & a \neq b \\ 1, & a = b \end{cases} \quad (1)$$

where c is the decay factor for SimRank (a constant between 0 and 1), $|\mathcal{I}(a)|$ or $|\mathcal{I}(b)|$ is the number of nodes in $\mathcal{I}(a)$ or $\mathcal{I}(b)$. Individual member of $\mathcal{I}(a)$ or $\mathcal{I}(b)$ is referred to as $\mathcal{I}_i(a)$, $1 \leq i \leq |\mathcal{I}(a)|$, or $\mathcal{I}_j(b)$, $1 \leq j \leq |\mathcal{I}(b)|$. As the base case, any object is considered maximally similar to itself, i.e., $\mathbf{S}(a, a) = 1$.

For preventing division by zero in the general formula (1) in case of $\mathcal{I}(a)$ or $\mathcal{I}(b)$ being an empty set, $\mathbf{S}(a,b)$ is specially defined as zero for $\mathcal{I}(a) = \varnothing$ or $\mathcal{I}(b) = \varnothing$.

3. NON-ITERATIVE SIMRANK COMPUTATION FRAMEWORK

Existing methods compute SimRank measure in an iterative manner; that is, SimRank scores are propagated through the graph in multiple iterations until convergence. As discussed earlier, this iterative computation framework suffers from some limitations. In this section, we introduce a non-iterative SimRank computation framework which lays the foundation for SimRank's optimization as well as incremental update.

3.1 key observation

To make the paper self-contained, we first briefly introduce two useful matrix operators. Interested readers can refer to [17] for more details.

DEFINITION 1 (KRONECKER PRODUCT). *Let $\boldsymbol{A} \in \Re^{s\times t}$, $\boldsymbol{B} \in \Re^{p\times q}$. Then the Kronecker product of $\boldsymbol{A}$ and $\boldsymbol{B}$ is defined as the matrix*

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{11}\boldsymbol{B} & \dots & a_{1t}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{s1}\boldsymbol{B} & \dots & a_{st}\boldsymbol{B} \end{bmatrix}.$$

Obviously, the Kronecker product of two matrices $\mathbf{A}$ and $\mathbf{B}$ is a $sp \times tq$ matrix.

DEFINITION 2 (VEC-OPERATOR). *Let $c_i \in \Re^s$ denote the columns of $\boldsymbol{C} \in \Re^{s\times t}$ so that $\boldsymbol{C} = [c_1, \dots, c_t]$. Then $vec(\boldsymbol{C})$ is defined to be the st-vector formed by stacking the columns of $\boldsymbol{C}$ on top of one another, i.e.,*

$$vec(\boldsymbol{C}) = \begin{bmatrix} c_1 \\ \vdots \\ c_t \end{bmatrix} \in \Re^{st}.$$

The Kronecker product and the vec operator have many useful properties. The following theorems are worth noting for the purpose of our further discussion:

THEOREM 1. *For any three matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ for which the matrix product $\boldsymbol{ABC}$ is defined,*

$$vec(\boldsymbol{ABC}) = (\tilde{\boldsymbol{C}} \otimes \boldsymbol{A}) vec(\boldsymbol{B}).$$

THEOREM 2. *Let $\boldsymbol{A} \in \Re^{m\times n}$, $\boldsymbol{B} \in \Re^{r\times s}$, $\boldsymbol{C} \in \Re^{n\times p}$, and $\boldsymbol{D} \in \Re^{s\times t}$. Then*

$$(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{C} \otimes \boldsymbol{D}) = \boldsymbol{AC} \otimes \boldsymbol{BD}.$$

Let $\mathbf{W}$ be the column-normalized matrix of $\mathbf{M}$. When the iteration number is sufficiently large, the iterative SimRank similarity computation equation (1) can be written as the following matrix form:

$$\mathbf{S} = c\tilde{\mathbf{W}}\mathbf{S}\mathbf{W} + (1-c)\mathbf{I} \tag{2}$$

where $\mathbf{I}$ is an identity matrix.

Our key observation is that Equation (2) is in the form of the well-known *Sylvester Equation* [16]. Specifically, by multiplying $(c\tilde{\mathbf{W}})^{-1}$ to both sides of Equation (2), we get,

$$(c\tilde{\mathbf{W}})^{-1}\mathbf{S} = \mathbf{S}\mathbf{W} + (c\tilde{\mathbf{W}})^{-1}(1-c)\mathbf{I}$$

Let $\mathbf{A} = (c\tilde{\mathbf{W}})^{-1}$, $\mathbf{B} = -\mathbf{W}$, $\mathbf{C} = (c\tilde{\mathbf{W}})^{-1}(1-c)\mathbf{I}$, Equation (2) thus takes the form: **AS+SB = C**, which fits the *Sylvester Equation*. $\mathbf{S}$ is the solution to this Equation if $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are known. This motivates us to find a different solution for SimRank.

3.2 SimRank Equation Re-write

Next, we introduce how to re-write Equation (2) into a non-iterative form.

After applying *vec* operator on Equation (2), we obtain

$$vec(\mathbf{S}) = c(vec(\tilde{\mathbf{W}}\mathbf{S}\mathbf{W})) + (1-c)vec(\mathbf{I})$$

According to Theorem 1,

$$vec(\mathbf{S}) = c(\tilde{\mathbf{W}} \otimes \tilde{\mathbf{W}})vec(\mathbf{S}) + (1-c)vec(\mathbf{I}) \tag{3}$$

Given a graph $\mathcal{G}$ of size n, intuitively, $\tilde{\mathbf{W}} \otimes \tilde{\mathbf{W}}$ of Equation (3) represents the normalized adjacent matrix of the derived graph $\mathcal{G}^2 = (\mathcal{V}^2, \mathcal{E}^2)$. The i^{th} element of $vec(\mathbf{S})$ represents the expected-f meeting distance in $\mathcal{G}^2$ from node (x, y) (x = i mod n, y = i/n + 1) to any singleton node $(z, z) \in \mathcal{V}^2$. In the original graph $\mathcal{G}$, it can be thought of as that one surfer starts from node x while the other starts from node y, and finally they meet at the node z. Thus, Equation (3) is exactly the same as the random surfer-pairs model discussed in [9]. It also strongly resembles the random walk with restart model used in [18, 19, 20]. The difference is Equation (3) computes all similarities for all node pairs while the random walk with restart model computes the similarities from one fixed node to all other nodes. If we evenly cut the $vec(\mathbf{I})$ as well as $vec(\mathbf{S})$ of Equation (3) into n segments , then each of them is actually the same as what the random walk with restart model uses and generates for the graph $\mathcal{G}^2$.

By further re-writing, now, the problem is reduced to compute

$$vec(\mathbf{S}) = (1-c)(\mathbf{I} - c(\tilde{\mathbf{W}} \otimes \tilde{\mathbf{W}}))^{-1}vec(\mathbf{I}) \tag{4}$$

Let $\mathbf{L} = \mathbf{I} - c(\tilde{\mathbf{W}} \otimes \tilde{\mathbf{W}})$. From Equation (4), we can see that $\mathbf{L}^{-1}$ contains all information desired to compute the similarity matrix $\mathbf{S}$. In fact, by rewriting the original iterative definition of SimRank into the form of Equation (4), we can derive $\mathbf{S}$ by computing the RHS of Equation (4) without multiple iterations. As will be discussed shortly, such rewriting enables us to develop algorithms to compute and update SimRank scores efficiently.

A slight technicality here is that the similarity of a data object to itself derived by Equation (4) may not be equal to 1 now, because we cannot set the diagonal values of $\mathbf{S}$ to 1 as SimRank does at each iteration. However, this is a trivial problem as it affects only the absolute similarity value but not the relative similarity ranking. For instance, in this setting, any object is still maximally similar to itself. We will further discuss this in Section 7.

4. FAST STATIC SIMRANK COMPUTATION

In contrast with existing methods, Equation (4) provides a completely different approach for SimRank computation. However, computing $\mathbf{L}^{-1}$ directly is infeasible when the data set is large since it requires cubic computation time.

4.1 An Approximation Algorithm with Quality Assurance

Algorithm 1 Non-Iterative SimRank Algorithm (N_Sim)

INPUT: the normalized adjacency matrix $\tilde{\mathbf{W}}$
OUTPUT: The similarity matrix $\mathbf{S}$
ALGORITHM:
01: Do low-rank approximation for $\tilde{\mathbf{W}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}$
02: $\mathbf{K}_u = \mathbf{U} \otimes \mathbf{U}$, $\mathbf{K}_\Sigma = \mathbf{\Sigma} \otimes \mathbf{\Sigma}$, $\mathbf{K}_v = \mathbf{V} \otimes \mathbf{V}$
03: $\mathbf{K}_{vu} = \mathbf{K}_v\mathbf{K}_u$
04: Compute the core matrix $\mathbf{\Lambda} = (\mathbf{K}_\Sigma^{-1} - c\mathbf{K}_{vu})^{-1}$
05: Compute the right vector $\mathbf{V}_r = \mathbf{K}_v vec(\mathbf{I})$
06: $\mathbf{P} = \mathbf{K}_u\mathbf{\Lambda}$
07: $vec(\mathbf{S}) = (1-c)(vec(\mathbf{I}) + c\mathbf{P}\mathbf{V}_r)$

Because linear correlations commonly exist in many real graphs, we resort to low-rank approximation to efficiently approximate $\mathbf{L}^{-1}$ (recall that a high-dimensional matrix can be well approximated by the product of several lower dimensional matrices).

Formally, a rank-k approximation of matrix $\mathbf{A}$ is a matrix $\bar{\mathbf{A}}$ where $\bar{\mathbf{A}}$ is of rank k and $\|\mathbf{A} - \bar{\mathbf{A}}\|$ is small. The low-rank approximation is usually presented in a factorized form, e.g., $\bar{\mathbf{A}} = \mathbf{LMR}$ where $\mathbf{L}$, $\mathbf{M}$, and $\mathbf{R}$ are of rank-k. There are many different low-rank approximations in the literature, for example, in SVD [21], $\mathbf{L}$ and $\mathbf{R}$ are orthogonal matrices whose columns/rows are singular vectors and $\mathbf{M}$ is a diagonal matrix whose diagonal entries are singular values. Since among all the possible rank-k approximations, SVD gives the best approximation in terms of squared error, in this paper, we adopt it as our low-rank approximation method. For the symmetric matrix, we will use eigen-value decomposition instead to save storage cost.

Algorithm 1 shows the pseudo-code of our non-iterative SimRank algorithm for a static graph. On the matter of its correctness, we have the following theorem:

THEOREM 3. *If $\boldsymbol{U\Sigma V}$ is a full decomposition for $\tilde{\boldsymbol{W}}$, Algorithm 1 outputs exactly the same result as Equation (4) does.*

proof: *Based on theorem 2, we have*

$$\tilde{\boldsymbol{W}} \otimes \tilde{\boldsymbol{W}} = (\boldsymbol{U} \otimes \boldsymbol{U})(\mathbf{\Sigma} \otimes \mathbf{\Sigma})(\boldsymbol{V} \otimes \boldsymbol{V})$$

Let

$$\mathbf{\Lambda} = ((\mathbf{\Sigma} \otimes \Sigma)^{-1} - c(\boldsymbol{V} \otimes \boldsymbol{V})(\boldsymbol{U} \otimes \boldsymbol{U}))^{-1} \quad (5)$$

Based on the Sherman-Morrison Lemma [22]:

$$(I - c(\tilde{\boldsymbol{W}} \otimes \tilde{\boldsymbol{W}}))^{-1} = \boldsymbol{I} + c(\boldsymbol{U} \otimes \boldsymbol{U})\mathbf{\Lambda}(\boldsymbol{V} \otimes \boldsymbol{V})$$

By Equation 4, we have

$$vec(\boldsymbol{S}) = (1-c)(\boldsymbol{I} + c(\boldsymbol{U} \otimes \boldsymbol{U})\mathbf{\Lambda}(\boldsymbol{V} \otimes \boldsymbol{V}))vec(\boldsymbol{I}) \quad (6)$$

Error Bound. Developing an error bound for the general case of Algorithm 1 is difficult. However, for the symmetric matrix, we have the following theorem:

THEOREM 4. *Assume $\boldsymbol{S}$ is the similarity matrix computed by Equation (4), and $\hat{\boldsymbol{S}}$ is the similarity matrix computed by Algorithm 1 which takes eigen-value decomposition as low-rank approximation, then*

$$\|\boldsymbol{S} - \hat{\boldsymbol{S}}\|_1 = c(1-c)\sum_{i=k+1}^{n} \frac{\lambda_1\lambda_i}{1-c\lambda_1\lambda_i}$$

in which λ_i is the i^{th} largest eigen-value of $\tilde{\boldsymbol{W}}$.

proof: *first, do a full eigen-value decomposition for $\tilde{\boldsymbol{W}}$.*

$$\tilde{\boldsymbol{W}} = \boldsymbol{U\Sigma}\tilde{\boldsymbol{U}}$$

in which, $\mathbf{\Sigma}$ = diag(λ_1, ..., λ_n) and λ_i is the i^{th} largest eigen-value of $\tilde{\boldsymbol{W}}$. We have,

$$\begin{aligned}(\mathbf{\Sigma} \otimes \mathbf{\Sigma})^{-1} &= (diag(\lambda_1\lambda_1, \ldots, \lambda_1\lambda_n, \ldots, \lambda_2\lambda_1, \ldots, \lambda_n\lambda_n))^{-1} \\ &= diag(\tfrac{1}{\lambda_1\lambda_1}, \ldots, \tfrac{1}{\lambda_n\lambda_n})\end{aligned}$$

By Equation (5), we have,

$$\mathbf{\Lambda} = ((\mathbf{\Sigma} \otimes \mathbf{\Sigma})^{-1} - c(\tilde{\boldsymbol{U}} \otimes \tilde{\boldsymbol{U}})(\boldsymbol{U} \otimes \boldsymbol{U}))^{-1}$$

Since $\tilde{\boldsymbol{U}} \otimes \tilde{\boldsymbol{U}} = (\boldsymbol{U} \otimes \tilde{\boldsymbol{U}})$ (property of Kronecker Product), and $(\boldsymbol{U} \otimes \tilde{\boldsymbol{U}})(\boldsymbol{U} \otimes \boldsymbol{U}) = I$,

$$\begin{aligned}\mathbf{\Lambda} &= ((\mathbf{\Sigma} \otimes \mathbf{\Sigma})^{-1} - c(\boldsymbol{U} \otimes \tilde{\boldsymbol{U}})(\boldsymbol{U} \otimes \boldsymbol{U}))^{-1} \\ &= ((\mathbf{\Sigma} \otimes \mathbf{\Sigma})^{-1} - cI)^{-1} \\ &= (diag(\tfrac{1-c\lambda_1\lambda_1}{\lambda_1\lambda_1}, \ldots, \tfrac{1-c\lambda_n\lambda_n}{\lambda_n\lambda_n}))^{-1} \\ &= diag(\tfrac{\lambda_1\lambda_1}{1-c\lambda_1\lambda_1}, \ldots, \tfrac{\lambda_n\lambda_n}{1-c\lambda_n\lambda_n})\end{aligned}$$

By Equations (6), we have:

$$vec(\boldsymbol{S}) = (1-c)(\boldsymbol{I} + c \cdot diag(\tfrac{\lambda_1\lambda_1}{1-c\lambda_1\lambda_1}, \ldots, \tfrac{\lambda_n\lambda_n}{1-c\lambda_n\lambda_n}))vec(\boldsymbol{I})$$

$$vec(\hat{\boldsymbol{S}}) = (1-c)(\boldsymbol{I} + c \cdot diag(\tfrac{\lambda_1\lambda_1}{1-c\lambda_1\lambda_1}, \ldots, \tfrac{\lambda_k\lambda_k}{1-c\lambda_k\lambda_k}))vec(\boldsymbol{I})$$

Thus, we have

$$\begin{aligned}\|\boldsymbol{S} - \hat{\boldsymbol{S}}\|_1 &= c(1-c)\|diag(\tfrac{\lambda_1\lambda_{k+1}}{1-c\lambda_1\lambda_{k+1}}, \ldots, \tfrac{\lambda_1\lambda_n}{1-c\lambda_1\lambda_n})\|_1 \\ &= c(1-c)\sum_{i=k+1}^{n} \tfrac{\lambda_1\lambda_i}{1-c\lambda_1\lambda_i}\end{aligned}$$

The error bound, $\|\mathbf{S} - \hat{\mathbf{S}}\|_1$, is very small in practice, since it is monotonically decreasing w.r.t. λ_i, which is small when $i > k$.

4.2 Further Efficiency Improvement

One drawback of Algorithm 1 is, the similarity matrix is computed for the *entire* graph in a holistic manner even if the similarities for a small subset of nodes are required. The size of network can cause computation to take very long time to complete. This delay is unacceptable in most real environments, as it severely limits productivity. The usual requirement for the computation time is a few seconds or a few minutes at the most.

There are many ways to achieve such performance goals. A commonly used technique is to do some pre-computation and then materialize the result. Picking the right information to materialize is an important task, since by materializing some information we may be able to get the similarities quickly. A nice observation to Algorithm 1 is that, matrices $\mathbf{K}_u$, $\mathbf{\Lambda}$, $\mathbf{K}_v$, and $\mathbf{V}_r$ carry all information desired to compute all possible similarity scores. If they can be pre-computed and stored, we can get the similarity on-the-fly for any query node pair.

Here, we propose another version of Algorithm 1, which consists of two phases: pre-computation phase and query phase. The pseudo-code for these two phases is shown in Algorithm 2. We can see that, having the pre-computed matrices $\mathbf{K}_u$, $\mathbf{\Lambda}$, $\mathbf{K}_v$, and $\mathbf{V}_r$, we only need to do one vector-matrix and one vector-vector multiplications in query phase to get the proper answer. Comparing to the pre-computation time, the query time is much less. So our algorithm can give the quick answer for any query nodes.

Algorithm 2 Improved version of N_Sim (NI_Sim)

INPUT: The normalized adjacency matrix $\tilde{\mathbf{W}}$
The query node pair i and j
OUTPUT: The similarity between nodes i and j
ALGORITHM:
I. Pre-computation
01-05: The same as Algorithm 1
06: Store the matrices: $\mathbf{K}_u$, $\mathbf{\Lambda}$, $\mathbf{K}_v$, $\mathbf{V}_r$
II. Query Processing
07: Compute the left vector $\mathbf{V}_l = \mathbf{K}_u((i-1)n+j,:)\mathbf{\Lambda}$
08: $\mathbf{S}(i,j) = (1-c)(\mathbf{I}(i,j) + c\mathbf{V}_l\mathbf{V}_r)$

4.3 Cost Analysis

In this section, we make a detailed analysis in terms of pre-computational, query, and storage cost for Algorithm 2.

Pre-computational Cost: In our implementation, we adopt the Krylov-Schur SVD algorithm to calculate a truncated SVD with the k largest singular values of the matrix $\tilde{\mathbf{W}}$ [23]. The Krylov-Schur SVD algorithm is an iterative method. Before iterations, the algorithm reduces the matrix to r-dimensional bidiagonal form, in which r is roughly twice of k, ie. $r \approx 2k$. In each iteration, the algorithm calculates the SVD of the r-dimensional bidiagonal matrix, picks up k singular values with desired tolerance and extends the picked-up matrix back into r-dimensional bidiagonal form. The algorithm stops until the total tolerance of the picked-up values is small enough. In this algorithm, the bidiagonal reduction step costs $O(rn^2)$ in time complexity; in each iteration, it only takes $O(r^3)$ time to calculate the SVD of the r-dimensional bidiagonal matrix and $O((r-k)n^2)$ time to extend the picked-up matrix. So after applying the Krylov-Schur SVD algorithm, the pre-computational cost is dominated by: 1) the multiplication of K_v and K_u; 2) the computation of the core matrix $\mathbf{\Lambda}$. They all take O(k^4n^2) ($k \ll n$). As an off-line procedure, such complexity should be acceptable in most cases. Moreover, we only need to do such pre-computation one time. Once it is done, the pre-computation result can be incrementally updated later when the graph is changed. Details will be explained in the next section.

Query Cost: It is not hard to see that, at the query stage, we only need to do: (1) multiply one vector and one matrix($O(1\times k^2 \times k^2)$); and (2) multiply two vectors ($O(k^2)$). Therefore, the complexity is $O(k^4)$. Since $k \ll n$, the algorithm is capable of meeting the near real-time response requirement.

Storage Cost: In terms of storage cost, we have to store one small $k^2 \times k^2$ core matrix ($\mathbf{\Lambda}$), one $n^2 \times k^2$ matrix ($\mathbf{K}_u$), one $k^2 \times n^2$ matrix ($\mathbf{K}_v$), and one small $k^2 \times 1$ matrix ($\mathbf{V}_r$). We can further save the storage cost as shown in the following:

- We observe that many elements in $\mathbf{K}_u$ and $\mathbf{K}_v$ are near zero. We introduce a threshold T and set those elements smaller than T to be zero and then store the matrix as sparse format. Experiments show that this step can significantly reduce the storage cost while almost not affecting the approximation accuracy[1].
- For the symmetric matrix, we can use eigen-value decomposition when computing the low-rank approximation. In this case, $\mathbf{K}_u = \tilde{\mathbf{K}}_v$, and 50% storage cost can be saved.
- Other low-rank decomposition methods such as CUR [24] or CMD [25] can be used. Since these methods can generate a sparse representation of the original matrix, significant savings in space can be achieved[2].

[1]In [20], the authors suggest similar strategies to save storage cost.

5. INCREMENTAL DYNAMIC SIMRANK UPDATE

In this section, we propose our similarity computation algorithm for dynamic, time-evolving graphs. Our goal is to obtain the similarity score between any two nodes at each time step t efficiently.

5.1 Principle and Algorithm

Obviously, Algorithm 2 can be called at each time step t to compute similarities between nodes. However, in a dynamic setting, the adjacency matrix changes over time, which means the pre-computed matrices $\mathbf{\Lambda}$, $\mathbf{K}_u$, and $\mathbf{K}_v$ are no longer applicable and we will have to re-compute them from the scratch. In other words, steps 1-6 of Algorithm 2 themselves become a part of on-line query processing. Since the worst case complexity of pre-computation is $O(k^4n^2)$, such performance is undesirable when on-line response is crucial and the dataset is large.

Thus, given a difference matrix $\triangle\tilde{\mathbf{W}}^t = \tilde{\mathbf{W}}^t - \tilde{\mathbf{W}}^{t-1}$, our goal is to efficiently update $\mathbf{\Lambda}^t$, $\mathbf{K}_u^t$, and $\mathbf{K}_v^t$ at time step t, based on $\mathbf{\Lambda}^{t-1}$, $\mathbf{K}_u^{t-1}$, $\mathbf{K}_v^{t-1}$, and $\triangle\tilde{\mathbf{W}}^t$. Intuitively, if we can incrementally update low-rank approximation matrices $\mathbf{U}^t$, $\mathbf{\Sigma}^t$, and $\mathbf{V}^t$, we should be able to update $\mathbf{\Lambda}^t$, $\mathbf{K}_u^t$, and $\mathbf{K}_v^t$.

Suppose there are a total of m ($m \ll n$) nodes which had been changed at time step t. Motivated by [26], we first decompose the difference matrix $\triangle\tilde{\mathbf{W}}^t$ into two smaller matrices $\mathbf{A}$ and $\mathbf{B}$, such that $\triangle\tilde{\mathbf{W}}^t = \mathbf{AB}$. $\mathbf{A}$ is an $n \times m$ matrix comprised of rows of zeros or rows of the m^{th} order identity matrix, $\mathbf{I}_m$, and $\mathbf{B}$ is a $m \times n$ matrix whose rows specify the actual differences between $\tilde{\mathbf{W}}^t$ and $\tilde{\mathbf{W}}^{t-1}$. For example,

if $\triangle\tilde{\mathbf{W}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$, then $\mathbf{AB} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$.

Let $\tilde{\mathbf{W}}^t = \mathbf{U}^t\mathbf{\Sigma}^t\mathbf{V}^t$, we first introduce how to update $\mathbf{U}^t$, $\mathbf{\Sigma}^t$, and $\mathbf{V}^t$ using $\mathbf{U}^{t-1}$, $\mathbf{\Sigma}^{t-1}$, $\mathbf{V}^{t-1}$, and $\triangle\tilde{\mathbf{W}}^t$, instead of re-doing the SVD decomposition.

Since,

$$\begin{aligned} \tilde{\mathbf{W}}^t &= \tilde{\mathbf{W}}^{t-1} + \triangle\tilde{\mathbf{W}}^t \\ &= \mathbf{U}^{t-1}\mathbf{\Sigma}^{t-1}\mathbf{V}^{t-1} + \mathbf{AB} \end{aligned}$$

Then,

$$(\tilde{\mathbf{U}}^{t-1})(\tilde{\mathbf{W}}^t)(\tilde{\mathbf{V}}^{t-1}) = \mathbf{\Sigma}^{t-1} + (\tilde{\mathbf{U}}^{t-1})\mathbf{AB}(\tilde{\mathbf{V}}^{t-1})$$

Let,

$$\mathbf{C} = \mathbf{\Sigma}^{t-1} + (\tilde{\mathbf{U}}^{t-1})\mathbf{AB}(\tilde{\mathbf{V}}^{t-1})$$

Now, compute the low-rank approximation for $\mathbf{C}$. Since $\mathbf{C}$ is a small $k \times k$ matrix, this step can be finished efficiently. Assume $\mathbf{C} = \mathbf{U}_C\mathbf{\Sigma}_C\mathbf{V}_C$, we have:

$$\begin{aligned} \mathbf{U}^t &= \mathbf{U}^{t-1}\mathbf{U}_C \\ \mathbf{V}^t &= \mathbf{V}_C\mathbf{V}^{t-1} \\ \mathbf{\Sigma}^t &= \mathbf{\Sigma}_C \end{aligned}$$

With this result, we next introduce how to update $\mathbf{K}_u^t$, $\mathbf{K}_v^t$, and $\mathbf{\Lambda}^t$. Let $\mathbf{K}_{uc} = \mathbf{U}_C \otimes \mathbf{U}_C$, $\mathbf{K}_{vc} = \mathbf{V}_C \otimes \mathbf{V}_C$, and $\mathbf{K}_{\mathbf{\Sigma}} = \mathbf{\Sigma}_C \otimes \mathbf{\Sigma}_C$.

[2]How to adapt our algorithms to these low-rank approximation methods is beyond the scope of the paper.

Algorithm 3 Incremental SimRank Algorithm (Inc_Sim)

INPUT: $\mathbf{K}_u^{t-1}$, $\mathbf{K}_v^{t-1}$, $\triangle\tilde{\mathbf{W}}^t$
OUTPUT: $\mathbf{\Lambda}^t$, $\mathbf{K}_u^t$, $\mathbf{K}_v^t$, $\mathbf{V}_r^t$
ALGORITHM:
01: Decompose $\triangle\tilde{\mathbf{W}}^t = \mathbf{AB}$ as discussed in Section 5
02: Let $\mathbf{C} = \mathbf{\Sigma}^{t-1} + (\tilde{\mathbf{U}}^{t-1})\mathbf{AB}(\tilde{\mathbf{V}}^{t-1})$
03: Do low-rank approximation for $\mathbf{C} = \mathbf{U}_C\mathbf{\Sigma}_C\mathbf{V}_C$
04: $\mathbf{K}_{uc} = \mathbf{U}_C \otimes \mathbf{U}_C$, $\mathbf{K}_{vc} = \mathbf{V}_C \otimes \mathbf{V}_C$, $\mathbf{K}_\Sigma = \mathbf{\Sigma}_C \otimes \mathbf{\Sigma}_C$
05: Update $\mathbf{K}_u^t = \mathbf{K}_u^{t-1}\mathbf{K}_{uc}$
06: Update $\mathbf{K}_v^t = \mathbf{K}_{vc}\mathbf{K}_v^{t-1}$
07: Update $\mathbf{\Lambda}^t = (\mathbf{K}_\Sigma^{-1} - c\mathbf{K}_v^t\mathbf{K}_u^t)^{-1}$
08: Compute the right vector $\mathbf{V}_r^t = \mathbf{K}_v^t vec(\mathbf{I})$

We have:

$$\begin{aligned}
\mathbf{K}_u^t &= \mathbf{U}^t \otimes \mathbf{U}^t \\
&= \mathbf{U}^{t-1}\mathbf{U}_C \otimes \mathbf{U}^{t-1}\mathbf{U}_C \\
&= (\mathbf{U}^{t-1} \otimes \mathbf{U}^{t-1})(\mathbf{U}_C \otimes \mathbf{U}_C) \\
&= \mathbf{K}_u^{t-1}\mathbf{K}_{uc} \\
\mathbf{K}_v^t &= \mathbf{K}_{vc}\mathbf{K}_v^{t-1} \\
\mathbf{\Lambda}^t &= (\mathbf{K}_\Sigma^{-1} - c\mathbf{K}_v^t\mathbf{K}_u^t)^{-1}
\end{aligned}$$

The complete pseudo-code to update $\mathbf{\Lambda}^t$, $\mathbf{K}_u^t$, and $\mathbf{V}_r^t$ from time step t-1 to t is given in Algorithm 3.

5.2 Theoretical Justification and Analysis

We have the following lemma for the correctness of Algorithm 3:

LEMMA 1. *If $\tilde{\boldsymbol{W}}^{t-1} = \boldsymbol{U}^{t-1}\boldsymbol{\Sigma}^{t-1}\boldsymbol{V}^{t-1}$, $\tilde{\boldsymbol{W}}^t = \boldsymbol{U}^t\boldsymbol{\Sigma}^t\boldsymbol{V}^t$, $\triangle\tilde{\boldsymbol{W}}^t = \boldsymbol{AB}$, and $\boldsymbol{\Sigma}^{t-1} + (\tilde{\boldsymbol{U}}^{t-1})\boldsymbol{AB}(\tilde{\boldsymbol{V}}^{t-1}) = \boldsymbol{U}_C\boldsymbol{\Sigma}_C\boldsymbol{V}_C$ hold, similarity matrix obtained by executing lines 7-8 of Algorithm 2 based on the output of Algorithm 3 is exactly the same as if we called Algorithm 2 for the time step t from the scratch.*

***proof**: Similar as for Theorem 3. Omitted for brevity.*

By lemma 1, the three matrices $\mathbf{\Lambda}^t$, $\mathbf{K}_u^t$, and $\mathbf{K}_v^t$ produced by Algorithm 3 are exactly the same as if we had executed lines 1-6 of Algorithm 2 for time step t from the scratch. Therefore, we have the following corollary:

COROLLARY 1. *Similarity matrix obtained by executing lines 7-8 of Algorithm 2 based on the output of Algorithm 3 has exactly the same approximation accuracy as Algorithm 2.*

In terms of incremental update efficiency, we have the following lemma for Algorithm 3:

LEMMA 2. *The computation cost of Algorithm 3 is bounded by $O(n^2)$.*

***proof**: The main incremental computation cost consists of the following parts:*

1. *decomposing $\triangle\tilde{\boldsymbol{W}}^t$, $(O(mn))$;*
2. *multiplication $\tilde{\boldsymbol{U}}^{t-1}$ and $\boldsymbol{A}$, $(O(kmn))$;*
3. *multiplication $\boldsymbol{B}$ and $\tilde{\boldsymbol{V}}^{t-1}$, $(O(kmn))$;*
4. *low-rank approximation for a small $k \times k$ matrix, $(O(k^2))$;*
5. *updating $\boldsymbol{K}_u^t$ or $\boldsymbol{K}_v^t$, $(O(nkk'))$, k' is the low-rank of C);*
6. *multiplication $\boldsymbol{K}_v^t$ and $\boldsymbol{K}_u^t$, $(O(n^2(k')^4))$;*
7. *inversion of $(\boldsymbol{K}_\Sigma^{-1} - c\boldsymbol{K}_v^t\boldsymbol{K}_u^t)$, $(O(k')^6)$;*
8. *computation of the right vector $\boldsymbol{V}_r^t$, $(O(n^2(k')^2))$;*

since $k \ll n$ and $k' \ll k$, the overall incremental computation cost is bound by $O(n^2)$.

Algorithm 4 Similarity tracking Algorithm (S_Track)

INPUT: The normalized adjacency matrix $\tilde{\mathbf{W}}$, $\triangle\tilde{\mathbf{W}}^{t_1}$, ..., $\triangle\tilde{\mathbf{W}}^{t_x}$, query node i, parameter N
OUTPUT: N most similar nodes of i
ALGORITHM:
01: Initialization (lines 1-6 of Algorithm 2)
02: **For** each time step t_i **Do**
03: **For** each node j **Do**
04: Compute $\mathbf{S}(i,j)$ (lines 7-8 of Algorithm 2)
05: **End**
06: Sort $\mathbf{S}(i,:)$ in descent order
07: Output top N nodes according to $\mathbf{S}(i,:)$
08: Incremental update (Algorithm 3)
09: **End**

6. NODE ANALYSIS

The non-iterative computation framework promises efficient processing of various node analysis. Following, we present two examples: node similarity tracking and node centrality tracking.

6.1 Node Similarity Tracking

In many real setting, the networks are evolving and growing over time, e.g., new links arrive or link weights change. For node similarity tracking, our task is to return the N most similar nodes for query node i at each time step t. For example, over a dynamic coauthor network, we want to answer "Who are the most similar authors to Prof. Jennifer Widom in the past five years?". Given an image network, we want to know "which are the N most similar images to a certain query image?".

Based on the algorithms we developed in previous sections, we can easily give the solution for the problem. The pseudo-code for similarity tracking is summarized in Algorithm 4. At the very beginning, we use lines 1-6 of Algorithm 2 to do an initialization for the matrices $\mathbf{K}_u$, $\mathbf{\Lambda}$, and $\mathbf{K}_v$. Then, at each time step t, we perform the query phase of Algorithm 2 and return the N most similar nodes of i; and after that we call Algorithm 3 to update $\mathbf{K}_u^t$, $\mathbf{K}_v^t$, and $\mathbf{\Lambda}^t$ to prepare for the query at next time step.

6.2 Node Centrality Tracking

For node centrality tracking, our task is to return the N most central nodes for the whole network at each time step t. For example, over a dynamic coauthor network, we want to track the top-5 most central/influential authors over time. Given an image network, we want to know "which are the N most representative images for an object?".

First, we introduce how to define the centrality measure based on SimRank. Intuitively, a node that has larger average SimRank score from all nodes in the graph would have high centrality. To our surprise, experiments show exactly opposite result: central nodes have low average SimRank scores, while peripheral nodes have high scores. For example, the SimRank similarity matrix for the

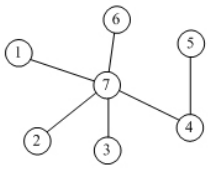


Figure 1: Example of a Network

network in Figure 1 is:

$$\begin{bmatrix} 1.00 & 0.90 & 0.90 & 0.79 & 0.00 & 0.90 & 0.00 \\ 0.90 & 1.00 & 0.90 & 0.79 & 0.00 & 0.90 & 0.00 \\ 0.90 & 0.90 & 1.00 & 0.79 & 0.00 & 0.90 & 0.00 \\ 0.79 & 0.79 & 0.79 & 1.00 & 0.00 & 0.79 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.75 \\ 0.90 & 0.90 & 0.90 & 0.79 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.75 & 0.00 & 1.00 \end{bmatrix}$$

The average SimRank scores of nodes 1-7 are:

$$[0.64 \quad 0.64 \quad 0.64 \quad 0.59 \quad 0.25 \quad 0.64 \quad 0.25]$$

It can be seen that the "central" node 7 in Figure 1 has the smallest score (0.25), while the "peripheral" nodes 1,2,3, and 6 in Figure 1 have the largest scores (0.64). The result actually makes good senses if we consider the semantic of SimRank: peripheral nodes have high average SimRank scores because the central nodes exist as their common ancestors, while central nodes themselves have low average scores because they have few common ancestors.

On the other hand, other factors such as the size of the network and the degree of a node would affect the centrality value as well. A node locates in a larger network or has a larger degree should have higher centrality.

In this paper, we combine several factors to gauge the centrality of a given node, instead of solely relying on the closeness (similarity).

DEFINITION 3 (CENTRALITY). *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the centrality of node $i \in \mathcal{V}$ is defined as:*

$$C(i) = F_s(i) \cdot F_d(i) \cdot F_c(i)$$

where $F_s(i)$ is the SimRank factor, $F_d(i)$ is the degree factor, and $F_c(i)$ is the connected component factor.

$C(i)$ consists of three components: $F_s(i)$, $F_d(i)$, and $F_c(i)$. We elaborate on each component in the following.

- $F_s(i)$ is the deciding factor in centrality. According to the above observation, the lower average similarity a node has, the more central it is. So $F_s(i)$ is defined as:

$$F_s(i) = 1 - \frac{1}{n}\sum_{j=1}^{n} \mathbf{S}(i,j)$$

- $F_d(i)$ represents the degree of node i. It is introduced to prevent those nodes which have few neighbors from being considered as a central node. For example, node 5 in Figure 1 has the same average SimRank score as node 7, but it is not a central node.
- $F_c(i)$ measures the connectivity of the network. Let $Con(i)$ be the connected component that node i belongs to, and $Size$ $(Con(i))$ be the size (i.e., the number of nodes) of $Con(i)$, $F_c(i)$ is defined as:

$$F_c(i) = \frac{Size(Con(i))}{n}$$

Algorithm 5 Centrality Tracking Algorithm (C_Track)

INPUT: The normalized adjacency matrix $\tilde{\mathbf{W}}$, $\triangle\tilde{\mathbf{W}}^{t_1}$, ..., $\triangle\tilde{\mathbf{W}}^{t_x}$, parameter N
OUTPUT: N most central nodes
ALGORITHM:
01: Initialization (lines 1-6 of Algorithm 2)
02: **For** each time step t_i **Do**
03: Compute **S** (lines 7-8 of Algorithm 2)
04: Compute centrality for each node
05: Output top N nodes with large centrality values
06: Incremental update (Algorithm 3)
07: **End**

One might argue that, if we just use $F_d(i)$ as the centrality measure, we can identify node 7 as well. This alternative, however, has two major drawbacks compared with our centrality measure. First, it only considers one-step connections. Second, it is not sensitive to any underlying network structure. For example, if we apply this measure to a binary tree, it would fail to find the central nodes.

The algorithm for tracking node centrality (C_Track) is summarized in Algorithm 5. It is quite similar to Algorithm 4, and we omit its details for space.

7. EMPIRICAL RESULTS

To evaluate the effectiveness and efficiency of our algorithms, we conducted extensive experiments. We implemented all experiments on a PC with Intel Xeon 2.0GHz CPU, 2.0GB main memory and 200G hard disk, running Microsoft Windows Server 2003 Edition. We first present a comprehensive study using the synthetic datasets, which shows high effectiveness and efficiency of our algorithms. We then evaluate the effectiveness of our tracking algorithms on two real data sets, the DBLP and Image data.

7.1 Experiments on synthetic datasets

We generated information networks using the complex network Package[3]. Configuration parameters for generating networks are as follows: (1) node number: 10000; and (2) total edge number: 135938.

7.1.1 Efficiency

In this experiment, we conducted experiments to evaluate the efficiency of our proposed approximate algorithms.

The runtime reported here includes the I/O time. We compare the performance of our algorithms (*NI_Sim* and *Inc_Sim*) with the iterative algorithm in [9] (*Ite_Sim*). When update is made to the original network, we re-run *Ite_Sim*. Although we realize that it is not viable to re-compute the whole matrix every time the network is updated, there is no other reasonable benchmark for comparison. Our experiments show that the non-iterative algorithm without improvement (*N_Sim*) can be up to a hundred time slower than the improved version of the non-iterative algorithm, as such we will only report results for the improved non-iterative algorithm (*NI_Sim*).

In the *NI_Sim* algorithm, parameter k is set to 5, 10, 15, 20 and 25 respectively in the experiments, while c is set to 0.8. In the *Ite_Sim* algorithm, the iterative time is set to 5 and 10, while c is set to 0.8 which is the same as *NI_Sim*.

[3]http://www.levmuchnik.net/Content/Networks/ComplexNetworksPackage.html

We first compare the runtime of *NI_Sim* and *Ite_Sim* on the static information network. Fig. 2(a) depicts the runtime for computing SimRank scores. The X-axis shows the number of query node pairs, and Y-axis shows the corresponding runtime in log scale. From this figure we can observe that *NI_Sim* is about 100 times faster than *Ite_Sim* when k is set to 25. The runtime can be further significantly reduced if a smaller k is set.

We next report the results on the dynamic network to evaluate the performance of our proposed incremental algorithm *Inc_Sim*. We observe that after initialization, at each step, most time is spent on updating matrices $\mathbf{K}_u^t$, $\mathbf{K}_v^t$, and $\mathbf{\Lambda}^t$.

To simulate a dynamic network environment, we first fetch 105938 edges to construct an initial network, then add 2000, 4000, 6000, 8000, and 10000 edges at each time step. Thus we have 5 time steps in total.

Figure 2(b) shows the update time with respect to the number of changed edges. Compared to *Ite_Sim*, *Inc_Sim* with small values of parameter k is much faster, achieving average 100x speed-up when k is 5 and 10x speed-up when k is 10.

7.1.2 Effectiveness

In this experiment, we evaluate the effectiveness of our algorithms. We adopted two widely-used measures, AvgDiff and NDCG, to evaluate the accuracy of our methods.

Average Difference (AvgDiff): Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of size n, assume $Sim_{ni}(i,j)$ represents SimRank values returned from *NI_Sim*, and $Sim_{it}(i,j)$ represents SimRank values returned from *Ite_Sim*, where $i, j \in \mathcal{V}$, AvgDiff is defined as

$$AvgDiff = \frac{\sum_{i,j \in V} |Sim_{ni}(i,j) - Sim_{it}(i,j)|}{n^2}$$

It is easy to see that AvgDiff actually measures the absolute accuracy of computation results.

Normalizing Discounted Cumulative Gain (NDCG): NDCG is widely used to measure the accuracy of multi-level ranking model [27, 28]. Here, it is used to measure the relative accuracy of computation results. Since our non-iterative computation model slightly differs from the original SimRank iterative computation model (for example, we do not set the diagonal values of $\mathbf{S}$ to 1 as SimRank does at each iteration), the absolute value difference of SimRank scores generated by *NI_Sim* and *Ite_Sim* may be large. But we contend that this is not a big problem if the relative ranking of node similarities keeps almost the same. For example, in Fig. 1, we order all nodes according to their similarities to node 1. If the two ranking lists, based on the computation results of *NI_Sim* and *Ite_Sim*, are all (1, 2, 3, 6, 4, 5, 7), we say algorithm *NI_Sim* has 100% accuracy since it loses nothing on precision of ranking results.

Given a query node, NDCG at position p is defined as

$$NDCG_p = \frac{1}{Z_p} \sum_{i=1}^{p} \frac{2^{rank_i} - 1}{\log_2(i+1)}$$

where p denotes position, $rank_i$ denotes the SimRank score of rank i from *NI_Sim*, and Z_p is a normalization factor to guarantee that NDCG of a perfect ranking generated by *Ite_Sim* at position p equals 1. In evaluation, NDCG is further averaged over all nodes. Considering that users always concern about the top-K SimRank scores, we will only report NDCG@5 and NDCG@10 in our experiments.

Figure 3(a) shows the average difference between results from *NI_Sim* and *Ite_Sim*. Clearly, the average difference is rather small (0.003 for k=25). When the parameter k increases, the average difference decreases monotonically, meaning that higher accuracy can be achieved with higher k.

Figure 3(b) shows the NDCG@5 and NDCG@10 of *NI_Sim*. One can find that, with a rank-25 approximation, our method can achieve very high ranking accuracy (94% for NDCG@5 and 92% for NDCG@10). When higher value of parameter k is set, the accuracy can be further enhanced.

7.1.3 Pre-Computation Cost

In Figures 4(a) and 4(b), results are evaluated from two perspectives: pre-compute time (PT) vs. # of nodes and pre-storage (PS) vs. parameter k. We increase the number of nodes from 10k to 50k. Figure 4(a) shows that although both algorithms are of linear scalability, the run time of the *NI_Sim* algorithm scales better than that of the *Ite_Sim* algorithm. When k is larger than 25, the pre-computation of *NI_Sim* may consume more time than *Ite_Sim* with 5 iterations. As discussed in Section 4, this result is acceptable since it can be done off-line and once for all. It is also worthwhile since it hugely benefits runtime query performance as shown in Figure 2(a).

7.2 Experiments on Real datasets

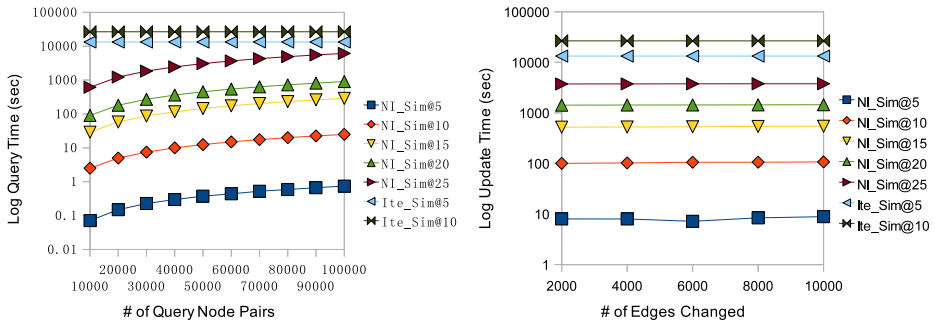
7.2.1 DBLP Dataset

The experiment is used to verify the effectiveness of our S_Track algorithm. We extract the 10-year (from 1998 to 2007) author-paper-term information from the whole DBLP data set[4]. Every two publication years form a time step, so there are 5 time steps in total. For each time step, we construct an information network. The network nodes represent authors, papers, or terms, and edges represent author-paper or paper-term relationships. We restrict papers published on 7 major conferences ('ICDE', 'ICML', 'KDD', 'SIGIR', 'SIGMOD', 'VLDB', 'WWW'), and get 5782 papers in total. We rank authors according to the number of papers they published on these conferences and take the top-1000 authors. Similarly, we rank terms according to their occurrence frequency in titles of papers and take the top-1000 terms. Thus, there are 7782 nodes in total with an average of 10041 edges per time step.

Figures 5 and 6 list the top-10 most similar terms and authors for 'Prof. Jennifer Widom' over the years. The results make good sense. The terms list in Figure 5 indicate the major research interest of 'Prof. Widom' is changing over time. For example, during 1998-2001, her major interests were semi-structured data processing ('xml' and 'semistructured') and data warehouse ('warehouse' and 'incremental'). Her research group developed and declared the 'Lore' and 'WHIPS' prototype systems at that time. However, during 2002-2005, data stream attracted lots of her attention ('stream' and 'continuous'), while recently (during 2006-2007), uncertainty and data lineage techniques became one of her new research focuses ('uncertainty','integrity', and 'uncertain'). In response to the change of her research interest, one can find that her top-10 most similar authors have changed accordingly. For example, in the years 2000-2001, the top similar authors of 'Prof. Widom' are: (1)'Jun Yang' and 'Chris Olston' (they are Prof. Widom's students); and (2) 'Latha S. Colby' and 'Ramez Elmasri'(they share similar research topics, xml or data warehouse, with 'Prof. Widom' at that time). During 2006-2007, the top similar authors of 'Prof. Widom' are 'Omar Benjelloun' and 'Anish Das Sarma', and they are Prof. Widom's coauthors.

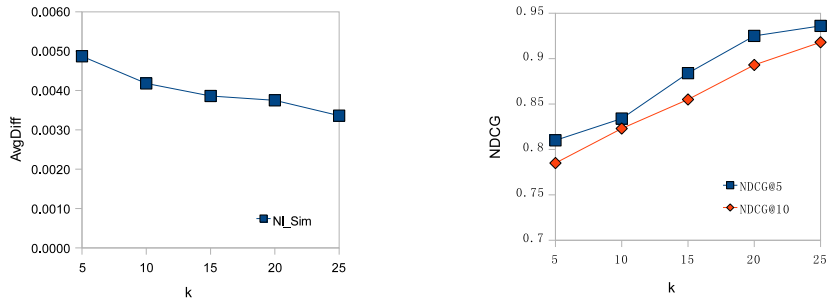
7.2.2 Image Dataset

[4] http://kdl.cs.umass.edu/data/dblp/dblp-info.html



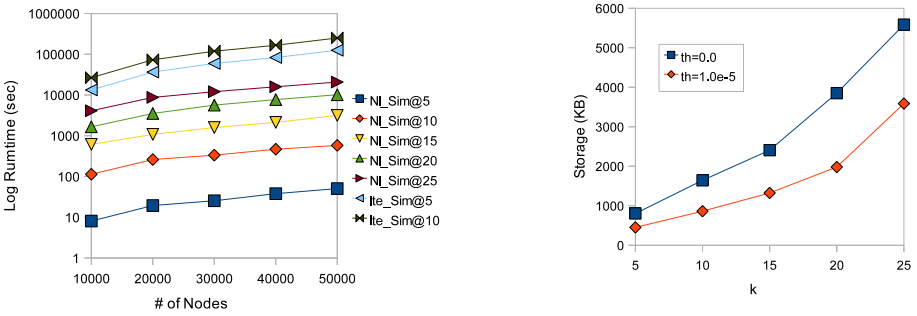
(a) Query Time vs. # of Query Node Pairs (b) Update Time vs. # of Update Edges

Figure 2: Efficiency of Algorithms



(a) AvgDiff vs. k (b) NDCG vs. k

Figure 3: Effectiveness of Algorithms



(a) PT vs. # of Nodes (b) PS vs. k

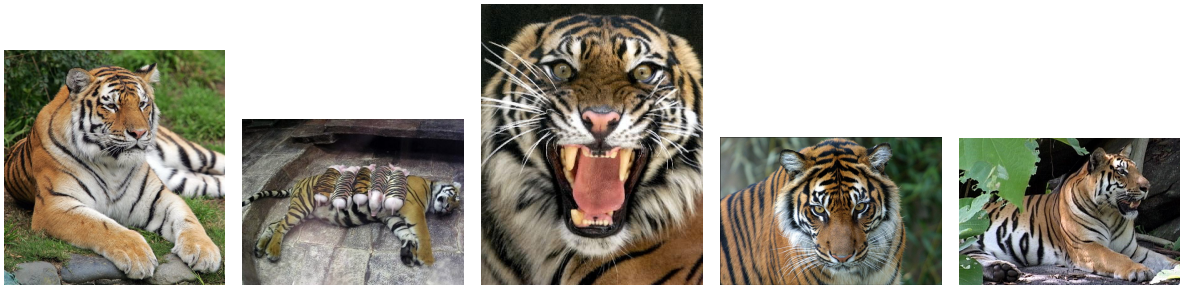
Figure 4: Pre-computation Cost

1998-1999	2000-20001	2002-2003	2004-2005	2006-2007
represent	extension	current	paper	pipelined
xml	tradeoff	panel	conference	work
change	trigger	manager	panel	uncertainty
semistructured	issue	context	review	uncertain
optimization	precision	personalize	pipelined	clean
rewrite	practical	stream	property	replicate
impact	replicate	structural	limit	integrity
query	warehouse	source	memory	skew
power	incremental	continuous	route	cad
protein	transformation	minimization	plan	service

Figure 5: Top-10 Most Similar Terms for 'Prof. Jennifer Widom' up to Each Time Step

1998-1999	2000-2001	2002-2003	2004-2005	2006-2007
Sudarshan Chawathe	Jun Yang	Chris Olston	Kamesh Munagala	Omar Benjelloun
Serge Abiteboul	Chris Olston	Shivnath Babu	Shivnath Babu	Anish Das Sarma
Ravi Krishnamurthy	Wilburt Labio	Mayur Datar	Zachary G. Ives	Rajeev Motwani
Z. Meral Ozsoyoglu	Stefano Ceri	Arvind Arasu	Rajeev Motwani	Shawn R. Jeffery
Gultekin Ozsoyoglu	Latha S. Colby	Roger King	Arvind Arasu	Wei Hong
Lei Sheng	William McKenna	Sudarshan Chawathe	Richard Snodgrass	Alon Halevy
Jarek Gryz	Roberta Cochrane	Joseph Hellerstein	Utkarsh Srivastava	Utkarsh Srivastava
Kalervo Jarvelin	Felipe Carino	David Maier	Christian Jensen	Kamesh Munagala
Sumit Ganguly	Ramez Elmasri	Tomasz Imielinski	David DeWitt	Michael J. Franklin
Limsoon Wong	Jose A. Blakeley	B. R. Badrinath	Kyu-Young Whang	Gustavo Alonso

Figure 6: Top-10 Most Similar Authors for 'Prof. Jennifer Widom' up to Each Time Step



(a) (b) (c) (d) (e)

Figure 7: Top-5 Most Central Images for 'tiger'

This experiment is used to evaluate the effectiveness of our centrality measure. The image dataset is obtained by querying on Google's Image Search and downloading the top result images (We download about 100 images for different query images such as 'tiger', 'white house', and we have totally 1019 images). We use two types of image features: color and texture features. For RGB color histogram, we set 10 bins for each of the three colors, and thus we have 30 dimensions. For the texture features, we have 22 dimensions[5]. There are overall 52 dimensions. Each color dimension is normalized to be in the range [0, 1], and each texture dimension is normalized to be within [0, 0.5] to set lower weight to the texture features. We first use Euclidean distance to calculate a similarity matrix, then convert the similarity matrix to a network by assigning an edge between two nodes when their similarity is over a threshold. After that, we run *C_Track* on this network and get the top-5 images as the result.

Figure 7 shows the top-5 central images for query image 'tiger'. We can find that the result makes perfect sense. All these images are the representative images for the given query. The result confirms the appropriateness of our centrality measure as indicator of "actual" centrality.

7.2.3 Wikipedia Dataset

Our practical goal of implementing the suggested algorithms was to speed up the time-consuming SimRank score computation on large data graph, such as Wikipedia.

Wikipedia[6] is "a multilingual, Web-based, free-content encyclopedia project which is written collaboratively by volunteers from all around the world". The English version of Wikipedia is the largest version among the available versions in many languages. There are 2.8M articles in English Wikipedia by March 2009. As a most popular online encyclopedia, Wikipedia has recently obtained a big interest from academic communities, such as [29] and [30].

In this experiment, we want to compute the SimRank scores of Wikipedia. With these scores, many applications including word disambiguation and concept classification can be conducted.To this end, we organized data from Wikipedia into the SimRank graph model by the method presented in [30]. An article in Wikipedia, which describes a single encyclopedia concept, becomes an node in the graph. The relationships "an article belongs to a category which is also an article itself" is chosen to be links in the graph. Note that, category links constitute a subset of links in Wikipedia, so the graph covers only a subset of the whole Wikipedia.

We set the threshold T to be 1.0e-6. For k=15, the pre-compute time of the Wikipedia dataset is approx. 5.68 hours, and the query time for every 1000 node pairs is 3.718 seconds. The result is promising and indicates that the non-iterative method proposed in this paper is practical and can scale well on large graphs.

8. RELATED WORK

A great many analytical techniques have been proposed toward a better understanding of information networks and their properties. Below we briefly describe the work that is most relevant to the current work. It can be categorized into three parts: static network analysis, dynamic network analysis, and the methods for low-rank approximation.

Static Network Analysis. There is a lot of research work on static information network analysis, including power laws discovery [1], frequent pattern mining [2, 3], clustering and community identification [4, 5], and node ranking [6, 7].

In terms of node similarity, a great number of measures have been reported. With respect to the focus of this paper, a detailed dis-

[5]use the Matlab code from http://www.mathworks.com/ matlabcentral/fileexchange/22187

[6]http://www.wikipedia.org/

cussion is given to link-based similarity measures. [11] proposed to use the total delivered current as the similarity measure. [10] developed a new way of measuring and extracting proximity in networks called ąřcycle free effective conductanceąś (CFEC). Random walk with restart model is used to compute the node similarity in [18, 19, 20]. In [12], the authors proposed a similarity measure based on the principle that i is similar to j if i has a network neighbor v that is itself similar to j.

Similar to SimRank, Xi et al. proposed another similarity-calculating algorithm called SimFusion that also utilizes the idea of recursively computing node similarity scores based on the scores of neighboring nodes [31]. The key of these methods is that they are applicable to any domain with object-to-object relationships. Nevertheless, the time complexity of the straightforward SimRank or SimFusion computation becomes a substantial obstacle for using them on practical applications. A variety of optimization techniques have been proposed to reduce the computation cost of SimRank. [15] presented a scalable framework for SimRank computation based on Monte Carlo method, in which three optimization strategies, finger-print trees, coupled random walks generation, and parallelization possibilities for SimRank computation, are proposed. [14] introduced another three excellent optimization ideas, selecting essential node pairs, partial sums, and threshold-sieved similarities, to speed up the computation of SimRank. As discussed in Section 1, they are all under the iterative framework. In contrast, the method introduced in this paper optimizes SimRank in a non-iterative mode.

Dynamic Network Analysis. Recently, there is an increasing interest in mining dynamic networks, such as group or community evolution [32, 33], power laws of dynamic networks [34], dynamic tensor analysis [35], and dynamic clustering [36]. In terms of similarity and centrality tracking, to the best of our knowledge, the only existing work is [20]. The authors proposed two fast algorithms to update the similarity matrix incrementally based on the Random Walk with Restart (RWR) model. Experiments on real data show that their methods are effective and efficient. Unfortunately, this paper has one inherent limitation: it is tailored to bipartite graphs; efficient extension of the method to graphs which are not bipartite is hard since the incremental algorithms rely on the assumption that one of the two partitions is small, and updates involve a small number of nodes in one of the two partitions. In contrast, the method introduced in this paper can be applied to arbitrary graphs.

Low-Rank Approximation. The SVD [21] principle has been successfully used for diverse applications, such as latent semantic index (LSI) [37], principle component analysis (PCA) [38], and so on. For LSI, a term-document matrix is constructed to describe the occurrences of terms in documents; it is a sparse matrix whose rows correspond to terms and whose columns correspond to documents. Here, the SVD principle is used to deal with linguistic ambiguity issues by calculating the best rank-k approximation of the keyword-document matrix. For image compression, an image is represented as a matrix. SVD is used to look for a compressed image to reduce the overhead for disk storage and network transmission. More recently, for sparse matrices, some new matrix decomposition techniques have been proposed; for instance, P. Drineas et al. proposed CUR [24] and J. Sun et al. proposed CMD [25].

9. DISCUSSION AND CONCLUSION

9.1 Discussion

We can further reduce the pre-computation and storage cost as shown in the following:

1. We observed that the most time-consuming part in the pre-computation is the multiplication of Ku and Kv. In our experiments, we only used the naive and non-parallel matrix multiplication method. Actually, we tried to improve the efficiency by utilizing various hardware acceleration methods (including GPU, Multi-core, and cluster). The results of our parallel methods are promising (for example, only several hours when k is 100 on a 256M NVIDIA GeForce 9600GT GPU). We are currently comparing different parallel methods.

2. We can improve the low-rank approximation performance by exploring the parallel too. Recently, parallel SVD algorithm[7] has been implemented by Google. We believe this can be utilized to enhance our non-iterative computation method.

3. Although the SVD gives the best approximation in terms of squared error, it does not preserve sparsity, i.e., after the decomposition, most of the entries of the result matrices are non-zero, even if the original matrix is sparse. Other low-rank approximation alternatives such as CUR and CMD can be considered. Since these methods can generate a sparse representation of the original matrix, dramatic saving in pre-computation and storage can be achieved.

9.2 Conclusion

This paper addresses the issues of optimization as well as incremental update of SimRank for static and dynamic information networks. We have proposed a novel non-iterative framework for SimRank computation. Based on this, we have developed three efficient algorithms to compute SimRank scores for static and dynamic information network. We provide theoretical guarantee for our methods and demonstrate its applications on real information network analysis. Extensive experimental studies on synthetic and real data sets verified the effectiveness and efficiency of the proposed methods. Overall, we believe that we have provided a new paradigm for exploration of and knowledge discovery in large information networks. This work is just the first step, and there are many challenging issues. We are currently investigating into detailed issues as a further study.

10. ACKNOWLEDGMENTS

The work was supported in part by NASA grant NNX08AC35A, the U.S. National Science Foundation grants IIS-08-42769 and IIS-09-05215, China National 863 grant 2008AA01Z120, and NSFC grants 60673138 and 60603046. Any opinions, findings, and conclusions expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

11. REFERENCES

[1] M.E.J.Newman, "The structure and function of complex netwroks," *SIAM Review*, 2003.

[2] X. Yan, P. S. Yu, and J. Han, "Substructure similarity search in graph databases," in *Proc. Of ACM-SIGMOD Int'l Conference on Management of Data*, 2005.

[3] X. Yan and J. Han, "Closegraph: Mining closed frequent graph patterns," in *Proc. of the 9th Int'l Conference on Knowledge discovery and data mining(KDD'03)*, 2003.

[7] http://googlechinablog.com/2007/01/blog-post.html

[4] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Proc. Of the Advances in Neural Information Processing Systems(NIPS)*, 2002.

[5] M. Girvan and M. Newman, "Community structure in social and biological networks," in *Proc. Of the National Academy of Sciences*, 2002.

[6] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," *Technical report, Stanford University Database Group, http://citeseer.nj.nec.com/368196.html*, 1998.

[7] J. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM*, 1999.

[8] P. Ganesan, H. Garcia-molina, and J. Widom, "Exploiting hierarchical domain structure to compute similarity," *ACM Transactions on Information Systems*, vol. 21, pp. 64–93, 2003.

[9] G. Jeh and J. Widom, "Simrank: a measure of structural-context similarity," in *Proc. of the 8th Int'l Conference on Knowledge discovery and data mining(KDD'02)*, 2002.

[10] Y. Koren, S. North, and C. Volinsky, "Measuring and extracting proximity in networks," in *Proc. of the 12th Int'l Conference on Knowledge discovery and data mining(KDD'06)*, 2006.

[11] C. Faloutsos, K. S. McCurley, and A. Tomkins, "Fast discovery of connection subgraphs," in *Proc. of the 10th Int'l Conference on Knowledge discovery and data mining(KDD'04)*, 2004.

[12] E. Leicht, P. Holme, and M. Newman, "Vertex similarity in networks," *Phys. Rev.*, vol. 026120, p. E 73, 2006.

[13] A. G. Maguitman, F. Menczer, F. Erdinc, H. Roinestad, and A. Vespignani, "Algorithmic computation and approximation of semantic similarity," in *Proc. of the 15th Int'l Conference on World Wide Web (WWW'06)*, 2006.

[14] D. Lizorkin, P. Velikhov, M. Grinev, and D. Turdakov, "Accuracy estimate and optimization techniques for simrank computation," in *Proc. of the 34st Int'l Conference on Very Large Databases (VLDB'08)*, 2008.

[15] D. Fogaras and B. Racz, "Scaling link-based similarity search," in *Proc. of the 14th Int'l Conference on World Wide Web (WWW'05)*, 2005.

[16] P. Benner, "Factorized solution of sylvester equations with applications in control," in *Proc. of the 16th International Symposium on Mathematical Theory of Network and Systems (MTNS 2004)*, 2004.

[17] A. J. Laub, *Matrix Analysis for Scientists and Engineers*. Society for Industrial and Applied Mathematics, 2004.

[18] J. Pan, H. Yang, C. Faloutsos, and P. Duygulu, "Automatic multimedia cross-modal correlation discovery," in *Proc. of the 9th Int'l Conference on Knowledge discovery and data mining(KDD'04)*, 2004.

[19] H. Tong, C. Faloutsos, and J. Pan, "Fast random walk with restart and its application," in *Proc. IEEE 2001 Int. Conf. Data Mining (ICDM'06)*, 2006.

[20] H. Tong, S. Papadimitriou, P. S. Yu, and C. Faloutsos, "Proximity tracking on time-evolving bipartite graphs." in *Proc. of SDM*, 2008.

[21] G. Golub and C. Loan, *Matrix Computation.* Johns Hopkins, 1996.

[22] W. Piegorsch and G. Casella, "Inverting a sum of matrices," *SIAM Rev.*, vol. 32, pp. 470–470, 1990.

[23] M. Stoll, "A krylov-schur approach to the truncated svd," in *NA Group technical reports, http://www.comlab.ox.ac.uk/files/721/NA-08-03.pdf*, 2008.

[24] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," *Technical report, Stanford University Database Group, http://citeseer.nj.nec.com/368196.html*, 1998.

[25] J. Sun, Y. Xie, H. Zhang, and C. Faloutsos., "Less is more: Compact matrix decomposition for large sparse graphs," in *Proc. of SDM*, 2007.

[26] M. W. Berry, S. T. Dumais, and G. W. O'brien, "Using linear algebra for intelligent information retrieval," *SIAM Rev.*, vol. 37, pp. 573–595, 1995.

[27] C. Burges, T. Shaked, E. Renshaw, A. .Lazier, M. Deeds, N. Hamilton, and G. Hullender., "Learning to rank using gradient descent," in *Proc. 22th Int. Conf. Machine Learning (ICML'05)*, 2005.

[28] K. Jarvelin and J. Kekalainen, "Cumulated gain-based evaluation of ir techniques," *ACM Transactions on Information Systems*, 2002.

[29] L. Buriol, C. Castillo, D. Donato, S. Leonardi, and S. Millozzi, "Temporal analysis of the wikigraph," in *Proceedings of the Web Intelligence Conference (WI 2006)*. Los Alamitos, CA, USA: IEEE Computer Society, December 2006, pp. 45–51. [Online]. Available: http://www.dcc.uchile.cl/ ccastill/papers/buriol_2006_ temporal_analysis_wikigraph.pdf

[30] D. Lizorkin, P. Velikhov, M. Grinev, and D. Turdakov, "Accuracy estimate and optimization techniques for simrank computation." *PVLDB*, vol. 1, no. 1, pp. 422–433, 2008. [Online]. Available: http://dblp.uni-trier.de/db/journals/pvldb/pvldb1.html

[31] W. Xi, E. A. Fox, W. Fan, B. Zhang, Z. Chen, J. Yan, and D. Zhuang, "Simfusion: measuring similarity using unified relationship matrix," in *Proc. Of the 28th international ACM SIGIR conference on Research and development in information retrieval*, 2005.

[32] C. Tantipathananandh, T. Y. Berger-Wolf, and D. Kempe, "A framework for community identification in dynamic social networks," in *Proc. of the 13th Int'l Conference on Knowledge discovery and data mining(KDD'07)*, 2007.

[33] L. Backstrom, D. Huttenlocher, and J. Kleinberg, "Group formation in large social networks: membership, growth, and evolution," in *Proc. of the 12th Int'l Conference on Knowledge discovery and data mining(KDD'06)*, 2006.

[34] J. Leskovec, J. M. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *Proc. of the 13th Int'l Conference on Knowledge discovery and data mining(KDD'07)*, 2007.

[35] J. Sun, D. Tao, and C. Faloutsos, "Beyond streams and graphs: dynamic tensor analysis," in *Proc. of the 12th Int'l Conference on Knowledge discovery and data mining(KDD'06)*, 2006.

[36] Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng, "Evolutionary spectral clustering by incorporating temporal smoothness," in *Proc. of the 13th Int'l Conference on Knowledge discovery and data mining(KDD'07)*, 2007.

[37] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman., "Indexing by latent semantic analysis." in *Journal of the Society for Information Science*, 1990.

[38] I. Jolliffe, "Principal component analysis," *Springer*, 2002.