Taming Computational Complexity: Efficient and Parallel SimRank Optimizations on Undirected Graphs

Weiren Yu[1,2], Xuemin Lin[2], and Jiajin Le[1]

[1] Donghua University, Shanghai 201620, China
ywr0708@hotmail.com, lejiajin@dhu.edu.cn
[2] University of New South Wales, NSW 2052, Australia
lxue@cse.unsw.edu.au

Abstract. SimRank has been considered as one of the promising link-based ranking algorithms to evaluate similarities of web documents in many modern search engines. In this paper, we investigate the optimization problem of SimRank similarity computation on undirected web graphs. We first present a novel algorithm to estimate the SimRank between vertices in $O\left(n^3 + K \cdot n^2\right)$ time, where n is the number of vertices, and K is the number of iterations. In comparison, the most efficient implementation of SimRank algorithm in [1] takes $O\left(K \cdot n^3\right)$ time in the worst case. To efficiently handle large-scale computations, we also propose a parallel implementation of the SimRank algorithm on multiple processors. The experimental evaluations on both synthetic and real-life data sets demonstrate the better computational time and parallel efficiency of our proposed techniques.

1 Introduction

SimRank is a useful and important similarity measure exploiting the relationships between vertices (web documents) on web graphs. It has been widely studied in the literature [2,3,4,1,5,6,7]. As a multi-step generalization of co-citation [8,9], SimRank similarity is based on the recursive concept that *two different vertices are similar if their neighbors are similar*. SimRank has broad applications involving "find-similar-document" query, search engine optimization, graph clustering, etc.

Existing techniques for SimRank computation can be distinguished into two categories: (i) *probabilistic* method [3,4] that estimates SimRank by the expected value $s\left(a,b\right) = \mathbb{E}\left(c^{\tau(a,b)}\right)$, where $\tau_{(a,b)}$ is a random variable denoting the first meeting time for vertices a and b, and $c \in (0,1)$ is a decay factor; (ii) *deterministic* method [1,5,6,7] that computes SimRank iteratively for finding a fixed point of the SimRank function $s\left(a,b\right)$. Since the latter approach produces better accuracy with high time complexity compared to the probabilistic method, there has been a growing interest in SimRank deterministic optimization over the recent years. To the best of our knowledge, there are several interesting pieces of work which can efficiently reduce the time complexity of SimRank *deterministic* computation. One work [1] is mainly based on *a partial sums function* reducing the computational time from $O\left(Kn^4\right)$ to $O\left(Kn^3\right)$ in the worst case. Another work [7] oriented on *matrix representations* takes $O\left(K \min\left(mn, n^r\right)\right)$ time, where m is the number of edges, and $r \in (2, \log_2 7]$ is a positive constant. Li et al.

[10] develop a novel approximate SimRank computation algorithm for static and dynamic information networks. They claim that their optimization technique is based on the non-iterative framework; however, the singular value decomposition (SVD) method they used for low-rank approximation inherently requires numerical iterations. Hence, their method is in essence iterative, not non-iterative. Details are described in the Related Work section.

Motivations: In spite of the substantial improvement achieved by these existing techniques [1,7], high performance of SimRank computation involving *fast algorithms* and *parallel implementation* is still a challenging problem. However, the time requirements of the available SimRank *deterministic* algorithms are still about cubic in the number of vertices for each iteration, which is costly over large web graphs. Additionally, in terms of *parallelization* strategies for SimRank computation, there is only one research work concerning SimRank optimization [3]. That work is based on SimRank *probabilistic* computation. As for SimRank *deterministic* computation, parallel implementation has not been addressed in scientific literature yet. This motivates our further exploration of SimRank deterministic optimization.

Contributions: In this paper, we provide efficient techniques for SimRank *deterministic* computation and parallelization on undirected graphs. The interesting aspect of our algorithm is that it uses a variant of *eigenvector centrality measure*, and we are able to prove strong theoretical guarantees on its performance. In summary, we make the following contributions:

– We present an efficient spectral decomposition based algorithm for SimRank computation over undirected graphs, which reduces the computational complexity from $O\left(Kn^3\right)$ to $O\left(n^3 + Kn^2\right)$ for K iterations in the worst case. We give theoretical results in Sect. 3.
– We develop a block partition technique in combination with the Parallel Linear Algebra Package (PLAPACK) to parallelize our SimRank algorithm on distributed memory multi-processors for achieving high parallel efficiency. We discuss these approaches in Sect. 4.
– We perform extensive evaluations of our proposed methods by using synthetic and real data sets, demonstrating the efficiency and effectiveness of our algorithms. We show experimental results in Sect. 5.

Organizations: The rest of this paper is organized as follows. Sect. 2 provides formal definition of SimRank similarity measure. Sect. 3 details optimization techniques for AUG-SimRank algorithm. Sect. 4 presents a parallel implementation of AUG-SimRank algorithm. We present experimental evaluations in Sect. 5. Sect. 6 reviews related work. And Sect. 7 concludes this paper.

2 Preliminaries

In this section, we briefly introduce the fundamental concepts, and formulate the basic problem of SimRank similarity scoring on undirected graphs. More details can be found in [2,1,7].

2.1 Problem Statement

Assume an undirected web graph $\mathcal{G} = (V, E)$, where V is the set of vertices representing web documents, and E is the set of edges. Each edge $(p, q) \in E$ between documents p and q corresponds to a reference between p and q. We denote the *derived graph* as $\mathcal{G}^2 = \left(V^2, E^2\right)$, where (a) each vertex in $V^2 = V \times V$ represents a pair (p, q) of vertices in $\mathcal{G}$, and (b) an edge $((p, q), (r, s))$ between (p, q) and (r, s) exists in E^2 *iff* the edges (p, r) and (q, s) exist in $\mathcal{G}$. Now we define a SimRank similarity function on $\mathcal{G}^2$.

Definition 1 (SimRank similarity [2]). *For two arbitrary vertices a and b, let $s : V^2 \to [0, 1] \subset \mathbb{R}$ be a real-valued function on $\mathcal{G}^2$ defined by*

$$s(a,b) = \begin{cases} 1, & a = b; \\ \frac{c}{|N(a)||N(b)|} \sum_{j=1}^{|N(b)|} \sum_{i=1}^{|N(a)|} s\left(N_i(a), N_j(b)\right), & N(a), N(b) \neq \varnothing; \\ 0, & otherwise. \end{cases} \tag{1}$$

where $c \in (0, 1)$ is a constant decay factor , $N(a)$ denotes the set of vertices neighboring to a , $|N(a)|$ is the cardinality of $N(a)$, and an individual member of $N(a)$ is referred to as $N_i(a)$ $\quad (1 \leq i \leq |N(a)|)$. The scalar $s(a, b)$ is called a SimRank similarity score between vertices a and b.

It is interesting to note that Eq.(1) has a unique solution $s(\cdot, \cdot)$ that can be reached by the following iterations to a fixed point.

Lemma 1 (SimRank iterative computation [1]). *For two arbitrary vertices a and b, let $s^{(k)}(\cdot, \cdot)$ be the SimRank value at the k-th iteration. We can use $s^{(k)}(\cdot, \cdot)$ to estimate $s(\cdot, \cdot)$ up to an error of c^{k+1} in worst-case time $O\left(k \cdot n^3\right)$, where $s^{(k)}(\cdot, \cdot)$ is generated by the following iterations:*

$$s^{(0)}(a,b) = \begin{cases} 1, \ a = b; \\ 0, \ a \neq b. \end{cases} \tag{2}$$

$$s^{(k)}(a,b) = \begin{cases} 1, & a = b; \\ \frac{c}{|N(a)||N(b)|} \sum_{j=1}^{|N(b)|} \sum_{i=1}^{|N(a)|} s^{(k-1)}\left(N_i(a), N_j(b)\right), & N(a), N(b) \neq \varnothing; \\ 0, & otherwise. \end{cases} \tag{3}$$

And the sequence $s^{(k)}$ nondecreasingly converges pointwise to a unique fixed point s, i.e.,

$$s(a,b) = \lim_{k \to +\infty} s^{(k)}(a,b) \quad \forall (a,b) \in V^2. \tag{4}$$

In matrix forms [7], Eq.(2) and Eq.(3) can be written as

$$\begin{cases} \mathbf{S}^{(0)} = \mathbf{I}_n \\ \mathbf{S}^{(k)} = \left(c \cdot \mathbf{Q} \cdot \mathbf{S}^{(k-1)} \cdot \mathbf{Q}^T\right) \bigvee \mathbf{I}_n \quad (\forall k = 1, 2, \cdots) \end{cases} \tag{5}$$

where $\mathbf{S}^{(k)}$ is the k-th iterative SimRank matrix whose entry $s_{i,j}^{(k)}$ denotes the SimRank similarity score between documents i and j at the k-th iteration.

2.2 Symbol Definitions

In Table 1, we list the notations used throughout this paper. Note that symbols defined and referenced in a local context are not listed here.

Table 1. Symbols & Notations

symbol	definition	symbol	definition
$\mathbf{Q}$	transition probability matrix of $\mathcal{G}$	c	decay factor, $0 < c < 1$
$\mathbf{S}$	SimRank matrix	n	number of vertices on $\mathcal{G}$
$\mathbf{I_n}$	$n \times n$ identity matrix	m	number of edges on $\mathcal{G}$
K	number of iterations	d	average vertex degree of $\mathcal{G}$
$\bigvee$	disjunction operator	ϵ	accuracy

3 AUG-SimRank Algorithm

In this section, we show an efficient algorithm to optimize the iterative SimRank computation over undirected graphs. We first formally introduce the notion of a graph spectrum in our paper[1].

Definition 2 (web graph spectrum). *Given a web graph $\mathcal{G}$, let $\mathbf{Q}_\mathcal{G}$ denote its transition probability matrix (i.e. the transpose of a column-normalized adjacency matrix). The spectrum of a web graph $\mathcal{G}$ is defined to be the set of the eigenvalues of $\mathbf{Q}_\mathcal{G}$. In symbols,*

$$\sigma(\mathcal{G}) := \{\lambda \in \mathbb{C} |\, \mathbf{Q}_\mathcal{G} \cdot \mathbf{u} = \lambda \cdot \mathbf{u}, \quad \forall \mathbf{u} \neq \mathbf{0}\}$$

Generally, for an arbitrary graph $\mathcal{G}$, the elements in $\sigma(\mathcal{G})$, given by the roots of the characteristic polynomial of the matrix $\mathbf{Q}_\mathcal{G}$, might be *complex* numbers. However, for *undirected* graphs, due to its *symmetric* adjacency matrix, we have the following special property of the spectrum.

Theorem 1. *Given an undirected graph $\mathcal{G}$, all the eigenvalues of its transition probability matrix $\mathbf{Q}_\mathcal{G}$ are real numbers associated with a complete set of orthonormal eigenvectors.*

Proof. Recall from linear algebra that an elementary row transformation on any row of $\mathbf{Q}$ corresponds to a row-multiplying operation on $\mathbf{Q}$ [11]. Therefore, $\mathbf{Q}_\mathcal{G}$ can be denoted as $\mathbf{Q} = \mathbf{\Lambda} \cdot \mathbf{P}$, where $\mathbf{\Lambda}$ is the real diagonal matrix normalizing $\mathbf{P}$, and $\mathbf{P}$ is the real symmetric adjacency matrix of $\mathcal{G}$.

Then we assume that $\mathbf{Q}$ contains complex eigenvalues. Let $\lambda_\mathbf{\Lambda}, \lambda_\mathbf{P}, \lambda_\mathbf{Q} \in \mathbb{C}$ be the eigenvalues of $\mathbf{\Lambda}$,$\mathbf{P}$ and $\mathbf{Q}$ respectively. We need to show that $\lambda_\mathbf{Q} = \overline{\lambda_\mathbf{Q}}$ for all $\lambda_\mathbf{Q} \in \sigma(\mathcal{G})$. By definition, we have

$$\begin{cases} \mathbf{\Lambda} \cdot \mathbf{u} = \lambda_\mathbf{\Lambda} \cdot \mathbf{u} \\ \mathbf{P} \cdot \mathbf{u} = \lambda_\mathbf{P} \cdot \mathbf{u} \end{cases} \quad (\forall \mathbf{u} \neq \mathbf{0})$$

[1] The terminology in this paper is slightly different from the traditional spectral radius of a web graph which is referred to as the dominant eigenvector of a graph *adjacency* matrix.

Taking complex conjugate of both sides, and noticing that $\mathbf{\Lambda} = \overline{\mathbf{\Lambda}^T}, \mathbf{P} = \overline{\mathbf{P}^T}$, we obtain

$$\begin{cases} \overline{\mathbf{u}^T} \cdot \mathbf{\Lambda} = \overline{\mathbf{u}^T} \cdot \overline{\lambda_{\mathbf{\Lambda}}} \\ \overline{\mathbf{u}^T} \cdot \mathbf{P} = \overline{\mathbf{u}^T} \cdot \overline{\lambda_{\mathbf{P}}} \end{cases} \quad (\forall \mathbf{u} \neq \mathbf{0})$$

Hence,

$$\overline{\mathbf{u}^T} \cdot \mathbf{Q} \cdot \mathbf{u} = \overline{\mathbf{u}^T} \cdot \mathbf{\Lambda} \cdot \underbrace{(\mathbf{P} \cdot \mathbf{u})}_{=\lambda_{\mathbf{P}} \cdot \mathbf{u}} = \lambda_{\mathbf{P}} \cdot \overline{\mathbf{u}^T} \cdot \underbrace{(\mathbf{\Lambda} \cdot \mathbf{u})}_{=\lambda_{\mathbf{\Lambda}} \cdot \mathbf{u}} = \lambda_{\mathbf{P}} \cdot \lambda_{\mathbf{\Lambda}} \cdot \left(\overline{\mathbf{u}^T} \cdot \mathbf{u}\right) = \lambda_{\mathbf{Q}} \cdot \|\mathbf{u}\|^2 \quad (\forall \mathbf{u} \neq \mathbf{0})$$

Also,

$$\overline{\mathbf{u}^T} \cdot \mathbf{Q} \cdot \mathbf{u} = \underbrace{\left(\overline{\mathbf{u}^T} \cdot \mathbf{\Lambda}\right)}_{=\overline{\mathbf{u}^T} \cdot \overline{\lambda_{\mathbf{\Lambda}}}} \cdot \mathbf{P} \cdot \mathbf{u} = \overline{\lambda_{\mathbf{\Lambda}}} \cdot \underbrace{\left(\overline{\mathbf{u}^T} \cdot \mathbf{P}\right)}_{=\overline{\mathbf{u}^T} \cdot \overline{\lambda_{\mathbf{P}}}} \cdot \mathbf{u} = \overline{\lambda_{\mathbf{\Lambda}}} \cdot \overline{\lambda_{\mathbf{P}}} \cdot \left(\overline{\mathbf{u}^T} \cdot \mathbf{u}\right) = \overline{\lambda_{\mathbf{Q}}} \cdot \|\mathbf{u}\|^2 \quad (\forall \mathbf{u} \neq \mathbf{0})$$

Thus,

$$\lambda_{\mathbf{Q}} \cdot \|\mathbf{u}\|^2 = \overline{\lambda_{\mathbf{Q}}} \cdot \|\mathbf{u}\|^2 \quad (\forall \mathbf{u} \neq \mathbf{0}) \quad \Rightarrow \quad \lambda_{\mathbf{Q}} = \overline{\lambda_{\mathbf{Q}}}$$

Therefore, for an undirected graph, every eigenvalue of the transition probability matrix $\mathbf{Q}_{\mathcal{G}}$ is a real number.

Additionally, the characteristic polynomial $\det(\mathbf{Q} - \lambda_{\mathbf{Q}} \cdot \mathbf{I}_n)$ has n roots, which implies that there exists a *complete* spectral decomposition for undirected graphs. □

Theorem 1 presents a sufficient condition that $\mathbf{Q}_{\mathcal{G}}$ has a complete real spectral decomposition, which can be applied to optimize SimRank iterative similarity computation over undirected graphs as follows.

Theorem 2. *For an undirected graph $\mathcal{G}$, let $\mathbf{Q} = \mathbf{U} \cdot \mathbf{\Lambda} \cdot \mathbf{U}^{-1}$ be a complete spectral decomposition of its transition probability matrix $\mathbf{Q}$, where $\mathbf{U}$ is an orthogonal matrix with real entities whose columns are eigenvectors of $\mathbf{Q}$, and $\mathbf{\Lambda}$ is a real diagonal matrix whose diagonal entities give the corresponding eigenvalues. Then we can construct the following iteration:*

$$\tilde{\mathbf{S}}_k = \begin{cases} \mathbf{I}_n & k = 0 \\ \left(\left[c \cdot diag(\mathbf{\Lambda}) \cdot diag(\mathbf{\Lambda})^T\right] \odot \tilde{\mathbf{S}}_{k-1}\right) \vee \mathbf{I}_n, & k = 1, 2, \cdots \end{cases} \tag{6}$$

where $diag(\mathbf{\Lambda})$ is a column vector whose elements are the main diagonal of $\mathbf{\Lambda}$, and $\odot$ is an element-wise matrix multiplication operator[2]*.*
And SimRank similarity can be thereby obtained as follows:

$$\mathbf{S}_k = \mathbf{U} \cdot \tilde{\mathbf{S}}_k \cdot \mathbf{U}^{-1} \tag{7}$$

[2] More explicitly, for $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times m}$, then

$$\mathbf{A} \odot \mathbf{B} := \begin{pmatrix} a_{11}b_{11} & \dots & a_{1m}b_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1}b_{n1} & \cdots & a_{nm}b_{nm} \end{pmatrix} \in \mathbb{R}^{n \times m}$$

Proof. When $k > 0$, we substitute $\mathbf{Q} = \mathbf{U} \cdot \mathbf{\Lambda} \cdot \mathbf{U}^{-1}$ back into the naive SimRank matrix equation (5) to get

$$\begin{aligned} \mathbf{S}_k &= \left(c \cdot \mathbf{Q} \cdot \mathbf{S}_{k-1} \cdot \mathbf{Q}^T\right) \vee \mathbf{I}_n \\ &= \left(c \cdot \left(\mathbf{U} \cdot \mathbf{\Lambda} \cdot \mathbf{U}^{-1}\right) \cdot \mathbf{S}_{k-1} \cdot \left(\mathbf{U} \cdot \mathbf{\Lambda} \cdot \mathbf{U}^{-1}\right)^T\right) \vee \mathbf{I}_n \end{aligned} \tag{8}$$

Theorem 1 guarantees that the decomposition $\mathbf{Q} = \mathbf{U} \cdot \mathbf{\Lambda} \cdot \mathbf{U}^{-1}$ is complete, which implies that $\mathbf{U}$ and $\mathbf{\Lambda}$ are nonsingular. Now we premultiply and postmultiply both sides of Eq.(8), respectively, by $\mathbf{U}^{-1}$ and $\mathbf{U}$ to produce

$$\mathbf{U}^{-1} \cdot \mathbf{S}_k \cdot \mathbf{U} = (c \cdot \underbrace{\mathbf{U}^{-1} \cdot \mathbf{U}}_{=\mathbf{I}_n} \cdot \mathbf{\Lambda} \cdot \mathbf{U}^{-1} \cdot \mathbf{S}_{k-1} \cdot \mathbf{U}^{-T} \cdot \mathbf{\Lambda} \cdot \underbrace{\mathbf{U}^T \cdot \mathbf{U}}_{=\mathbf{I}_n}) \vee \underbrace{\left(\mathbf{U}^{-1} \cdot \mathbf{I}_n \cdot \mathbf{U}\right)}_{=\mathbf{I}_n}$$

Applying $\mathbf{U}^{-1} \cdot \mathbf{U} = \mathbf{I}_n$ and $\mathbf{U}^{-T} = \mathbf{U}$ to the above equation yields

$$\Rightarrow \quad \underbrace{\mathbf{U}^{-1} \cdot \mathbf{S}_k \cdot \mathbf{U}}_{:=\tilde{\mathbf{S}}_k} = (c \cdot \mathbf{\Lambda} \cdot \underbrace{\mathbf{U}^{-1} \cdot \mathbf{S}_{k-1} \cdot \mathbf{U}}_{:=\tilde{\mathbf{S}}_{k-1}} \cdot \mathbf{\Lambda}) \vee \mathbf{I}_n$$

We may assume that $\tilde{\mathbf{S}}_k := \mathbf{U}^{-1} \cdot \mathbf{S}_k \cdot \mathbf{U}$, and obtain

$$\begin{cases} \mathbf{S}_k = \mathbf{U} \cdot \tilde{\mathbf{S}}_k \cdot \mathbf{U}^{-1} \\ \tilde{\mathbf{S}}_k = \left(c \cdot \mathbf{\Lambda} \cdot \tilde{\mathbf{S}}_{k-1} \cdot \mathbf{\Lambda}\right) \vee \mathbf{I}_n \end{cases} \quad (k = 1, 2, 3, \cdots) \tag{9}$$

When $k = 0$, it follows from our assumption $\tilde{\mathbf{S}}_k := \mathbf{U}^{-1} \cdot \mathbf{S}_k \cdot \mathbf{U}$ that

$$\tilde{\mathbf{S}}_0 = \mathbf{U}^{-1} \cdot \mathbf{S}_0 \cdot \mathbf{U} = \mathbf{U}^{-1} \cdot \mathbf{I}_n \cdot \mathbf{U} = \mathbf{I}_n$$

In the following, we show that $\mathbf{\Lambda} \cdot \tilde{\mathbf{S}} \cdot \mathbf{\Lambda} = \left[diag\left(\mathbf{\Lambda}\right) \cdot diag(\mathbf{\Lambda})^T\right] \odot \tilde{\mathbf{S}}$, which is our trick to reduce the time complexity from $O\left(n^3\right)$ to $O\left(n^2\right)$ per iteration.

As all the off-diagonal entries of $\mathbf{\Lambda}$ are zeros, then

$$\begin{aligned} \mathbf{\Lambda} \cdot \tilde{\mathbf{S}} \cdot \mathbf{\Lambda} &= \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix} \cdot \begin{pmatrix} \tilde{s}_{11} & \tilde{s}_{12} & \dots & \tilde{s}_{1n} \\ \tilde{s}_{21} & \tilde{s}_{22} & \dots & \tilde{s}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{s}_{n1} & \tilde{s}_{n2} & \dots & \tilde{s}_{nn} \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix} \\ &= \begin{pmatrix} {\lambda_1}^2 \cdot \tilde{s}_{11} & \lambda_1 \lambda_2 \cdot \tilde{s}_{12} & \dots & \lambda_1 \lambda_n \cdot \tilde{s}_{1n} \\ \lambda_2 \lambda_1 \cdot \tilde{s}_{21} & {\lambda_2}^2 \cdot \tilde{s}_{22} & \dots & \lambda_2 \lambda_n \cdot \tilde{s}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_n \lambda_1 \cdot \tilde{s}_{n1} & \lambda_n \lambda_2 \cdot \tilde{s}_{n2} & \dots & {\lambda_n}^2 \cdot \tilde{s}_{nn} \end{pmatrix} \end{aligned}$$

On the other hand,

$$\left[diag\,(\mathbf{\Lambda})\cdot diag(\mathbf{\Lambda})^T\right]\odot\tilde{\mathbf{S}}=\left[\begin{pmatrix}\lambda_1\\ \lambda_2\\ \vdots\\ \lambda_n\end{pmatrix}\cdot(\lambda_1\ \lambda_2\ \cdots\ \lambda_n)\right]\odot\begin{pmatrix}\tilde{s}_{11} & \tilde{s}_{12} & \dots & \tilde{s}_{1n}\\ \tilde{s}_{21} & \tilde{s}_{22} & \dots & \tilde{s}_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ \tilde{s}_{n1} & \tilde{s}_{n2} & \dots & \tilde{s}_{nn}\end{pmatrix}$$

$$=\begin{pmatrix}\lambda_1{}^2 & \lambda_1\lambda_2 & \dots & \lambda_1\lambda_n\\ \lambda_2\lambda_1 & \lambda_2{}^2 & \dots & \lambda_2\lambda_n\\ \vdots & \vdots & \ddots & \vdots\\ \lambda_n\lambda_1 & \lambda_n\lambda_2 & \dots & \lambda_n{}^2\end{pmatrix}\odot\begin{pmatrix}\tilde{s}_{11} & \tilde{s}_{12} & \dots & \tilde{s}_{1n}\\ \tilde{s}_{21} & \tilde{s}_{22} & \dots & \tilde{s}_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ \tilde{s}_{n1} & \tilde{s}_{n2} & \dots & \tilde{s}_{nn}\end{pmatrix}=\begin{pmatrix}\lambda_1{}^2\cdot\tilde{s}_{11} & \lambda_1\lambda_2\cdot\tilde{s}_{12} & \dots & \lambda_1\lambda_n\cdot\tilde{s}_{1n}\\ \lambda_2\lambda_1\cdot\tilde{s}_{21} & \lambda_2{}^2\cdot\tilde{s}_{22} & \dots & \lambda_2\lambda_n\cdot\tilde{s}_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ \lambda_n\lambda_1\cdot\tilde{s}_{n1} & \lambda_n\lambda_2\cdot\tilde{s}_{n2} & \dots & \lambda_n{}^2\cdot\tilde{s}_{nn}\end{pmatrix}$$

Hence,

$$\mathbf{\Lambda}\cdot\tilde{\mathbf{S}}\cdot\mathbf{\Lambda}=\left[diag\,(\mathbf{\Lambda})\cdot diag(\mathbf{\Lambda})^T\right]\odot\tilde{\mathbf{S}}\tag{10}$$

Finally, we substitute Eq.(10) back into Eq.(9), which establishes Eq.(6) and completes the proof. □

Theorem 2 provides an efficient accelerative technique for SimRank computation based on the following algorithm.

Algorithm 1. AUG-SimRank: Accelerative SimRank for Undirected Graphs

```
Input  : adjacency matrix $\mathbf{P}=(p_{i,j})\in\mathbb{R}^{n\times n}$, decay factor $c$, accuracy $\epsilon$
Output: SimRank matrix $\mathbf{S}=(s_{i,j})\in[0,1]^{n\times n}$, iteration number $k$
1  begin
2      Calculate transition probability matrix $\mathbf{Q}$ according to $q_{i,j}\leftarrow p_{j,i}/\sum_{i=1}^{n}p_{j,i}$
3      Decompose $\mathbf{Q}\rightarrow\mathbf{U}\cdot\mathbf{\Lambda}\cdot\mathbf{U}^{-1}$, yielding $\mathbf{U}$, orthogonal, and $\mathbf{\Lambda}$, diagonal
4      Initialize $\tilde{\mathbf{S}}_0\leftarrow\mathbf{I}_n$, $\mathbf{M}\leftarrow c\cdot diag\,(\mathbf{\Lambda})\cdot diag(\mathbf{\Lambda})^T$, $k\leftarrow 0$
5      repeat                    /* each iteration takes $O(n^2)$ time */
6          Set $k\leftarrow k+1$
7          Update $\tilde{\mathbf{S}}_k\leftarrow\left(\mathbf{M}\odot\tilde{\mathbf{S}}_{k-1}\right)\vee\mathbf{I}_n$
8      until the auxiliary matrix $\tilde{\mathbf{S}}_k$ converges with an error of $\epsilon$
9      Calculate SimRank matrix $\mathbf{S}_k\leftarrow\mathbf{U}\cdot\tilde{\mathbf{S}}_k\cdot\mathbf{U}^{-1}$
10     Return $\mathbf{S}_k$, $k$
```

Now we describe the overall idea of Algorithm 1 which involves the following three steps.

(i) **Spectral Predecomposition (Line 2-3).** The first step in our algorithm is to diagonalize the transition probability matrix $\mathbf{Q}$ in Eq.(5) by eigen-decomposition, so that we may obtain a diagonal matrix $\mathbf{\Lambda}$ which makes it much easier in the next step to compute the power series of matrices derived from the recursive substitutions of Eq.(5). An important problem underlying this step is to determine whether there exists a *full* and *real* (not singular or complex) spectral decomposition for any arbitrary web graphs. We theoretically prove in Theorem 1 that only for *undirected* graphs, all the eigenvalues of the transition probability matrix $\mathbf{Q}$ are real numbers, and the number of eigenvalues is n. Hence, in this case, $\mathbf{Q}$ can be *completely* decomposed as $\mathbf{Q} = \mathbf{U} \cdot \mathbf{\Lambda} \cdot \mathbf{U}^{-1}$. The implementation of this step requires the well-known numerical methods given in [12,11].

(ii) **Iterative Element-wise Matrix Multiplication (Line 4-8).** In this step, we develop a novel iterative method to reduce the SimRank computational complexity. The key point of our optimization technique is based on Eq.(10), where $\mathbf{\Lambda}$ is diagonal, derived from the previous step, and $\tilde{\mathbf{S}}$ is an iterative auxiliary matrix with an initial value $\mathbf{I}_n$. Note that from Eq.(10) a standard matrix multiplication $\mathbf{\Lambda} \cdot \tilde{\mathbf{S}} \cdot \mathbf{\Lambda}$ can be accelerated significantly by performing the matrix product of a column vector and a row vector (i.e. a rank-1 tensor[3] $diag\,(\mathbf{\Lambda}) \cdot diag(\mathbf{\Lambda})^T$) first, and then multiplying this rank-1 matrix *element-wise* with $\tilde{\mathbf{S}}$. Due to the diagonal structure of $\mathbf{\Lambda}$, this computation only takes $O\left(n^2\right)$ time per iteration, whereas a standard implementation of $\mathbf{\Lambda} \cdot \tilde{\mathbf{S}} \cdot \mathbf{\Lambda}$ often requires $O\left(n^3\right)$ time. Moreover, to calculate $diag\,(\mathbf{\Lambda}) \cdot diag(\mathbf{\Lambda})^T$, preconditioning techniques may be adopted, which implies that once computed, this rank-1 matrix is memorized and is therefore not recomputed when subsequently required. Hence, in Line 4, $\mathbf{M}$ is precomputed only once, and can be used in this step for K iterations.

(iii) **SimRank Matrix Computation (Line 9).** In the final step, we obtain the SimRank matrix $\mathbf{S}_k$ from the k-th iterative auxiliary matrix $\tilde{\mathbf{S}}_k$ according to Eq.(7). This computation is performed only once, taking $O\left(n^3\right)$ time in the worst case.

Analyzing the computational complexity, we now summarize the above discussion in the following result.

Theorem 3. *For undirected graphs, Algorithm 1 can perform the SimRank computation for K iterations in $O\left(n^3 + K \cdot n^2\right)$ time in the worst case, where n is the number of vertices, and $n \gg K$.*

Proof. For Algorithm 1, Step (i) and (iii) are performed only once during the whole procedure, and their computational complexities involve $O\left(n^3\right)$ operations in the worst case. In Step (ii), due to the element-wise matrix products, the computational time required for K iterations is $O\left(K \cdot n^2\right)$. Hence, the total time required for K iterations of this algorithm is $O\left(n^3 + K \cdot n^2\right)$ in the worst case. □

4 Parallel AUG-SimRank Algorithm

In this section, we parallelize the AUG-SimRank algorithm which is highly efficient on distributed multiprocessors. To implement our parallel SimRank algorithm, we utilize

[3] Here, since $diag(\mathbf{\Lambda})^T$ is a row vector, we obtain

$$diag\,(\mathbf{\Lambda}) \cdot diag(\mathbf{\Lambda})^T = diag\,(\mathbf{\Lambda}) \cdot \begin{pmatrix}\lambda_1 & \cdots & \lambda_n\end{pmatrix} = \begin{pmatrix}\lambda_1 \cdot diag\,(\mathbf{\Lambda}) & \cdots & \lambda_n \cdot diag\,(\mathbf{\Lambda})\end{pmatrix}$$

Thus,

$$Rank\left[diag\,(\mathbf{\Lambda}) \cdot diag(\mathbf{\Lambda})^T\right] = 1$$

the available PLAPACK [13] (Parallel Linear Algebra Package) in combination with our matrix partition techniques on distributed memory architectures. PLAPACK is a parallel ARPACK version based on MPI (Message Passing Interface) for constructing parallel linear algebra libraries. It provides a high-level object-oriented programming interface so that coding of parallel linear algebra routines becomes a straightforward translation of algorithms.

We now show how to parallelize Algorithm 1 for each step.

In the *spectral predecomposition* step, we deploy a parallel solver that is capable of spectral factorization. Several works have studied parallel eigen-decomposition [14,13]. We adopt *a PLAPACK eigen-solver* in this step to decompose matrix $\mathbf{Q}$ due to its better performance exhibited for large-scale problems.

In the *iterative element-wise matrix multiplication* step, the parallel implementation of $\tilde{\mathbf{S}}_k = \left(\mathbf{M} \odot \tilde{\mathbf{S}}_{k-1}\right) \vee \mathbf{I}_n \quad (k = 1, 2, \cdots)$ is our main concern. We first vertically partition

$$\tilde{\mathbf{S}}_k = \left(\tilde{\mathbf{S}}_k^{(1)} \middle| \cdots \middle| \tilde{\mathbf{S}}_k^{(N)}\right), \quad \mathbf{M} = \left(\mathbf{M}^{(1)} \middle| \cdots \middle| \mathbf{M}^{(N)}\right), \quad \mathbf{I} = \left(\mathbf{I}^{(1)} \middle| \cdots \middle| \mathbf{I}^{(N)}\right),$$

where $\tilde{\mathbf{S}}_k^{(i)}, \mathbf{M}^{(i)}$, and $\mathbf{I}^{(i)}$ are $n \times n_i$ submatrices satisfying $n_i < n$ and $\sum_{i=1}^{N} n_i = n \quad (i = 1, 2, \cdots, N)$, and N is the number of partitions. Then, we have

$$\begin{aligned}
\left(\tilde{\mathbf{S}}_k^{(1)} \middle| \cdots \middle| \tilde{\mathbf{S}}_k^{(N)}\right) &= \left[\left(\mathbf{M}^{(1)} \middle| \cdots \middle| \mathbf{M}^{(N)}\right) \odot \left(\tilde{\mathbf{S}}_{k-1}^{(1)} \middle| \cdots \middle| \tilde{\mathbf{S}}_{k-1}^{(N)}\right)\right] \vee \left(\mathbf{I}^{(1)} \middle| \cdots \middle| \mathbf{I}^{(N)}\right) \\
&= \left(\mathbf{M}^{(1)} \odot \tilde{\mathbf{S}}_{k-1}^{(1)} \middle| \cdots \middle| \mathbf{M}^{(N)} \odot \tilde{\mathbf{S}}_{k-1}^{(N)}\right) \vee \left(\mathbf{I}^{(1)} \middle| \cdots \middle| \mathbf{I}^{(N)}\right) \\
&= \left(\left(\mathbf{M}^{(1)} \odot \tilde{\mathbf{S}}_{k-1}^{(1)}\right) \vee \mathbf{I}^{(1)} \middle| \cdots \middle| \left(\mathbf{M}^{(N)} \odot \tilde{\mathbf{S}}_{k-1}^{(N)}\right) \vee \mathbf{I}^{(N)}\right)
\end{aligned}$$

It follows that

$$\tilde{\mathbf{S}}_k^{(i)} = \left(\mathbf{M}^{(i)} \odot \tilde{\mathbf{S}}_{k-1}^{(i)}\right) \vee \mathbf{I}^{(i)} \quad (\forall i = 1, \cdots, N, \quad k = 1, 2, \cdots) \tag{11}$$

Note that in Eq.(11), there are no dependencies among the partitions, so they can all be executed in parallel. Observe that $\tilde{\mathbf{S}}_k$ and $\mathbf{M}$ are symmetric, and therefore we only need to compute the upper (or lower) triangular portion of these matrices.

In the *SimRank matrix computation* step, we focus our attention on the parallel computation of $\mathbf{S}_k \leftarrow \mathbf{U} \cdot \tilde{\mathbf{S}}_k \cdot \mathbf{U}^{-1}$. This operation can be performed by the following substeps:

(a) $\mathbf{V}_k \leftarrow \tilde{\mathbf{S}}_k \cdot \mathbf{U}^{-1}$ as a symmetric matrix-matrix multiplication.
(b) $\mathbf{S}_k \leftarrow \mathbf{U} \cdot \mathbf{V}_k$, which requires a matrix-matrix multiplication that only updates the upper (or lower) triangular part of $\mathbf{S}_k$.

Substep (a) is a standard matrix-matrix operation which can be parallelized in PLAPACK, whereas Substep (b) is not one of those existing widely used kernels.

We now apply matrix partitioning techniques to parallelize Substep (b). We partition the matrix $\mathbf{U}$ and $\mathbf{V}_k$ as

$$\mathbf{U} = \left(\mathbf{U}^{(1)} \middle| \cdots \middle| \mathbf{U}^{(N')}\right), \quad \mathbf{V}_k = \left(\begin{array}{c} \mathbf{V}_k^{(1)} \\ \hline \vdots \\ \hline \mathbf{V}_k^{(N')} \end{array}\right), \tag{12}$$

where $\mathbf{U}^{(i)}$ and $\mathbf{V}_k^{(i)}$ are $n \times n_i{}'$ and $n_i{}' \times n$ submatrices, respectively, satisfying $n_i{}' < n$ and $\sum_{i=1}^{N'} n_i{}' = n \quad (i = 1, 2, \cdots, N')$, and N' is the number of partitions. Then, we obtain

$$\mathbf{S}_k \leftarrow \mathbf{U} \cdot \mathbf{V}_k = \left(\mathbf{U}^{(1)} \middle| \cdots \middle| \mathbf{U}^{(N')} \right) \cdot \begin{pmatrix} \mathbf{V}_k^{(1)} \\ \hline \vdots \\ \hline \mathbf{V}_k^{(N')} \end{pmatrix} = \sum_{i=1}^{N'} \mathbf{U}^{(i)} \cdot \mathbf{V}_k^{(i)}$$

Hence, Substep (b) can be parallelized by updating the upper (or lower) triangular part of $\mathbf{S}_k \leftarrow \mathbf{S}_k + \mathbf{U}^{(i)} \cdot \mathbf{V}_k^{(i)} \quad (\forall i = 1, \cdots, N')$ for each loop.

We now summarize the above parallel techniques for SimRank computation. We present the following parallel SimRank algorithm that utilizes PLAPACK and matrix partitioning techniques.

Algorithm 2 provides a parallel version of SimRank computation over undirected web graphs. In Line 2 and 12, operations can be performed in parallel by calling the corresponding PLAPACK's routines respectively. In Line 4-10 and Line 14-15, each of

Algorithm 2. PAUG-SimRank: Parallel Accelerative SimRank for Undirected Graphs

```
   Input  : transition probability matrix $\mathbf{Q} = (q_{i,j}) \in \mathbb{R}^{n \times n}$,  decay factor $c$,  accuracy $\epsilon$
   Output: SimRank matrix $\mathbf{S} = (s_{i,j}) \in [0,1]^{n \times n}$
1  begin
       /* Step 1. Spectral Predecomposition */
2      Use a PLAPACK eigen-solver to decompose $\mathbf{Q} \rightarrow \mathbf{U} \cdot \mathbf{\Lambda} \cdot \mathbf{U}^{-1}$ in parallel
       /* Step 2. Iterative Elementwise Matrix Multiplication */
3      Partition the identity matrix $\mathbf{I}$ and the row vector $diag(\mathbf{\Lambda})^T$ as
       $\mathbf{I} \rightarrow \left( \mathbf{I}^{(1)} \middle| \mathbf{I}^{(2)} \middle| \cdots \middle| \mathbf{I}^{(N)} \right)$,   $diag(\mathbf{\Lambda})^T \rightarrow \left( \mathbf{\Lambda}^{(1)} \middle| \mathbf{\Lambda}^{(2)} \middle| \cdots \middle| \mathbf{\Lambda}^{(N)} \right)$
4      foreach partition $i \leftarrow 1 : N$ in parallel do
5          Initialize the upper triangular part of $\tilde{\mathbf{S}}_0^{(i)} \leftarrow \mathbf{I}^{(i)}$ and $\mathbf{M}^{(i)} \leftarrow c \cdot diag(\mathbf{\Lambda}) \cdot \mathbf{\Lambda}^{(i)}$
6          Fix the iteration number $k_i \leftarrow 0$
7          repeat
8              Set $k_i \leftarrow k_i + 1$
9              Calculate the upper triangular part of $\tilde{\mathbf{S}}_{k_i}^{(i)} \leftarrow \left( \mathbf{M}^{(i)} \odot \tilde{\mathbf{S}}_{k_i - 1}^{(i)} \right) \vee \mathbf{I}^{(i)}$
10         until this auxiliary partition $\tilde{\mathbf{S}}_{k_i}^{(i)}$ converges with an error of $\epsilon / N$
       /* Step 3. SimRank Matrix Computation */
11     Merge all the partitions of $\tilde{\mathbf{S}}$ into one matrix, i.e. $\tilde{\mathbf{S}} \leftarrow \left( \tilde{\mathbf{S}}_{k_1}^{(1)} \middle| \tilde{\mathbf{S}}_{k_2}^{(2)} \middle| \cdots \middle| \tilde{\mathbf{S}}_{k_N}^{(N)} \right)$
12     Use PLAPACK to operate $\mathbf{V} \leftarrow \tilde{\mathbf{S}} \cdot \mathbf{U}^{-1}$ as a symmetric matrix-matrix
       multiplication
13     Initialize $\mathbf{S} \leftarrow \mathbf{0}$ , and partition the matrix $\mathbf{U}$ and $\mathbf{V}$ as in Eq.(12)
14     foreach partition $i \leftarrow 1 : N'$ in parallel do
15         Calculate the upper triangular part of $\mathbf{S} \leftarrow \mathbf{S} + \mathbf{U}^{(i)} \cdot \mathbf{V}^{(i)}$
16     Return $\mathbf{S}$
```

the loops is decomposed into smaller tasks which can be executed simultaneously on the different processors of a parallel computer.

5 Experimental Studies

In this section, we conduct experimental studies to evaluate the efficiency of our algorithms (i.e. AUG-SimRank and PAUG-SimRank) on both synthetic and real data sets. We compare their performances with the most efficient existing algorithms proposed in [1] and [7]. The experiments were done on 2.0GHz Pentium(R) Dual-Core CPU with 2GB main memory, running Windows Vista OS. All the algorithms were implemented in C++ and compiled using Visual C++ 6.0 compiler.

5.1 Experimental Data Sets

We use two data sets for evaluation in our experiments.

Synthetic Data Sets. We randomly generated 10 undirected web graphs with vertices ranging from 1K to 10K respectively. For each generated graph, every vertex has an average of $\lceil \xi \rceil$ links, where ξ is a random variable satisfying $\xi \sim$*uniform*$[0, 15]$. The web graphs are represented in the MATLAB sparse storage organization, using a (x, y, val) triple to describe the index and the value of the element.

Real-life Data Sets. We use a full English Wikipedia data set exported in October 2007 to show the high efficiency of our proposed algorithms. The Wikipedia link graph contains 3.2M articles (vertices) with 110M intra-wiki links (edges). SimRank computation over the Wikipedia data set has the semantics of obtaining similarity scores for encyclopedic concept pairs. We build the Wikipedia graph, choosing the relationship "*a category contains an article*" to be an edge between the category and the article.

5.2 Comparison Methods and Evaluation

We compare the performances of the following algorithms:

- **SimRank with partial sums. [1]** This algorithm employs a partial sums function to speed up SimRank computation by SimRank values clustering.
- **SOR SimRank. [7]** This algorithm develops a successive over-relaxation method for accelerating the convergence rate of SimRank computation. We take the optimal relaxation factor $\omega = 1.3$.
- **AUG SimRank.** This is our proposed algorithm based on the graph spectral decomposition.
- **PAUG SimRank.** This is a parallel version of AUG SimRank that combines PLAPACK solvers and block partitioning techniques.

Evaluation Measures. We use *CPU time* and *absolute speedup* as two measures to evaluate the computational complexity and parallel efficiency of these algorithms respectively. The definition of absolute speedup is as follows [13]:

$$S_p := \frac{T_1}{T_p} \tag{13}$$

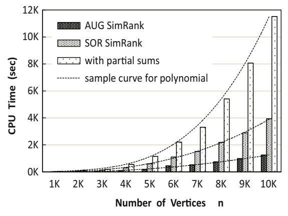
where p is the number of processors, T_1 is the execution time of the best sequential algorithm on one processor, and T_p is the time taken on p processors.

Parameter Settings. In the experiments, we set the decay factor $c = 0.8$, and the accuracy $\epsilon = 0.05$ (unless otherwise specified).

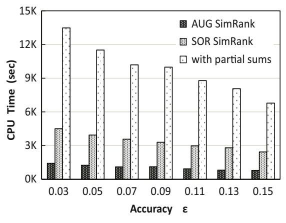
5.3 Time Efficiency Evaluation

Figure 1(a) shows the total computational time comparison among the three methods on 10 generated graphs when we set the vertices number $n = 1\text{K}, \cdots, 10\text{K}$. For each algorithm, a least-squares polynomial is used to fit the bar series that are nonlinear. It can be seen from the chart that AUG-SimRank outperforms the other two algorithms significantly as n is growing, and CPU time value by AUG-SimRank remains a steady increase around 1.5K. This is because the time complexity of AUG-SimRank has a constant cubic coefficient that is independent of the iteration number.

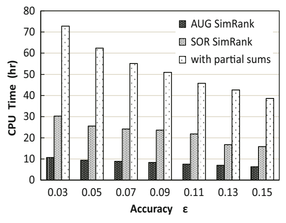
Figure 1(b) and 1(c) present the results of accuracy factor that influences the computational time for our three test algorithms on a 10K synthetic graph and a Wikipedia real data set, respectively. We set the decay factor $c = 0.6$ when testing results on Wikipedia data set. Note that different scale is chosen across the vertical axis in Figure 1(b) and 1(c) for providing a more illustrative look for each data set. We notice that both SOR-SimRank and SimRank with partial sums algorithms make a massive drop in CPU time while we are varying ϵ from 0.03 to 0.15, due to the more iterations needed for achieving a high accuracy. However, the trend of AUG-SimRank computational time shows



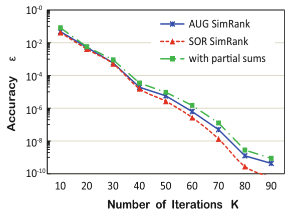
(a) CPU Time w.r.t. # of Vertices on Generated Graphs



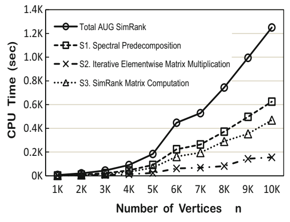
(b) CPU Time w.r.t. Accuracy on a Generated Graph (with n=10K)



(c) CPU Time w.r.t. Accuracy on Wikipedia (with c=0.6)



(d) Accuracy w.r.t. # of Iterations on Wikipedia (with c=0.6)



(e) AUG-SimRank Algorithm on Generated Graphs

Fig. 1. Scalability, Accuracy & Efficiency for AUG-SimRank Computation

a slight downward turn as ϵ is increasing, which also implies that AUG-SimRank is insensitive to the iteration number.

Figure 1(d) compares computational accuracy with respect to the number of iterations among the three algorithms on the Wikipedia data set. It shows that for all the algorithms, ϵ decreases dramatically while the number of iterations is increasing. The accuracy values by AUG-SimRank and SimRank with partial sums algorithms are close. This demonstrates that AUG-SimRank retains the convergence power of the SimRank with partial sums algorithm. For all of the algorithms, the figure also shows a logarithmic growth in CPU time when K is growing, since Figure 1(d) is in logarithm scale, and all these algorithms approximately exhibit linear descending tendencies.

Figure 1(e) precisely visualizes the CPU time consumption of each step for AUG-SimRank algorithm on a 10K generated data set. Specifically, we plot the computational time of (a) Spectral Predecomposition, (b) Iterative Elementwise Matrix Multiplication, and (c) SimRank Matrix Computation in the same chart. Note that the time consumed by Iterative Elementwise Matrix Multiplication is minimal and remains relatively stable, whereas the operation time for the other two steps rises substantially as n is increasing. This is because $K \ll n$ so that the step of Iterative Elementwise Matrix Multiplication takes quadratic time after K iterations, whereas the other two steps require cubic time, making a large portion of the total AUG-SimRank time.

5.4 Parallel Efficiency Evaluation

We test our parallel algorithm PAUG-SimRank on a distributed memory multiprocessor over synthetic and real-life data sets respectively. We obtain the following computational results.

Table 2. CPU Time (secs) for PAUG-SimRank

# of processors p	1	2	3	4	5	6	7	8	9	10
10K Generated Graph	1,250	814	559	397	356	343	297	274	255	222
Wikipedia Data Set	27,072	18,488	11,933	10,276	8,670	7,124	6,381	5,649	5,213	5,254

The parallel efficiency E_p of PAUG-SimRank can be computed by the following equation.

$$E_p := \frac{S_p}{p} = \frac{T_1}{T_p \cdot p}$$

where S_p is the speedup on p processors defined by Eq.(13). We use the execution time of the best sequential algorithm on one processor as T_1. We present the speedup and parallel efficiency of PAUG-SimRank algorithm in the tables and charts below.

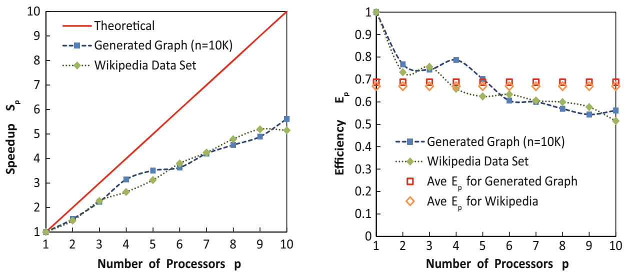
Figure 2(a) illustrates how the PAUG-SimRank implementation scales with the number of processors over a 10K generated graph and a Wikipedia data set, respectively. On both two data sets, the speedup is acceptable but not ideal due to the time required to communicate among the processors is growing. Note that there has been a dramatic increase in speedup when we have less than 4 processors. This is because of the cache effect resulting from the various memory hierarchies of a modern computer.

Table 3. PAUG-SimRank Efficiency on Generated Graph (with n=10K)

# of processors p	1	2	3	4	5	6	7	8	9	10
speedup S_p	1	1.53	2.23	3.14	3.51	3.63	4.20	4.55	4.89	5.61
efficiency E_p	1	0.76	0.74	0.78	0.70	0.60	0.60	0.56	0.54	0.56

Table 4. PAUG-SimRank Efficiency on Wikipedia

# of processors p	1	2	3	4	5	6	7	8	9	10
speedup S_p	1	1.46	2.26	2.63	3.12	3.80	4.24	4.79	5.19	5.15
efficiency E_p	1	0.73	0.75	0.65	0.62	0.63	0.60	0.59	0.57	0.51



(a) Absolute Speedup (b) Parallel Efficiency

Fig. 2. PAUG-SimRank on a generated graph (with n=10K) and a Wikipedia data set

Figure 2(b) visualizes the parallel efficiency of PAUG-SimRank on two data sets. Notice that the parallel efficiency makes a general decrease due to the same arguments as for the speedup above. For each data set, we obtain a mean of approx. 67% efficiency which has been shown in the chart with hollow markers.

6 Related Work

Hyperlink-based approaches for similarity computation on web graphs have become popular since the famous result of Page et. al. [15] on Google PageRank citation ranking. Since then, there has been a surge of papers aiming at various problems in the optimization of web ranking algorithms, such as HITS [16], ReCom [17], SimRank [2], SimFusion [18], P-Rank [19], etc. In particular, there has been significant attention to computing SimRank similarities as they provide a simple and intuitive graph-theoretic model recursively refining the cocitation measure that says "*similar objects are referenced by similar objects*". Recent SimRank results include randomized approximations [3,4], estimating a precise accuracy for iterative computation [1,5,7], graph clustering [20], query rewriting for sponsored search [6], etc.

Jeh and Widom [2] first proposed a SimRank model based on the idea that "*vertices are similar if their neighbors are similar*". Unfortunately the computational time of this model is very costly as it takes $O\left(Kn^2d^2\right)$ time, where n is the number of vertices, and d is the average degree of a web graph. Hence, the naive SimRank algorithm is not suitable for practical use.

Fogaras and Racz [3,4] adopted a *probabilistic* method called *Monte Carlo* to optimize SimRank computation. The main idea of their approach is to generate a finger print tree in combination of random permutations. They estimate $s\left(a,b\right)$ by $\mathbb{E}\left(c^{\tau(a,b)}\right)$, where $\tau_{(a,b)}$ is *the first meeting time* for a pair of random walks started in nodes a and b. Though this probabilistic model can significantly reduce the computational complexity, it does not predict *deterministic* quantities of SimRank scores but rather *stochastic* estimates of these quantities.

In comparison to the work on *stochastic* SimRank approximation, the work on optimizing *deterministic* SimRank computation is limited. Recent years have witnessed a growing interest in the optimization issue of *iterative* SimRank algorithms.

A very interesting piece of work is due to Lizorkin et al. [1] who give approaches to significantly reducing the computational time complexity from $O\left(Kn^2d^2\right)$ to $O\left(Kn^2d\right)$. Their study is not under the stochastic framework; rather, they use *a partial sums function* that allows clustering $s_k\left(*,*\right)$ by the first argument to accelerate access operations. They also give the number of iterations required for achieving the desired accuracy. Given an error of ϵ and a decay factor c, the upper bound of the total number of iterations is $K = \lceil \log_c \epsilon \rceil + 1$.

There has also been work on deterministic SimRank optimization. Antonellis et al. [6] present two SimRank extensions: one exploits the *weights* of the edges in the click graph, and the other that takes into account the "*evidence*" supporting the similarity between queries. Recent work by Yu et al. [7] shows a compressed storage scheme for sparse graphs to reduce the space requirement and a fast matrix multiplication for dense graph to further reduce the time complexity from $O\left(Kn^2d\right)$ to $O\left(K \cdot \min\left(m \cdot n, n^r\right)\right)$, where r is a constant, and $r \in (2, \log_2 7]$. They also developed optimization techniques for minimizing matrix bandwidths to improve the I/O efficiency of SimRank iteration. Li et al. [10] proposed a novel approximate SimRank computation algorithm for static and dynamic information networks. They claim that their optimization technique is based on the non-iterative framework; however, the singular value decomposition (SVD) method they used for low-rank approximation inherently requires numerical iterations. Hence, their method is in essence iterative, not non-iterative. The Kronecker product in their approach is prohibitively costly in computational time and therefore is not preferable. Li et al. [21] developed a BlockSimRank algorithm that partitions the web graph into several blocks to efficiently compute similarity of each node-pair in the graph. Their approach takes $O(n^{\frac{4}{3}})$ time, which is based on the random walk model. Zhao et al. [19] proposed a new structural similarity measure called P-Rank (Penetrating Rank) that says "two entities are similar if (a) they are referenced by similar entities; and (b) they reference similar entities." This similarity takes into account of both in- and out-link relationships of entity pairs and penetrates the structural similarity computation beyond neighborhood of vertices to the entire information network.

7 Conclusions

In this paper, we study the problem of optimizing SimRank computation on undirected graphs. An efficient AUG-SimRank algorithm has been proposed to significantly reduce the SimRank computational time from $O\left(K \cdot n^3\right)$ to $O\left(n^3 + K \cdot n^2\right)$ after K iterations in the worst case. We also present a parallel version of the AUG-SimRank algorithm on distributed memory multi-processors as we combine the PLAPACK solvers with our partition techniques. Extensive experimental evaluations on synthetic and real data sets show the proposed methods outperform the existing techniques in terms of total computation cost and parallelization.

Acknowledgment

We would like to thank the anonymous reviewers for their valuable comments. The second author was supported by three ARC DPs (DP0666428, DP0881035, and DP0987557) and a Google research award.

References

1. Lizorkin, D., Velikhov, P., Grinev, M., Turdakov, D.: Accuracy estimate and optimization techniques for simrank computation. PVLDB 1(1) (2008)
2. Jeh, G., Widom, J.: Simrank: a measure of structural-context similarity. In: KDD (2002)
3. Fogaras, D., Rácz, B.: Scaling link-based similarity search. In: WWW (2005)
4. Fogaras, D., Rácz, B.: A scalable randomized method to compute link-based similarity rank on the web graph. In: Lindner, W., Mesiti, M., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) EDBT 2004. LNCS, vol. 3268, pp. 557–567. Springer, Heidelberg (2004)
5. Cai, Y., Li, P., Liu, H., He, J., Du, X.: S-simrank: Combining content and link information to cluster papers effectively and efficiently. In: Tang, C., Ling, C.X., Zhou, X., Cercone, N.J., Li, X. (eds.) ADMA 2008. LNCS (LNAI), vol. 5139, pp. 317–329. Springer, Heidelberg (2008)
6. Antonellis, I., Garcia-Molina, H., Chang, C.C.: Simrank++: query rewriting through link analysis of the click graph. PVLDB 1(1) (2008)
7. Yu, W., Lin, X., Le, J.: A space and time efficient algorithm for simrank computation. In: APWeb (2010)
8. Weinberg, B.H.: Bibliographic coupling: A review. Information Storage and Retrieval 10(5-6) (1974)
9. Wijaya, D.T., Bressan, S.: Clustering web documents using co-citation, coupling, incoming, and outgoing hyperlinks: a comparative performance analysis of algorithms. IJWIS 2(2) (2006)
10. Li, C., Han, J., He, G., Jin, X., Sun, Y., Yu, Y., Wu, T.: Fast computation of simrank for static and dynamic information networks. In: EDBT (2010)
11. Bhatia, R.: Matrix Analysis. Springer, Heidelberg (1997)
12. Hernandez, V., Roman, J.E., Vidal, V.: Slepc: A scalable and flexible toolkit for the solution of eigenvalue problems. ACM Trans. Math. Softw. 31(3) (2005)
13. Maschhoff, K.J., Sorensen, D.C.: A portable implementation of arpack for distributed memory parallel architectures. In: CMCIM (1996)

14. Wu, K., Simon, H.: A parallel lanczos method for symmetric generalized eigenvalue problems. Technical report, Lawrence Berkeley National Laboratory (1997)
15. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking bringing order to the web, Technial report (1998)
16. Mendelzon, A.O.: Review - authoritative sources in a hyperlinked environment. ACM SIGMOD Digital Review 1 (2000)
17. Wang, J., Zeng, H., Chen, Z., Lu, H., Tao, L., Ma, W.: Recom: reinforcement clustering of multi-type interrelated data objects. In: SIGIR (2003)
18. Xi, W., Fox, E.A., Fan, W., Zhang, B., Chen, Z., Yan, J., Zhuang, D.: Simfusion: measuring similarity using unified relationship matrix. In: SIGIR (2005)
19. Zhao, P., Han, J., Sun, Y.: P-rank: a comprehensive structural similarity measure over information networks. In: CIKM 2009: Proceeding of the 18th ACM conference on Information and knowledge management (2009)
20. Zhou, Y., Cheng, H., Yu, J.X.: Graph clustering based on structural/attribute similarities. PVLDB 2(1) (2009)
21. Li, P., Cai, Y., Liu, H., He, J., Du, X.: Exploiting the block structure of link graph for efficient similarity computation. In: Theeramunkong, T., Kijsirikul, B., Cercone, N., Ho, T.-B. (eds.) PAKDD 2009. LNCS, vol. 5476, pp. 389–400. Springer, Heidelberg (2009)