Noname manuscript No.
(will be inserted by the editor)

A Space and Time Efficient Algorithm for SimRank Computation

Weiren Yu · Wenjie Zhang · Xuemin Lin · Qing Zhang · Jiajin Le

Received: date / Accepted: date

Abstract SimRank has become an important similarity measure to rank web documents based on a graph model on hyperlinks. The existing approaches for conducting SimRank computation adopt an iteration paradigm. The most efficient deterministic technique yields $O\left(n^3\right)$ worst-case time per iteration with the space requirement $O\left(n^2\right)$, where n is the number of nodes (web documents). In this paper, we propose novel optimization techniques such that each iteration takes $O\left(\min\left\{n \cdot m, n^r\right\}\right)$ time and $O\left(n+m\right)$ space, where m is the number of edges in a web-graph model and $r \leq \log_2 7$. In addition, we extend the similarity transition matrix to prevent random surfers getting stuck, and devise a pruning technique to eliminate impractical similarities for each iteration. Moreover, we also develop a reordering technique combined with an over-relaxation method, not only speeding up the convergence rate of the existing techniques, but achieving I/O efficiency as well. We conduct extensive experiments on both synthetic and real data sets to demonstrate the efficiency and effectiveness of our iteration techniques.

Keywords Graph Similarity · SimRank · Link-based Analysis · Optimal Algorithms

1 Introduction

In recent years, the complex hyperlink-based similarity search has attracted considerable attention in the field of Information Retrieval. One of the promising measures is the Sim-

The work was supported by ARC Grants DP0987557 and DP0881035 and Google Research Award.

Weiren Yu · Wenjie Zhang · Xuemin Lin
School of Computer Science & Engineering
University of New South Wales & NICTA, Sydney, NSW 2052, Australia
E-mail: {weirenyu,zhangw,lxue}@cse.unsw.edu.au

Qing Zhang
E-Health Research Center
Australia CSIRO ICT Center, Australia
E-mail: qing.zhang@csiro.au

Jiajin Le
School of Computer Science & Technology
Donghua University, Shanghai, China
E-mail: lejiajin@dhu.edu.cn

Rank similarity with ubiquitous applications to search engine ranking, document corpora clustering, collaborative filtering for recommender systems, etc. SimRank is a recursive refinement of co-citation measure that computes similarity by common neighbours alone [1]. The intuitive model for SimRank measure is based on random walk over a web-graph like Google PageRank [2]. The SimRank similarity between two web pages is defined recursively as the average similarity between their neighbours, along with the base case that one page is maximally similar to itself. Unlike many other domain-specific measures that require human-built hierarchies, SimRank can be used in any domain in combination with traditional textual similarity to produce an overall similarity measure [3,4].

Motivations: For achieving high efficiency of SimRank computation, it is desirable to develop optimization techniques that improve the time and space complexity of the similarity algorithm. The state-of-the-art approaches for SimRank optimization can be outlined into the following two categories: (a) *deterministic* techniques [5–9] that estimate the exact SimRank solution $s(\cdot,\cdot)$ (i.e., a fixed point) by utilizing an iterated similarity function. (b) *stochastic* techniques [3,4] that approximate similarities by the expected value $s(a,b) = \mathbb{E}\left(c^{\tau(a,b)}\right)$, where $\tau_{(a,b)}$ is a random variable denoting the first hitting time for nodes a and b, and $c \in (0,1)$ is a decay factor.

Several deterministic strategies have been studied as the iterative algorithms often produce better accuracy than stochastic counterparts for similarity estimation. The straightforward SimRank iterative paradigm is time-consuming, yielding $O\left(Kn^4\right)$ worst-case time and $O\left(n^2\right)$ space [1]. Recent work on structural/attribute-similarity-based graph clustering [10] has also raised the need for developing efficient methods for similarity computation. Nonetheless, unlike content-based similarities, it is considerably more challenging to design a deterministic and efficient algorithm that allows significant savings in time and space for SimRank computation.

To the best of our knowledge, to the present there are three most efficient techniques [6,11,12] for deterministic SimRank computation. Lizorkin et al. [5,6] deployed *a partial sums function* reducing the computational time from $O\left(Kn^4\right)$ to $O\left(Kn^3\right)$ in the worst case. Li et al. [11] developed an approximation algorithm for estimating similarities on static and *dynamic* information networks. He et al. [12] proposed a parallel algorithm based on *graphics processing units* (GPU) for similarity computation.

Contributions: In this extended paper of the conference version [13], we present the additional optimization techniques to further improve the efficiency of SimRank deterministic computation by orders of magnitude. We outline the main contributions made in this paper as follows:

- We extend the SimRank transition matrix in [13] to avoid locking random surfer into an enclosed subsection of the entire graph by overlaying a teleportation matrix.
- We represent similarity equations in a matrix form and employ a sparse storage scheme for SimRank transition matrix, which lays a foundation for optimizing computational time from $O\left(n^3\right)$ to $O\left(\min\left\{n \cdot m, n^r\right\}\right)$ in the worst case, where $r \leq \log_2 7$ with the space requirement from $O\left(n^2\right)$ to $O\left(m+n\right)$.
- We devise a pruning technique for the similarity matrix to eliminate impractical almost zero similarity values by setting a threshold, keeping the sparseness of similarity matrix for each iteration.
- We develop a heuristic optimal permutation technique for minimizing SimRank transition matrix bandwidth for digraphs, improving the I/O efficiency of SimRank iteration.
- We show a successive over-relaxation method for SimRank computation to significantly speed up the convergence rate of the existing technique.

- We validate the performance of our algorithm with an extensive experimental study on both synthetic and real-life data sets, demonstrating that it outperforms the state-of-the-art algorithms by orders of magnitude.

Organizations: The rest of the paper is organized as follows. In the next section, the problem definition for SimRank is formally introduced. In Sect. 3, a solution framework for SimRank optimization techniques is established. In Sect. 4, three optimization techniques for SimRank computation are suggested; the time and space complexity of the proposed algorithm is analyzed. In Sect. 5, the experimental results are reported on the efficiency of our methods over synthetic and real-life data sets. The related work appears in Sect. 6 and Sect. 7 concludes the paper.

2 Preliminaries

In this section, the formal definition of SimRank is given and some notations are presented. The material in this section recalls Jeh's previous work [1].

2.1 Problem Definition

Given a directed graph $\mathcal{G} = (V, E)$, each node in V represents a web page and a directed edge $\langle a, b\rangle$ in E corresponds to a hyperlink from page a to b, we can derive *a node-pair graph* $\mathcal{G}^2 \triangleq \left(V^2, E^2\right)$, where

(i) $\forall (a, b) \in V^2$ *if* $a, b \in V$;
(ii) $\forall \langle (a_1, b_1), (a_2, b_2)\rangle \in E^2$ *if* $\langle a_1, a_2\rangle, \langle b_1, b_2\rangle \in E$.

On a node-pair graph $\mathcal{G}^2$, we formally define a similarity function measured by SimRank score.

Definition 1 (SimRank similarity) Let $s : V^2 \to [0, 1] \subset \mathbb{R}$ be a real-valued function on $\mathcal{G}^2$ defined by

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

where $c \in (0, 1)$ is a constant decay factor , $I(a)$ denotes all in-neighbours of node a , $|I(a)|$ is the cardinality of $I(a)$, an individual member of $I(a)$ is referred to as $I_i(a)$ $(1 \le i \le |I(a)|)$, then $s(a, b)$ is called *SimRank similarity score between node* a *and* b.

The underlying intuition behind SimRank definition is that "*two pages are similar if they are referenced by similar pages*". Figure 1 visualizes the propagation of SimRank similarity in $\mathcal{G}^2$ from node to node, which corresponds to the propagation from pair to pair in $\mathcal{G}$, starting with the singleton node $\{4, 4\}$. Since a unique solution to the SimRank recursive equation (1) is reached by iteration to a fixed-point, we can carry out the following iteration for SimRank computation.

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

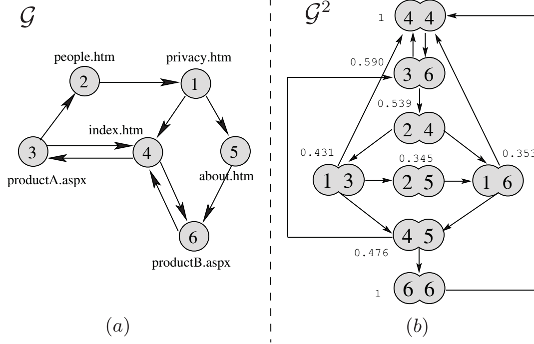


Fig. 1 SimRank propagating similarity from pair to pair in $\mathcal{G}$ associated with the propagation from node to node in $\mathcal{G}^2$ with a decay factor $c = 0.8$

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

where $s^{(k)}(a,b) \quad (\forall k = 0, 1, 2, \cdots)$ gives the score between a and b on the k-th iteration, and this sequence nondecreasingly converges to $s(a,b)$, i.e.,

$$s(a,b) = \lim_{k \to +\infty} s^{(k)}(a,b).$$

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

Table 1 SYMBOLS AND NOTATIONS

Symbol	Definition	Symbol	Definition
$\mathbf{P}$	adjacency matrix of $\mathcal{G}$	π	permutation function
$\mathbf{Q}$	transpose of column-normalized matrix $\mathbf{P}$	$\mathbf{\Theta}_\pi$	permutation matrix corresponding to π
$\mathbf{S}$	SimRank matrix	n	number of nodes on $\mathcal{G}$
$\mathbf{I}_n$	$n \times n$ identity matrix	m	number of edges on $\mathcal{G}$
K	number of iterations	d	average node degree of $\mathcal{G}$
$\bigvee$	disjunction operator	ϵ	accuracy
$\beta(\mathbf{Q})$	bandwidth of matrix $\mathbf{Q}$	c	decay factor, $0 < c < 1$
$N(a)$	set of neighbors of node a	$\lvert N(a) \rvert$	degree of node a

3 Solution Framework

In this section, we present our solution framework. The main optimization issue of SimRank computation covers the following three consecutive steps.

Firstly, a scheme for SimRank matrix representation is adopted. We introduce *a compressed storage scheme* for *sparse* graphs and *a fast matrix multiplication* for *dense* graphs

respectively, reducing the space requirement from $O\left(n^2\right)$ to $O\left(m+n\right)$ and the time complexity from $O\left(n^3\right)$ to $O\left(\min\left\{n\cdot m, n^r\right\}\right)$ in the worst case, where $r \le \log_2 7$. We show the results in Sect. 4.1.

Secondly, a technique for *permuted SimRank equation* is proposed. For the SimRank computation to be I/O-efficient, the adjacency matrix needs to be preordered, which requires off-line precomputation to minimize the bandwidth at query time. We discuss the approaches in detail in Sect. 4.2.

Finally, a method for *successive over-relaxation (SOR) iteration* is suggested to speed up the convergence rate of SimRank computation. We show that our SimRank iterative method is practically faster than the most efficient existing techniques [5]. We show theoretical results in Sect. 4.3.

4 Optimizations For SimRank Algorithms

In what follows, each of the three outlined techniques is presented in its own subsection accordingly.

4.1 Matrix Representations for SimRank Model

For an elaborate discussion on the subject, we first consider the SimRank similarity problem in matrix formulation. Let $\mathbf{S} = \left(s_{i,j}\right) \in \mathbb{R}^{n\times n}$ be a SimRank matrix of $\mathcal{G}$ whose entry $s_{i,j}$ equals the similarity score between page i and j, and $\mathbf{P} = \left(p_{i,j}\right) \in \mathbb{N}^{n\times n}$ be an adjacency matrix of $\mathcal{G}$ whose entry $p_{i,j}$ equals the number of edges from vertex i to j. Clearly, we can write (2) and (3) as

$$s_{a,b}^{(0)} = \begin{cases} 1, \ a = b; \\ 0, \ a \neq b. \end{cases} \tag{4}$$

$$\begin{aligned} s_{a,b}^{(k+1)} &= \frac{c}{\left|I\left(a\right)\right|\left|I\left(b\right)\right|} \sum_{i=1}^{n}\sum_{j=1}^{n} p_{i,a}\cdot s_{i,j}^{(k)} \cdot p_{j,b} \\ &= c\cdot \sum_{i=1}^{n}\sum_{j=1}^{n} \left(\frac{p_{i,a}}{\sum_{i=1}^{n} p_{i,a}}\right) \cdot s_{i,j}^{(k)} \cdot \left(\frac{p_{j,b}}{\sum_{j=1}^{n} p_{j,b}}\right) \end{aligned} \tag{5}$$

where we assume, without loss of generality, that $a \neq b$ (otherwise, $s_{a,a}^{(k)} \equiv 1 \ (k = 0, 1, \cdots)$).

In matrix notation, equations (4) and (5) become

$$\begin{cases} \mathbf{S}^{(0)} = \mathbf{I}_n \\ \mathbf{S}^{(k+1)} = \left(c\cdot \mathbf{Q}\cdot \mathbf{S}^{(k)} \cdot \mathbf{Q}^T\right) \bigvee \mathbf{I}_n \ \ (\forall k = 0, 1, \cdots) \end{cases} \tag{6}$$

As we have seen in equation (6), the computational complexity is $O\left(n^3\right)$ per iteration with the space requirement $O\left(n^2\right)$ since the naive matrix multiplication algorithm $u_{i,j} = \sum_{k=1}^{n} q_{i,k}\cdot s_{k,j} \quad (\forall i, j = 1, \cdots, n)$ performs $O\left(n^3\right)$ operations for all entries of $\mathbf{U} \in \mathbb{R}^{n\times n}$.

In the following, three techniques are investigated to obtain the time and space efficient algorithms for SimRank computation. We first give a slight modification of the naive SimRank transition matrix in order to avoid locking the random surfer into the pitfall on $\mathcal{G}^2$. Then, for sparse graph, the compressed storage scheme is adopted to reduce the space requirement to $O(n+m)$ with time complexity $O(n \cdot m)$. For dense graph, the fast matrix multiplication algorithm is suggested to reduce the time complexity from $O(n^3)$ to $O(n^r)$ in the worst case, where $r \leq \log_2 7$.

4.1.1 The modification of SimRank Transition Matrix

As SimRank similarity score $s(\cdot,\cdot)$ can be seen as a random walker defined on a node-pair graph $\mathcal{G}^2$ depicted in Fig.1 (b), the walker may wander into an enclosed subsection of the entire graph which has no out-link web documents so that he will get stuck in the small subgraph with no possibility to return to any outside web documents. The aforementioned scenario is associated with a reducible transition probability matrix, meaning that its corresponding graph is not strongly connected such that the walker on $\mathcal{G}^2$ might easily fall into the enclosed subsection.

To avoid locking the walker into the pitfall, we harness the technique termed *teleportation* on the reducible graph by overlaying *a teleportation graph* to make the new induced graph irreducible. We replace zero rows in the transition matrix $\mathbf{Q}$ with $\frac{1}{n}$ in all its entries. In matrix notations, this can be mathematically described as:

$$\tilde{\mathbf{Q}} = \lambda \cdot \mathbf{Q} + (1-\lambda) \cdot \frac{1}{n} \cdot \mathbf{E}_n, \tag{7}$$

where $\lambda \in [0,1]$ is a teleportation probability factor, and $\mathbf{E}_n$ is an $n \times n$ matrix with the i-th row being all ones if v_i has no out-links whatsoever.

It can be observed that the choice of value λ greatly influences the SimRank iterative result. A large λ would well describe that real-life potential link structure of the web graph, but it would result in a very slow convergence rate of the iteration. As a compromise, we empirically set $\lambda = 0.8$ for practical use.

In the following, we give a theoretical explanation of the above observation. Taking the vectorization operator (vec) on both sides of the equation (6) and applying the Kronecker property $vec\left(\mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T\right) = (\mathbf{Q} \otimes \mathbf{Q}) \cdot vec(\mathbf{S})$, we can obtain

$$\begin{cases} vec\left(\mathbf{S}^{(0)}\right) = vec(\mathbf{I}_n) \\ vec\left(\mathbf{S}^{(k+1)}\right) = c \cdot (\mathbf{Q} \otimes \mathbf{Q}) \cdot vec\left(\mathbf{S}^{(k)}\right) \vee vec(\mathbf{I}_n) \end{cases} \tag{8}$$

Note that the above equation is actually a variant of *power iteration* paradigm on $\mathcal{G}^2$. According to the power method [14], the convergence rate of $vec\left(\mathbf{S}^{(k)}\right)$ to the exact solution is geometrically governed by the ratio $O\left(\left|\frac{\lambda_2}{\lambda_1}\right|\right)$, where $\lambda_i \ (i=1,2)$ is the i-th eigenvalue of the matrix $c \cdot (\mathbf{Q} \otimes \mathbf{Q})$.

To determine the first (dominant) eigenvalue of $c \cdot (\mathbf{Q} \otimes \mathbf{Q})$, we need to compute the eigenvalue of $\mathbf{Q}$. Since $\mathbf{Q}$ is a row-stochastic matrix, it follows from the Perron-Frobenius theorem [14] that 1 is dominant eigenvalue of $\mathbf{Q}$. Hence, according to the Kronecker property [14], $c \cdot (\mathbf{Q} \otimes \mathbf{Q})$ has the eigenvalues $\{c, \lambda_2, \cdots\}$. Similarly, $c \cdot \left(\tilde{\mathbf{Q}} \otimes \tilde{\mathbf{Q}}\right)$ has the eigenvalues $\left\{c, \lambda^2 \cdot \lambda_2, \cdots\right\}$. Thus, when we replace $\mathbf{Q}$ with $\tilde{\mathbf{Q}}$ in the equation (8), the convergence rate of $vec\left(\mathbf{S}^{(k)}\right)$ will become $O\left(\frac{\lambda^2 \cdot |\lambda_2|}{c}\right)$, where λ is the teleportation factor of

(8), and λ_2 is the second largest eigenvalue of the matrix $c \cdot (\mathbf{Q} \otimes \mathbf{Q})$. Therefore, a value of large λ would lead to the slow convergence of $vec\left(\mathbf{S}^{(k)}\right)$.

4.1.2 Compressed Matrix Storage Scheme for Sparse Graph

For certain large scale web graphs, the relative sparseness of the adjacency matrix increases with the growth of the matrix dimension. To calculate SimRank for large domains, the memory requirements do not allow the adjacency matrix stored in its full format. Hence we suggest a *compressed sparse matrix representation* to be kept in main memory.

There are various compressed storage schemes to store a matrix [15], including *Compressed Sparse Row* (CSR), *Compressed Sparse Column* (CSC), *Jagged Diagonal* (JAD) *format*, etc. We use the CSR storage scheme for the sparse row-normalized adjacency matrix $\mathbf{Q}$ due to the high compression ratio. Observing that the directed graph $\mathcal{G}$ implies that $\mathbf{Q}$ is a non-symmetric sparse matrix, we construct a triple $\langle val,\ col_idx,\ row_ptr \rangle$, where val is a floating-point vector whose element stores the nonzero entry of the matrix $\mathbf{Q}$, col_idx is an integer vector whose element stores the column index of the nonzero entry in $\mathbf{Q}$ to make random jumps in the val vector, row_ptr is an integer vector whose element stores the location in the val vector that starts a row. Therefore we may infer from $val\,(k) = q_{i,j}$ that $col_idx\,(k) = j$ and $k \in [row_ptr\,(i)\,, row_ptr\,(i+1))$.

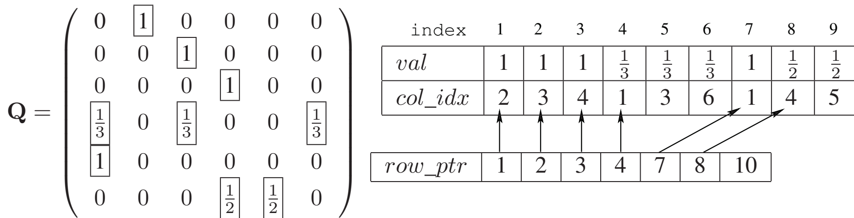


Fig. 2 CSR representation of the adjacency matrix $\mathbf{Q}$

In Figure 2, we give an illustrative example of CSR representation of the adjacency matrix $\mathbf{Q}$. And many basic mathematical operations on the sparse matrix such as matrix-matrix multiplication should be implemented in a new way. For our application, to calculate $\mathbf{U} = \mathbf{Q} \cdot \mathbf{S}$ in (6) , where $\mathbf{Q}$ is a sparse matrix and $\mathbf{S}$ is a dense matrix, we cannot use the sum $u_{i,j} = \sum_{k=1}^{n} q_{i,k} \cdot s_{k,j} \quad (\forall i, j = 1, \cdots, n)$ directly because the column traversal operation in CSR format matrix $\mathbf{Q}$ is costly. We adopt the following algorithm that is more efficient for sparse matrix multiplication [15].

In Algorithm 1, $\mathbf{Q}$ is stored in CSR format and the performance of matrix multiplication $\mathbf{Q}{\cdot}\mathbf{S}$ requires only $O\left(\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=row_ptr_{\mathbf{Q}}(j)}^{row_ptr_{\mathbf{Q}}(j+1)-1} 1\right) \equiv O\,(n \cdot m)$ time and $O\,(n+m)$ storage. If $\mathcal{G}$ is sparse, then $m = O\,(n)$. It follows that the complexity for computing the whole SimRank matrix $\mathbf{S}$ reduces to quadratic time and linear intermediate memory, which is a substantial improvement achieved by CSR storage schemes.

4.1.3 Fast Matrix Multiplication for Dense Graph

Even when the input graph is rather dense, we still consider that our algorithm is more time-efficient than the existing work [5]. Though in this case the naive dense matrix multiplication

Algorithm 1: SpMxDeM $(\mathbf{Q}, \mathbf{S})$

Input : sparse adjacency matrix $\mathbf{Q} = \langle val_{\mathbf{Q}},\ col_idx_{\mathbf{Q}},\ row_ptr_{\mathbf{Q}} \rangle \in \mathbb{R}^{n \times n}$,
dense SimRank matrix $\mathbf{S} = (s_{i,j}) \in \mathbb{R}^{n \times n}$
Output: dense matrix $\mathbf{U} = (u_{i,j}) \in \mathbb{R}^{n \times n}$

```
1 Initialize U ← 0
2 for i ← 1 : n do
3     for j ← 1 : n do
4         for k ← row_ptr_Q (j) : row_ptr_Q (j + 1) − 1 do
5             Calculate U (j, i) ← U (j, i) + val_Q (k) · S (col_idx_Q (k) , i)
6 Return U
```

requires $O\left(n^3\right)$ time complexity, fast matrix multiplication algorithms can be applied in our algorithms to speed up the computation of the dense matrices product. To the best of our knowledge, in standard matrix storage format, *the Coppersmith-Winograd algorithm* [16] is the fastest technique for square matrix multiplication, with a complexity of $O\left(n^{2.38}\right)$ which is a considerable improvement over the naive $O\left(n^3\right)$ time algorithm and the $O\left(n^{\log_2 7}\right)$ time *Strassen algorithm* [17]. The interested reader can refer to [18, 16, 17] for a detailed description. For our purpose, we implemented the Coppersmith-Winograd algorithm in dense graphs for achieving high performances of our algorithms. Therefore, combined with the sparse case, the time efficiency of our techniques is guaranteed with $O\left(\min\left\{n \cdot m, n^r\right\}\right)$ per iteration, where $r \leq \log_2 7$, much preferable to the existing approach [5] with a complexity of $O\left(n^3\right)$ in the worst case.

4.1.4 Pruning Similarity Matrix For Each Iteration

In many real applications, the similarity matrix often contains an extremely large fraction of non-zeros entries whose values are almost 0 after several iterations. These small similarity values require a significant amount of storage space with less practical information between web documents.

In order to keep the sparseness of similarity matrix, we develop a pruning technique to get rid of these almost zero values by setting a threshold Δ. If the similarity value $s_{i,j}^{(k)}$ between nodes i and j at k-th iteration is below Δ, we drop the corresponding this entry from the SimRank matrix. In symbols, we define the pruned similarity matrix by a threshold Δ at k-th iteration in the following:

$$\bar{s}_{i,j}^{(k)} = \begin{cases} s_{i,j}^{(k)}, & s_{i,j}^{(k)} \in [\Delta, 1] \\ 0, & s_{i,j}^{(k)} \in [0, \Delta) \end{cases} \tag{9}$$

This dropping will also decrease the redundant similarity computations and space per iteration. In practice, it is empirically preferable to set the threshold value $\Delta = 0.01$, which may improve the algorithmic efficiency by orders of magnitude.

4.2 Permuted SimRank Iterative Approach

After the CSR storage scheme has been created for the sparse adjacency matrix $\mathbf{Q}$, the optimization technique suggested in this subsection allows improving I/O efficiency for SimRank computation. The main idea behind this optimization involves two steps: (a) *Reversed*

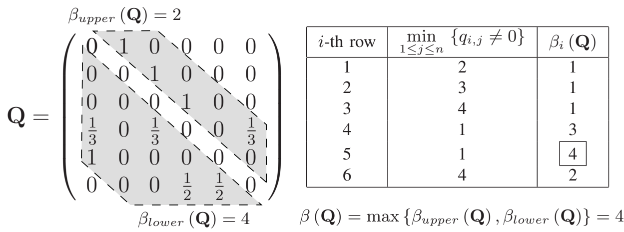
Cuthill-McKee(RCM) algorithm [19] for non-symmetric matrix is introduced for finding an optimal permutation π while reordering the matrix $\mathbf{Q}$ during the precomputation phase. (b) *Permuted SimRank iterative equation* is developed for reducing the matrix bandwidth for SimRank computation.

We first introduce the notion of *matrix bandwidth* [20].

Definition 2 (Matrix Bandwidth) Given a matrix $\mathbf{Q} = \left(q_{i,j}\right) \in \mathbb{R}^{n\times n}$, let $\beta_i\left(\mathbf{Q}\right) \triangleq \left| i - \min_{1\le j\le n} \left\{q_{i,j} \neq 0\right\}\right|$ denote the i-th bandwidth of matrix $\mathbf{Q}$. We define *the bandwidth of matrix* $\mathbf{Q}$ to be the quantity

$$\beta\left(\mathbf{Q}\right) \triangleq \max_{1\le i\le n} \beta_i\left(\mathbf{Q}\right)$$

If $\mathbf{Q}$ is non-symmetric, $\beta\left(\mathbf{Q}\right)$ is the maximum of its distinct upper and lower bandwidths $\beta_{upper}\left(\mathbf{Q}\right)$ and $\beta_{lower}\left(\mathbf{Q}\right)$. Figure 3 briefly illustrates an example of the above concept.



i-th row	$\min_{1\le j\le n} \{q_{i,j} \neq 0\}$	$\beta_i\left(\mathbf{Q}\right)$
1	2	1
2	3	1
3	4	1
4	1	3
5	1	4
6	4	2

Fig. 3 Bandwidth of the adjacency matrix $\mathbf{Q}$

A matrix bandwidth is introduced for measuring the I/O efficiency for SimRank computation. For achieving smaller bandwidth, we need to reorder the sparse matrix $\mathbf{Q}$ with precomputation by finding an optimal permutation π.

We now give the notions of *permutation* and *permutation matrix* which are helpful for further discussion [19].

Definition 3 (Permutation Matrix) Given a *permutation* π of n objects, $\pi : \{1, 2, \cdots, n\} \rightarrow \{1, 2, \cdots, n\}$ defined in two-line form by

$$\begin{pmatrix} 1 & 2 & \cdots & n \\ \pi\left(1\right) & \pi\left(2\right) & \cdots & \pi\left(n\right) \end{pmatrix}$$

The corresponding *permutation matrix* is $\mathbf{\Theta}_\pi = \left(\theta_{i,j}\right) \in \{0, 1\}^{n\times n}$, whose entries satisfy

$$\theta_{i,j} = \begin{cases} 1, & j = \pi\left(i\right); \\ 0, & otherwise. \end{cases}$$

One important property of a permutation matrix is that multiplying any matrix $\mathbf{Q}$ by a permutation matrix $\mathbf{\Theta}_\pi$ on the left/right has the same effect of rearranging the rows/columns of $\mathbf{Q}$. With this property, we may find an optimal permutation π while reordering the sparse matrix $\mathbf{Q}$ and can thus effectively minimize the bandwidth for SimRank computation.

4.2.1 Reversed Cuthill-McKee (RCM) algorithm for directed graph

The RCM algorithm for directed graph [19] is used for finding an optimal permutation π corresponding to $\mathbf{Q}$. With this permutation π, we can separate $\mathbf{Q}$ into dense blocks, store them individually in a CSR format and remove as many empty blocks as possible from $\mathbf{Q}$. However, it is an NP-complete problem [19] for finding such a permutation π, which may also be viewed as a web graph labeling problem in our models. We give an intuitive example in Figure 4.

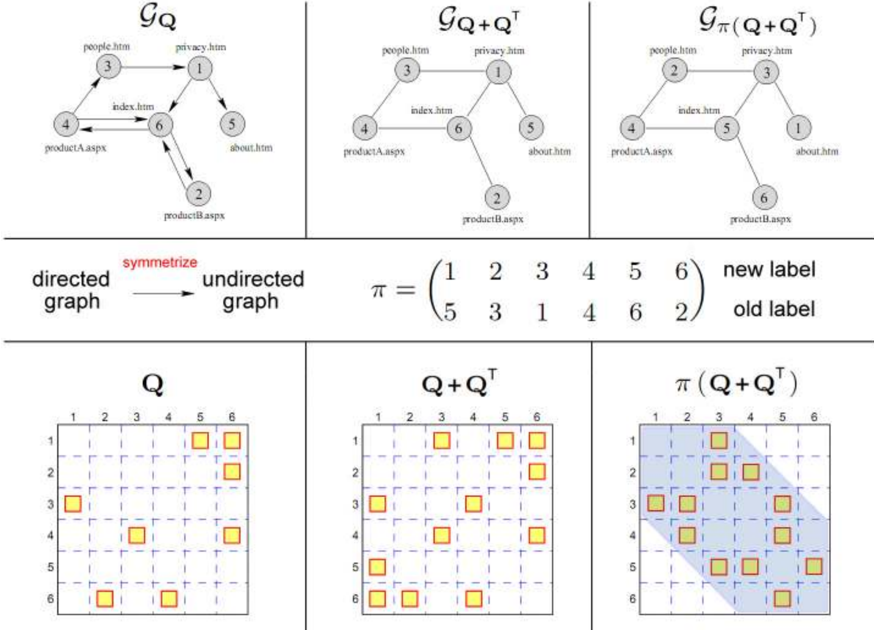


Fig. 4 A simplified version of SimRank minimum bandwidth ordering problem with permutation π on the adjacency matrix $\mathbf{Q}$

Figure 4 indicates that our permutation problem for adjacency matrix is equivalent to the graph labeling problem. It is easy to see that the graph $\mathcal{G}_{\mathbf{Q}+\mathbf{Q}^T}$ and $\mathcal{G}_{\pi(\mathbf{Q}+\mathbf{Q}^T)}$ have the identical structure and the different node labeling when we choose a new permutation π on both rows and columns of the matrix $\mathbf{Q}+\mathbf{Q}^T$. Thus, the permutation π can be thought of as a bijection between the vertices of the labeled graph $\mathcal{G}_{\mathbf{Q}+\mathbf{Q}^T}$ and $\mathcal{G}_{\pi(\mathbf{Q}+\mathbf{Q}^T)}$. And the bandwidth $\beta\left(\pi\left(\mathbf{Q}+\mathbf{Q}^T\right)\right)$ is often no greater than $\beta\left(\mathbf{Q}+\mathbf{Q}^T\right)$. In Figure 4, the SimRank bandwidth is optimized by permutation π as follows:

$$2=\beta\left(\pi\left(\mathbf{Q}\right)\right)=\beta\left(\pi\left(\mathbf{Q}+\mathbf{Q}^T\right)\right)<\beta\left(\mathbf{Q}+\mathbf{Q}^T\right)=\beta\left(\mathbf{Q}\right)=5$$

In the following, our goal is to find a better permutation π minimizing the bandwidth of the matrix $\mathbf{Q}$.

There have been several heuristic approaches available for determining the better permutation π for the given matrix $\mathbf{Q}$. Observe that the popular *Reversed Cuthill-McKee(RCM) algorithm* [19] is most widely used for ordering sparse *symmetric* matrices. We extend the original RCM to the directed graph associated with the *non-symmetric* adjacency matrix $\mathbf{Q}$. We reorder the rows of $\mathbf{Q}$ by adding "*the mate* $\mathbf{Q}^T$" of each entry and applying RCM to $\mathbf{Q}+\mathbf{Q}^T$ whose structure is symmetric since the bandwidth of $\pi(\mathbf{Q})$ is no greater than that of $\pi\left(\mathbf{Q}+\mathbf{Q}^T\right)$. We describe Algorithm 2 in high-level terms for finding the optimal permutation π, which is essentially an extenstion of RCM algorithm [19].

Algorithm 2: Extended_RCM $(\mathcal{G})$

```
    Input  : web graph G(V, E)
    Output: permutation array Π'
 1  begin
        /* 1. Initialization                                              */
 2      T.empty()                             /* Create an empty queue T */
 3      Π ← ∅                        /* Create an permutation array Π */
        /* 2. Determination of a starting node with minimum degree        */
 4      foreach a' ∈ arg min_{a∈V−Π} |N(a)| do
 5          Π ← Π ⋃ {a'}
 6          Sort the nodes in N(a') by degrees in ascending order
 7          T.enqueue(N(a'))
            /* 3. Main loop                                               */
 8          while T ≠ ∅ do
 9              b ← T.dequeue()
10              if b ∈ Π then continue
11              Π ← Π ⋃ {b}
12              Sort the nodes in N(b) by degrees in ascending order
13              T.enqueue(N(b))
        /* 4. Reverse ordering                                            */
14      for i ← 1 : n do
15          Π'(n + 1 − i) ← Π(i)
16      return Π'
```

We now show how the algorithm Extended_RCM computes the permutation π in the graph $\mathcal{G}_{\mathbf{Q}}$. Firstly, we need to symmetrize the digraph $\mathcal{G}_{\mathbf{Q}}$ to an undirected graph $\mathcal{G}_{\mathbf{Q}+\mathbf{Q}^T}$ by removing the direction of each edge. Then, we build a level structure for $\mathcal{G}_{\mathbf{Q}}$ by breadth-first search (BFS) and order the nodes by increasing degrees from the starting node. More concretely, we start to select a node with a minimal degree of 1 in $\mathcal{G}_{\mathbf{Q}+\mathbf{Q}^T}$, say $v_1=2$, and add it to $\mathbf{\Pi}$. We also compute all the neighbors of v_1, denoted by $N(v_1)=\{6\}$ and insert them in an increasing order at the back queue $\mathbf{T}$. Since $\mathbf{T}$ is a FIFO data structure, we remove the node at the front of $\mathbf{T}$, say $v_2=6$, and add it into $\mathbf{\Pi}$. The algorithm then repeatedly calculates its neighbors and pushes them into $\mathbf{T}$ with an increasing order until $\mathbf{T}$ is empty. If the size of $\mathbf{\Pi}$ is n (meaning that all nodes have been relabeled), then Extended_RCM returns the reversed order of $\mathbf{\Pi}$. If not, the algorithm starts to explore another component, and again chooses another starting node with a minimal degree in its component till all nodes in all components of the graph have been visited. Table 2 illustrates the detailed process on how Extended_RCM computes $\mathbf{\Pi}$ and $\mathbf{T}$.

Complexity. The algorithm consists of three phases: initialization (Lines 1-3), permutation computation (Lines 4-11), and reversed ordering result collection (Lines 12-14). One can verify that these phases take $O(1)$, $O(m+2d)$ and $O(n)$ time, respectively, where d

Table 2 COMPUTATIONAL PROCESS OF PERMUTATION

i	v_i	$\mathbf{\Pi}$	$N(v_i)$	$\mathbf{T}$
1	2	{2}	{6}	{6}
2	6	{2,6}	{$\underline{2}$,4,1}	{4,1}
3	4	{2,6,4}	{3,$\underline{6}$}	{1,3}
4	1	{2,6,4,1}	{5,3,$\underline{6}$}	{3,5,3}
5	3	{2,6,4,1,3}	{$\underline{4}$,$\underline{1}$}	{5,3}
6	5	{2,6,4,1,3,5}	{$\underline{1}$}	{3}
7	3	{2,6,4,1,3,5}	{$\underline{1}$}	{}

is the average degree of the graph $\mathcal{G}_{\mathbf{Q}}$. Therefore, the algorithm totally takes $O\left(m+2d\right)$ worst-case time.

4.2.2 Permuted SimRank iterative equation

We now combine the extended RCM techniques into the SimRank equation (6) for achieving smaller memory bandwidths and better I/O efficiency. We develop a permuted SimRank equation based on the following theorem.

Theorem 1 (Permuted SimRank Equation) *Let π be an arbitrary permutation with an induced permutation matrix $\mathbf{\Theta}$. For a given sparse graph $\mathcal{G}$, SimRank similarity score can be computed as*

$$\mathbf{S}^{(k)} = \pi^{-1}\left(\hat{\mathbf{S}}^{(k)}\right),$$

where $\hat{\mathbf{S}}^{(k)}$ satisfies

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

Proof Since $\pi^{-1}\left(\mathbf{I}_n\right) = \mathbf{I}_n$, we shall consider only the case when $k > 0$. Taking permutation π at both sides of SimRank equation (6) gives that

$$\begin{aligned}
\pi\left(\mathbf{S}\right) &= \pi\left(c \cdot \mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T \bigvee \mathbf{I}_n\right) \\
&= \mathbf{\Theta} \cdot \left(c \cdot \mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T\right) \cdot \mathbf{\Theta}^T \bigvee \pi\left(\mathbf{I}_n\right) \\
&= c \cdot \mathbf{\Theta} \cdot \mathbf{Q} \cdot \underbrace{\left(\mathbf{\Theta}^T \cdot \mathbf{\Theta}\right)}_{=\mathbf{I}} \cdot \mathbf{S} \cdot \underbrace{\left(\mathbf{\Theta}^T \cdot \mathbf{\Theta}\right)}_{=\mathbf{I}} \cdot \mathbf{Q}^T \cdot \mathbf{\Theta}^T \bigvee \mathbf{I}_n \\
&= c \cdot \underbrace{\left(\mathbf{\Theta} \cdot \mathbf{Q} \cdot \mathbf{\Theta}^T\right)}_{=\pi(\mathbf{Q})} \cdot \underbrace{\left(\mathbf{\Theta} \cdot \mathbf{S} \cdot \mathbf{\Theta}^T\right)}_{=\pi(\mathbf{S})} \cdot \underbrace{\left(\mathbf{\Theta} \cdot \mathbf{Q} \cdot \mathbf{\Theta}^T\right)^T}_{=\pi(\mathbf{Q})^T} \bigvee \mathbf{I}_n \\
&= c \cdot \pi\left(\mathbf{Q}\right) \cdot \pi\left(\mathbf{S}\right) \cdot \pi(\mathbf{Q})^T \bigvee \mathbf{I}_n
\end{aligned}$$

Let $\hat{\mathbf{S}} \triangleq \pi\left(\mathbf{S}\right) = \mathbf{\Theta} \cdot \mathbf{S} \cdot \mathbf{\Theta}^T$, it follows that

$$\mathbf{S} = \mathbf{\Theta}^T \cdot \hat{\mathbf{S}} \cdot \mathbf{\Theta} \triangleq \pi^{-1}\left(\hat{\mathbf{S}}\right)$$

so that

$$\begin{cases} \mathbf{S} = \pi^{-1}\left(\hat{\mathbf{S}}\right) \\ \hat{\mathbf{S}} = c \cdot \pi\left(\mathbf{Q}\right) \cdot \hat{\mathbf{S}} \cdot \pi(\mathbf{Q})^T \bigvee \mathbf{I}_n \end{cases} \tag{10}$$

and this results in the above iterations, which completes the proof.

This theorem implies that the optimal bandwidth compression technique for sparse non-symmetric adjacency matrix is a very promising choice for large scale SimRank computations. The concentration of nonzero entries about the main diagonal may result in a significant reduction on not only the banded SimRank solvers but also memory storage and arithmetic operations consumed.

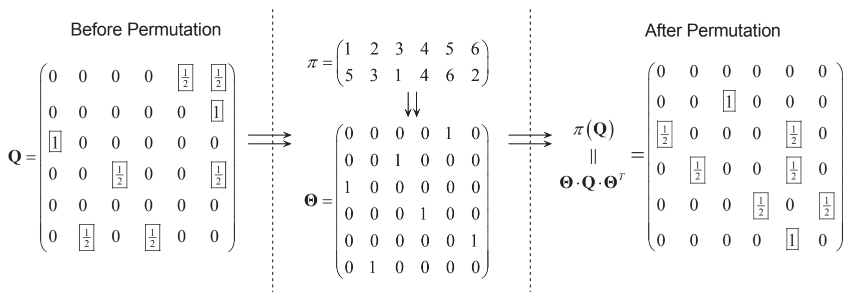


Fig. 5 The relationship between permutation π and its corresponding permutation matrix $\mathbf{\Theta}$

For the SimRank computation to be I/O efficient, $\mathbf{Q}$ needs to be preordered during the precomputation. Figure 5 describes the relationship between the permutation π and its corresponding permutation matrix $\mathbf{\Theta}$. We first determine the permutation matrix $\mathbf{\Theta}$ produced by RCM Algorithm 2, for which $\pi\left(\mathbf{Q}\right) = \mathbf{\Theta} \cdot \mathbf{Q} \cdot \mathbf{\Theta}^T$ has a smaller bandwidth. Then based on equation (10), the optimal techniques in earlier subsections can be applied to compute the k-th iterative permuted SimRank matrix $\hat{\mathbf{S}}^{(k)}$. And we can obtain the SimRank matrix by $\mathbf{S}^{(k)} = \pi^{-1}\left(\hat{\mathbf{S}}^{(k)}\right) = \mathbf{\Theta}^T \cdot \hat{\mathbf{S}} \cdot \mathbf{\Theta}$.

4.3 SOR SimRank Acceleration

When the permuted SimRank equation is established, the optimization technique presented in this subsection allows significantly accelerating the convergence rate for computing $\mathbf{S}^{(k)}$. The main idea behind the optimization is that a *successive over-relaxation (SOR)* iterative method is used for computing $\mathbf{S}^{(k)}$ and can thus effectively exhibit faster convergence than the existing technique [5].

We consider the SimRank problem $\mathbf{S}^{(k+1)} = c \cdot \mathbf{Q} \cdot \mathbf{S}^{(k)} \cdot \mathbf{Q}^T \bigvee \mathbf{I}_n$, where $\mathbf{Q} = \left(q_{i,j}\right)_{n \times n}$, $\mathbf{S}^{(k)} = \left(s_1{}^{(k)}\, s_2{}^{(k)} \cdots s_n{}^{(k)}\right)$, and $s_i{}^{(k)}$ denotes the i-th column vector of matrix $\mathbf{S}^{(k)}$. For each $s_i{}^{(k)} \quad (i = 1, 2, \cdots, n)$, we can write (6) in the component form

$$
\begin{aligned}
s_i &= c \cdot \mathbf{Q} \cdot \left(\sum_{j=1}^{n} q_{i,j} \cdot s_j \right) \bigvee \mathbf{I}_n \\
&= c \cdot \mathbf{Q} \cdot \left(\sum_{j<i} q_{i,j} \cdot s_j + q_{i,i} \cdot s_i + \sum_{j>i} q_{i,j} \cdot s_j \right) \bigvee \mathbf{I}_n
\end{aligned}
$$

Since $q_{i,i} = 0$, we can carry out the following iteration

$$
s_i^{GS\ (k+1)} = c \cdot \mathbf{Q} \cdot \left(\sum_{j<i} q_{i,j} \cdot {s_j}^{(k)} + \sum_{j>i} q_{i,j} \cdot {s_j}^{(k+1)} \right) \bigvee \mathbf{I}_n \tag{11}
$$

where $s_i^{GS\ (k+1)}$ is a Gauss-Seidel auxiliary vector. The actual components $s_i^{SOR\ (k+1)}$ of this iterative method are then defined from

$$
\begin{aligned}
s_i^{SOR\ (k+1)} &= s_i^{SOR\ (k)} + \omega \left(s_i^{GS\ (k+1)} - s_i^{SOR\ (k)} \right) \\
&= (1-\omega)\, s_i^{SOR\ (k)} + \omega \cdot s_i^{GS\ (k+1)}
\end{aligned} \tag{12}
$$

where ω is a *relaxation factor*, $s_i^{SOR\ (k+1)}$ is a weighted mean of $s_i^{SOR\ (k)}$ and $s_i^{GS\ (k+1)}$, which can be computed sequentially using forward substitution. Now we substitute (11) back into the above equation to get

$$
\begin{aligned}
s_i^{SOR\ (k+1)} &= (1-\omega)\, s_i^{SOR\ (k)} \\
&\quad + \omega \cdot c \cdot \mathbf{Q} \left(\sum_{j<i} q_{i,j} \cdot {s_j}^{(k)} + \sum_{j>i} q_{i,j} \cdot {s_j}^{(k+1)} \right) \bigvee \mathbf{I}_n
\end{aligned}
$$

And we call this equation the *successive over-relaxation SimRank iteration*.

Choosing the value of ω plays a crucial part in the convergence rate of our algorithm. It has been proven in [14] that when $0 < \omega < 2$, the SOR iterative method converges; $\omega = 1$ shows that the iteration simplifies to the Gauss-Seidel iteration; $\omega > 1$ is used to significantly accelerate convergence, corresponding to overrelaxation.

To determine the optimal over-relaxation factor ω_{opt} for SOR, [14] gives an a priori estimate in terms of the spectral radius of the Jacobi matrix, say $\rho(\mathbf{Q}_J)$ as follows: $\omega_{opt} = \frac{2}{1+\sqrt{1-\rho^2(\mathbf{Q}_J)}}$. However, computing $\rho(\mathbf{Q}_J)$ requires an impractical amount of computation. Hence, for our purpose, we empirically take the optimal value $\omega \approx 1.3 \pm 0.1$, which gives a significant improvement in the convergence rate of the existing technique [5].

4.4 The Complete Algorithm

Algorithm 3 depicts the complete algorithm that combines the SOR accelerative technique with the CSR format representation, the pruning technique and the permuted SimRank equation introduced in the previous subsections.

Algorithm 3: SOR_SimRank_Iteration $\left(\mathcal{G}_{\mathbf{Q}}, \epsilon, c, \omega, \lambda, \varDelta\right)$

Input : weh graph $\mathcal{G}_{\mathbf{Q}}$ with the transpose of column-normalized adjacency matrix in CSR format $\mathbf{Q} = \left\langle val_{\mathbf{Q}},\ col_idx_{\mathbf{Q}},\ row_ptr_{\mathbf{Q}} \right\rangle \in \mathbb{R}^{n \times n}$,
accuracy ϵ, decay factor c, relaxation factor ω, teleportation factor λ, threshold $\varDelta$

Output: SimRank matrix $\mathbf{S} = (s_{i,j}) \in [0,1]^{n \times n}$,
the number of iterations k

```
1  begin
2      Initialize S̃ ← 0,   S ← I_n,   k ← 0
       /* Modify the transition matrix for deadlock prevention */
3      Update Q ← λ · Q + (1 − λ) · (1/n) · E_n
       /* Reorder the transition matrix to minimize its bandwidth */
4      Initialize π ← ExtendedRCM (G_{Q+Q^T})
5      while (‖Ŝ − S̃‖_2 ≥ ε) do
6          for i ← 1 : n do
7              Initialize v ← 0
8              Initialize j ← row_ptr_Q (i)
9              while (j ≤ row_ptr_Q (i + 1) − 1 && col_idx_Q (j) ≤ i) do
10                 Set v ← v + val_Q (j) · S̃ (:, col_idx_Q (j))
11                 Set j ← j + 1
12             Set r ← j
               /* SOR accelerative iteration */
13             if i = 0 then Set Ŝ ← (1 − ω) · Ŝ + ω · S̃
14             for j ← r : row_ptr_Q (i + 1) − 1 do
15                 Set v ← v + val_Q (j) · Ŝ (:, col_idx_Q (j))
16             Set S̃ (:, i) ← 0
17             for m ← 1 : n do
18                 for n ← row_ptr_Q (m) : row_ptr_Q (m + 1) − 1 do
19                     Set S̃ (m, i) ← S̃ (m, i) + c · val_Q (n) · v (col_idx_Q (n))
20             Set S̃ (i, i) ← 1
               /* Prune the similarity matrix by the threshold Δ */
21             if S̃ (i, j) ≤ Δ then Set S̃ (i, j) = 0
22         Set k ← k + 1
23     Set S ← π^{-1} (Ŝ)
24     return S, k
```

- In Line 3, the iteration is justified by Equation 7.
- In Line 4, π can be calculated by Algorithm 2.
- In Line 9, the condition in the header of the while loop is justified by Algorithm 1.
- In Line 10-11, the iteration is justified by Equation 11.
- In Line 13-14, the expression is calculated by Equation 12.
- In Line 16-20, the iteration is justified by Equation 11.
- In Line 14 and 18, the condition in the header of the for loop is justified by Algorithm 1.
- In Line 21, the pruning technique is justified by Equation 9.

It is easy to analyze that Algorithm 3 has the time complexity $O(n \cdot m)$ with the space requirement $O(n + m)$ for each iteration. It is worth mentioning that originally there might have been an inherent trade-off between computational time complexity and I/O efficiency. However, in our optimization techniques, we can achieve higher I/O efficiency while retaining computational time complexity. The reason is that the I/O efficiency of our algorithm is significantly achieved by our extended RCM and pruning techniques, whereas the two tech-

niques themselves require less time as compared to the complete SimRank algorithm, being $O(2m + 2d)$ in the worst case and even can be reduced to $O(2n + 2d)$ when the graph is sparse. By contrast, as the SOR SimRank iteration takes $O(\min\{n \cdot m, n^r\})$ time, the time consumption of the extended RCM algorithm and the pruning techniques can be ignored when compared with the total time of the SimRank iteration.

5 Experimental Evaluation

In this section, we present a comprehensive empirical study of our proposed algorithms. Using both synthetic and real-life data, we conduct two sets of experiments to evaluate: (1) the effectiveness, (2) the efficiency (i.e., computational time, storage space, convergence rate and I/O operations) and scalability of our algorithms for similarity computation.

All experiments were carried out on a Microsoft Windows Vista machine with an Intel Pentium(R) Dual-Core 2.0G CPU and 2GB main memory. We implemented the algorithms using Visual C++.

5.1 Experimental Setup

5.1.1 Datasets

Two kinds of datasets were used in the evaluation: the synthetic data sets were used to show the scalability of the algorithm as well as the parameter setting mechanism, while the real-life datasets were used to demonstrate the effectiveness and efficiency of our algorithm.

Synthetic Datasets: To test our implementations, we simulated the web graph with an average of 8 links per page. We generated 10 sample adjacency matrices with the size (number of web documents) increased from 1K to 10K and with ξ out-links on each row, where ξ ∼*uniform*$[0, 16]$ is a random variable. Two storage schemes were used respectively to represent these graphs: (a) the CSR-styled compression for sparse graphs; (b) the full matrix format for dense graphs.

Real-life Datasets: For real datasets, we verified our algorithms over (1) 10-year (from 1998 to 2007) DBLP dataset, and (2) three English Wikipedia category graphs.

From the DBLP datasets, we extracted the 10-year (from 1998 to 2007) author-paper information, and picked up papers published on 6 conferences (‘WWW’, ‘ICDE’, ‘KDD’, ‘SIGIR’, ‘VLDB’, ‘SIGMOD’). We built the digraphs where nodes represent authors or papers, and one edge represents an author-paper relationship. We reordered the authors by the number of papers they published on these conferences. Table 3 gives the details of DBLP datasets.

Table 3 THE DETAILS OF 10-YEAR DBLP DATASETS

Period	98-99	98-01	98-03	98-05	98-07
authors	1,525	3,208	5,307	7,984	10,682
edges	5,929	13,441	24,762	39,399	54,844

We also used three real-life Wikipedia category graphs (exported in 2009) to investigate the effectiveness of the our algorithms. As Wikipedia is a popular online encyclopedia, it has recently attached a growing interest in many academic fields [5,6,11,12,9,13]. We

built three category graphs from the English Wikipedia, choosing the relationship "*a category contains an article to be a link from the category to the article*". The details of these Wikipedia category graphs are in the following:

Table 4 THE DETAILS OF 3 WIKIPEDIA CATEGORY GRAPH DATASETS

Wiki Dataset	Articles	Links	Archived Date
wiki0715	3,088,185	1,126,662	Jul 15, 2009
wiki0827	3,102,904	1,134,871	Aug 27, 2009
wiki0919	3,116,238	1,139,156	Sep 19, 2009

5.1.2 Parameter Settings

For a correspondence with experiment conditions in [5], the following parameters were used as default values: (unless otherwise specified)

Table 5 DEFAULT PARAMETER SETTINGS

Notation	Description	Default Value
c	decay factor	0.8
ω	SOR over-relaxation factor	1.3
ϵ	accuracy	0.05
λ	teleportation factor	0.8
Δ	pruning threshold	0.01

5.1.3 Evaluation Metrics

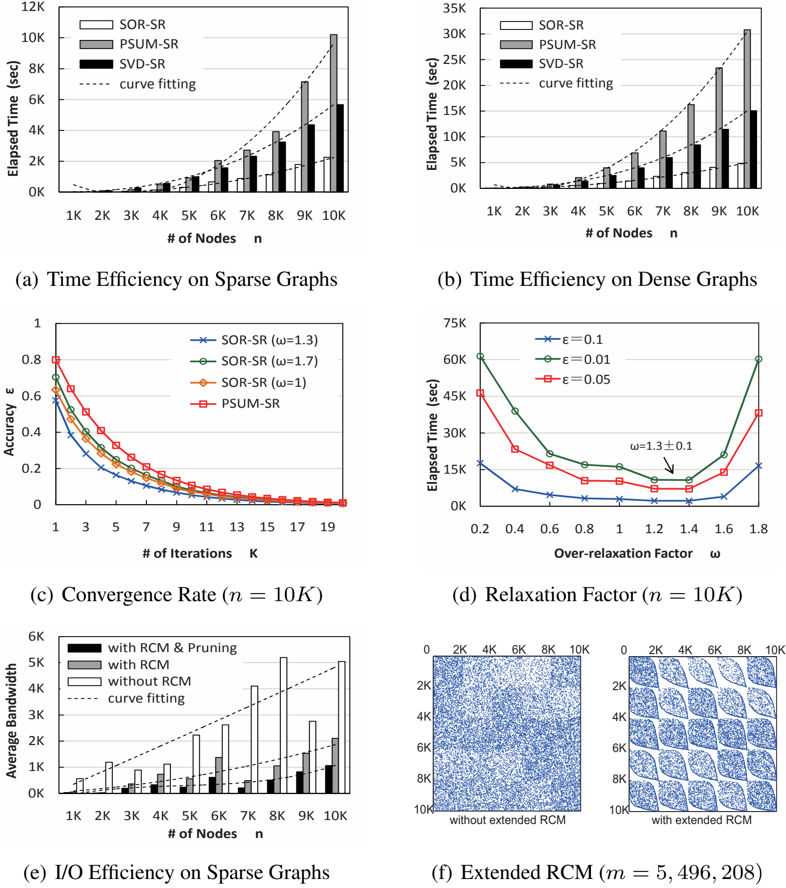
In our experiments, we evaluated the *efficiency* and *effectiveness* of our algorithms.

– The *efficiency* is measured by the computation time complexity, the space requirement, the I/O operations and the convergence rate needed to reach a certain desired SimRank accuracy.
– The *effectiveness* is measured by the average differences between two similarity matrices $ave_err(\cdot,\cdot)$ defined as

$$ave_err\left(\mathbf{S}_{n\times n}, \tilde{\mathbf{S}}_{n\times n}\right) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left|s_{i,j} - \tilde{s}_{i,j}\right| \tag{13}$$

5.1.4 Compared Algorithms

– **PSUM-SR** This algorithm adopts the iterative paradigm for similarity estimation optimized using a partial sums function (PSUM) to cluster similarity values. [5,6]
– **SVD-SR** This is a new deterministic approximation algorithm for SimRank similarity computation via a singular value decomposition (SVD) approach. [11]
– **SOR-SR** This is our proposed algorithm which considers both time and space efficiency.



(a) Time Efficiency on Sparse Graphs

(b) Time Efficiency on Dense Graphs

(c) Convergence Rate ($n = 10K$)

(d) Relaxation Factor ($n = 10K$)

(e) I/O Efficiency on Sparse Graphs

(f) Extended RCM ($m = 5,496,208$)

Fig. 6 Efficiency and Scalability on Synthetic Datasets

5.2 Experimental Results

5.2.1 Time Efficiency

In the first set of experiments, we compare the computation time of our method with that of the existing algorithms [5,6,11]. Figure 6(a) and 6(b) show the dynamics in SimRank computation time with respect to the number of nodes for 10 generated sparse and dense graphs respectively. Each bar chart is approximated by a polynomial curve, in a least squares sense. We can see that given an accuracy ϵ, our algorithms with matrix representation and SOR iteration is more time-efficient than PSUM-SR and SVD-SR algorithms for both sparse and dense graphs. Note that the different maximum value is chosen across the vertical axis in Figure 6(a) and 6(b). For sparse graphs, our method may reduce over half of the SVD-SR algorithmic time when nodes are growing, whereas for dense graphs, the time complexity of our method has been significantly improved due to the fast matrix multiplication.

The results have been well consistent with the theoretical analysis. The time complexity in [5,6] requires $O\left(n^2 \cdot d\right)$ per iteration, where d is the average node degree, resulting in

$O(n^3)$ for dense graphs. For comparison, our approach requires $O(n \cdot m)$ per iteration for computing sparse matrix multiplication. For each iteration, since $m = n \cdot d$, it is not surprising that our method for sparse graphs has the same time complexity as [5]. Hence, in Figure 6(a), it is reasonable to see that for a given accuracy ϵ, our method has improved the computational time four fold due to (1) the SOR techniques accelerating the convergence rate and reducing the number of iterations to reach a desired accuracy, and (2) the pruning techniques eliminating impractical almost zero similarity values and making similarity matrix sparse for each iteration. By contrast, for dense graphs, our method in Figure 6(b) has a significant improvement in computation time because the time consumption in [5] requires $O(n^3)$ in the dense case whilst our technique adopts the fast matrix multiplication for computing SimRank score, involving $O(n^r)$, where $r \le \log_2 7$.

5.2.2 Convergence Rate

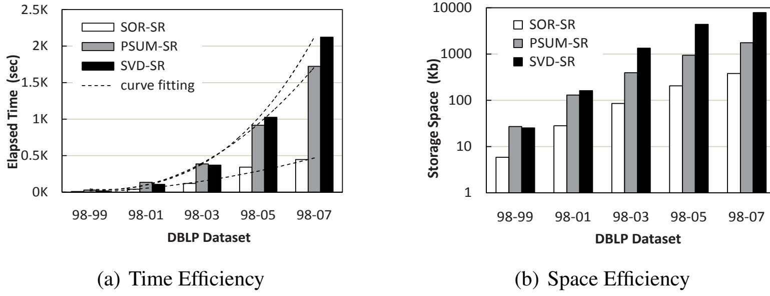
To investigate the correlation between the residual accuracy ϵ and the number of iterations K, we increase the iteration number K from 1 to 20 over a 10K generated sparse graph. As SVD-SR is based on a non-iterative framework [11], here we focus on the comparison between PSUM-SR and SOR-SR algorithms. We also vary the over-relaxation factor ω to see how the speed of convergence is influenced by the choice of ω.

Figure 6(c) compares the convergence rate of SOR-SR with that of PSUM-SR. In [5], for achieving accuracy ϵ, the existing algorithm requires $K = \lceil \log_c \epsilon \rceil - 1$ iterations. It is interesting to note that for a given accuracy ϵ, the number of iterations needed for SOR computation is much fewer than [5]. It follows that the SOR technique with $\omega = 1.3$ for computing SimRank can speed up the convergence roughly twice faster over the algorithm in [5] when $\omega = 1.3$. It also can be discerned from Figure 6(c) that choosing the relaxation factor $\omega = 1.3$, SOR-SR can achieve the most algorithmic efficiency.

Furthermore, to investigate how the relaxation factor ω and accuracy ϵ affects the total computational time of the SOR-SR algorithm, we vary ω from 0 to 2 for every given accuracy. The results in Figure 6(c) indicate that given any accuracy ϵ, the SOR-SR computational time bottomed out when $\omega \in [1.2, 1.4]$; when $\omega = 0$ or 2, our algorithm is not convergent, which fully agrees with the theoretical expectation for SOR in [14]. That is the reason why we choose $\omega = 1.3$ for achieving the best performance of our SOR-SR algorithm.

5.2.3 I/O Efficiency

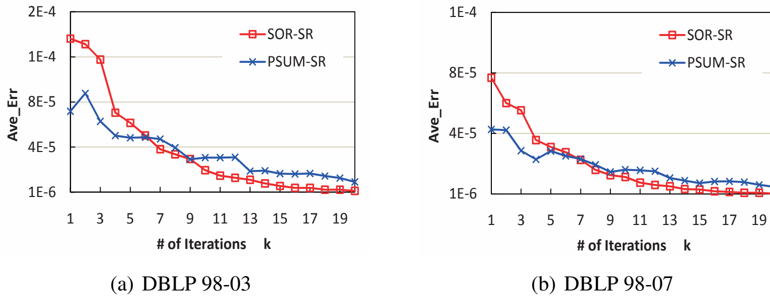
Next we show the results of applying the extended RCM algorithms and pruning techniques to the sparse matrix $\mathbf{Q}$ for the SimRank precomputation. Figure 6(e) depicts the effect of using the reordering Algorithm 2 and pruning techniques to 10 generated sparse graphs. We can see that the extended RCM and pruning technqiues do reduce the total bandwidths, keeping the matrices sparse, and can thus improve the I/O efficiency of our algorithm. In Figure 6(f), we visualize the sparsity pattern of our generated 10K $\times$ 10K adjacency matrix (a) without and (b) with the extended RCM algorithm. Here, we separate the large matrix into 25 blocks. For each block, the nonzeros will cluster as much as possible about the main diagonal of the submatrix so that the computation bandwidth may be greatly minimized.



(a) Time Efficiency

(b) Space Efficiency

Fig. 7 Efficiency and Scalability on DBLP Datasets



(a) DBLP 98-03

(b) DBLP 98-07

Fig. 8 Effectiveness on DBLP Datasets

5.2.4 Space Efficiency

For achieving storage efficiency, the CSR scheme is adopted for our large and sparse matrix representations, yielding significant savings in memory usage. From the space perspective, we implement corresponding arithmetic operations such as matrix-matrix multiplications for our algorithm.

Figure 7(b) shows the space comparison between the three methods on DBLP datasets with the five year periods: 98-99, 98-01, 98-03, 98-05 and 98-07. Note that a logarithmic scale has been used on the vertical axis, corresponding to the space consumption for the given method over DBLP with different data sizes (year periods). As shown in the figure, SOR-SR achieves better space efficiency with small memory consumption, whereas SVD-SR requires a significant amount of space as the dataset size is growing. When the year period is chosen to be 98-99, the space required by PSUM-SR is similar to that of SVD-SR, but not as good as that of SOR-SR. When the span time increases, SVD-SR needs far more space than PSUM-SR. This is because SVD-SR does not preserve sparseness of the similarity matrix. Therefore, after SVD decomposition, most of the entries in the result matrices are nonzeros even if the original DBLP matrix is sparse. In comparison, SOR-SR uses a sparse matrix representation combined with a pruning techniques per iteration, thus achieving high space efficiency.

5.2.5 Effectiveness

In the final set of experiments, we evaluated the effectiveness of SOR-SR vs. PSUM-SR over two kinds of real-life datasets (i.e., DBLP and Wikipedia) as these two algorithms are both based on iterative techniques. We used $ave_err(\cdot,\cdot)$ defined by Equation (13) to measure the accuracy of computational results. We denote by $\mathbf{S}_{PSUM}^{(k)}$ and $\mathbf{S}_{SOR}^{(k)}$ the k-th iterative similarity matrix for the algorithms SOR-SR and PSUM-SR, respectively. Notice that for each of the algorithm, the iterative similarity matrices both converge to the same exact (theoretical) similarity matrix $\mathbf{S}^*$, i.e., $\lim_{k\to\infty}\mathbf{S}_{PSUM}^{(k)} = \lim_{k\to\infty}\mathbf{S}_{SOR}^{(k)} = \mathbf{S}^*$. By Cauchy criterion for convergence [14], we can infer that $ave_err\left(\mathbf{S}_{PSUM}^{(k)}, \mathbf{S}_{PSUM}^{(k-1)}\right) \to 0$ and $ave_err\left(\mathbf{S}_{SOR}^{(k)}, \mathbf{S}_{SOR}^{(k-1)}\right) \to 0$ as $k \to \infty$. Hence, (1) for the given algorithm, the gap between two adjacent iterates , say $ave_err\left(\mathbf{S}^{(k)}, \mathbf{S}^{(k-1)}\right)$ (for $k = 1, 2, \cdots$) , implies the speed of convergence to reach the exact solution $\mathbf{S}^*$; (2) for two distinct algorithms, the gap $ave_err\left(\mathbf{S}_{SOR}^{(k)}, \mathbf{S}_{PSUM}^{(k)}\right)$ indicates the average similarity difference between SOR-SR and PSUM-SR at the k-th iteration.

Period	98-99		98-01		98-03		98-05		98-07	
K	SOR-SR	PSUM-SR	SOR-SR	PSUM-SR	SOR-SR	PSUM-SR	SOR-SR	PSUM-SR	SOR-SR	PSUM-SR
1	2.80E-4	1.52E-4	6.40E-4	2.38E-4	1.37E-4	7.27E-5	9.93E-5	4.63E-5	7.78E-5	4.36E-5
5	1.31E-4	1.08E-4	2.38E-4	1.66E-4	6.24E-5	4.92E-5	4.51E-5	3.28E-5	3.19E-5	2.93E-5
10	4.59E-5	6.65E-5	8.04E-5	1.02E-4	2.05E-5	3.16E-5	1.32E-5	1.95E-5	1.21E-5	1.70E-5
15	1.43E-5	3.49E-5	2.55E-5	5.82E-5	6.52E-6	1.74E-5	4.22E-6	1.12E-5	3.87E-6	8.12E-6
20	4.42E-6	2.23E-5	8.85E-6	4.55E-5	2.21E-6	1.01E-5	1.34E-6	7.20E-6	1.22E-6	5.72E-6

Table 6 EFFECTIVENESS OF SOR-SR & PSUM-SR ON DBLP DATASETS

Table 6 shows the results for DBLP datasets with average similarity difference between SOR-SR and PSUM-SR when we set the iteration number $k = 1, 5, 10, 15, 20$. From the results, we can see that SOR-SR outperformed PSUM-SR in terms of effectiveness since the values of $ave_err\left(\mathbf{S}_{SOR}^{(k)}, \mathbf{S}_{SOR}^{(k-1)}\right)$ converge faster than that of PSUM-SR per iteration, which enables SOR-SR achieves a higher accuracy than PSUM-SR with the same number of total iterations. More concrete comparable results are reported in Figures 8(a) and 8(b), which visualizes the changes of the average differences between the two adjacent similarity matrix iterates for SOR-SR and PSUM-SR algorithms over DBLP 98-03 and 98-07 datasets, respectively.

We next verified the efficiency and effectiveness of our methods for similarity computation over the Wikipedia category graphs described in Table . Since the Wikipedia category graphs are huge and sparse (i.e., the corresponding transition matrices with few non-zero entries), we represent them in the CSR format and set a threshold to eliminate impractical similarity values in SimRank matrices per iteration, thus saving significant amount of space. We chose $c = 0.6, \epsilon = 0.1$ corresponding with the evaluation conditions in [5]. We set the cache size of 128MB for Oracle Berkeley DB and kept the Wikipedia graphs in the CSR format.

From the efficiency perspective, our evaluations on three Wikipedia category graphs reported that SOR-SR takes an average time of approximately 12 hours with only 3 iterations to complete the similarity computation on one processor for reaching a desired accuracy. In comparison, PSUM-SR requires almost an average of 25 hours time with 6 iterations, almost

doubling the amount of SOR-SR time. SVD-SR takes about 19 hours (including about 5-hour in the precomputation phase) to get approximate similarity values and its memory consumption seems rather significant due to the Kronecker tensor product operations.

Wiki DataSet	wiki0715		wiki0825		wiki0919	
K	SOR-SR	PSUM-SR	SOR-SR	PSUM-SR	SOR-SR	PSUM-SR
2	8.74E-5	5.76E-4	8.63E-5	4.76E-4	5.91E-5	4.50E-4
4	3.09E-5	1.09E-4	4.04E-5	1.32E-4	4.82E-5	1.64E-4
6	3.19E-5	5.22E-5	3.31E-5	6.59E-5	3.41E-5	9.02E-5

Table 7 EFFECTIVENESS OF SOR-SR & PSUM-SR ON WIKIPEDIA DATASETS

From the effectiveness perspective, we showed the values of $ave_err\left(\mathbf{S}^{(k)},\mathbf{S}^{(k-1)}\right)$ $(k=2,4,6)$ in Table 7 to compare the effectiveness of SOR-SR and PSUM-SR over the three Wikipedia graphs. As we can observe in Table 7 that both of the algorithms have relatively small average similarity differences between the adjacent iterates. A reasonable explanation is that both algorithms have achieved a fast rate of convergence. Another interesting finding is that for almost any iteration number k, we have $ave_err\left(\mathbf{S}_{PSUM}^{(k)},\mathbf{S}_{PSUM}^{(k-1)}\right) > ave_err\left(\mathbf{S}_{SOR}^{(k)},\mathbf{S}_{SOR}^{(k-1)}\right)$, which indicates that SOR-SR has high computational accuracy due to over-relaxation accelerative iteration.

The results over the real-life datasets demonstrate that our method is preferable on a single machine as it yields less computation time and storage requirement, which agrees with our theoretical analysis addressed in Sect. 4.

6 Related Work

The issue of measuring object-to-object similarity has attracted a lot of attention. Existing work on similarity search techniques can be distinguished into two broad categories: text-based and link-based [1,3,4,21,22].

The link-based similarity computation can be modeled by a web-graph, with vertices corresponding to web pages and edges to the hyperlinks between pages. In terms of a graph structure, the methods of *bibliographic coupling* [23] and *co-citation* [24] have been applied to cluster scientific papers according to topic. In both schemes, similarities between two nodes are measured only from their *immediate* neighbors. As a generalization of similarity functions to exploit the information in *multi-step* neighborhoods, HITS [25], PageRank [22], SimRank [1] and SimFusion [26] algorithms were suggested by adapting link-based ranking schemes.

Jeh first introduced a similarity measure called SimRank [1] aiming at "*two pages are similar if they are referenced by similar pages*". The underlying intuition behind the SimRank approach somewhat resembles the one for SimFusion "*integrating relationships from multiple heterogeneous data sources*". In [1], SimRank is known to be efficient since it recursively refines the co-citation measure and forms a homogenous language-independent data set.

Optimization algorithms for SimRank computation have been explored in [3,5,7,8]. Results show that the use of fingerprint trees and random permutations with extended Jaccard coefficient can approximately compute SimRank scores under a scalable Monte Carlo

framework. The algorithms in [3] use *probability theory* to calculate *the expected-f meeting time* $\tau(u,v)$ and estimate $s(u,v)$ by $\mathbb{E}\left(c^{\tau(u,v)}\right)$. The solution is rather *stochastic*. In comparison, our algorithm can get a *deterministic* solution by using numerical techniques for computing SimRank.

There has also been a host of work for computing SimRank deterministically, the most efficient optimization techniques presented in [5,6] introduced *a partial sum function* to reduce the number of access operations to the SimRank function and speed up similarity scores calculation by $s_k(u,*)$ values clustering. The algorithm in [5,6] has improved SimRank computational complexity from $O\left(Kn^2 \cdot d^2\right)$ in [1] to $O\left(Kn^2 \cdot d\right)$, where d is the average node degree, n is the number of nodes. In comparison, our method has achieved the same time complexity $O(Kn \cdot m)$ for sparse graphs, where m is the number of edges. When the graph is rather dense, the time complexity in [5,6] is $O\left(Kn^3\right)$, whereas our technique only requires $O(n^r)$ operations, where $r \leq \log_2 7$, taking advantage of fast matrix multiplications. In addition, our algorithm also accelerates the convergence rate of [5,6].

Li et al. [11] proposed a novel approximate SimRank computation algorithm for static and dynamic information networks. Their approximation algorithm is based on the non-iterative framework, taking $O\left(n^2\alpha^4\right)$ time and $O\left(n^2\right)$ space, where α is the rank of graph adjacency matrix. However, the optimization technique of Kronecker product in their approach is prohibitively costly in computational time. Additionally, for a sparse graph, the SVD decomposition cannot keep the sparsity of the graph adjacency matrix. Therefore, in practice their method is not preferable.

As for SimRank parallel algorithms, Yu et al. [9] devised an AUG-SimRank algorithm over undirected graphs on distributed memory multi-processors. They combined the PLAPACK solvers with their partition techniques to parallelize the SimRank algorithm. He et al. [12] also proposed a *graphics processing units* (GPU) based *parallel* framework for similarity computation.

Li et al. [27] 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. [28] 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. Antonellis et al. [8] extended the weighted and evidence-based SimRank yielding better query rewrites for sponsored search; however, their framework lacks a solid theoretical background and the edge weight in the transition probability is an empirical distribution.

Meanwhile, Xi et al. [26] introduced SimFusion algorithm to represent heterogeneous data objects. The *Unified Relationship Matrix* (URM) approach is employed to support for various intra-nodes relations and information spaces. SimFusion iterative reinforcement similarity score takes the form:

$$
\begin{aligned}
\mathbf{S}_{usm}^{k}(a,b) &= \mathbf{L}_{urm}(a) \cdot \mathbf{S}_{usm}^{k-1}(a,b) \cdot \left(\mathbf{L}_{urm}(b)\right)^T \\
&= \frac{1}{|I(a)|\,|I(b)|} \sum_{j=1}^{|I(b)|} \sum_{i=1}^{|I(a)|} \mathbf{S}_{usm}^{k-1}\left(I_i(a), I_j(b)\right)
\end{aligned}
$$

where $\mathbf{L}_{urm}$ is a single step probability transformation matrix in a Markov Chain that combines all the relationships among nodes, $\mathbf{S}_{urm}$ is a *Unified Similarity Matrix* (USM) that

represents similarity values between node pairs. The computational complexity for SimFusion is $O\left(n^3\right)$ whilst our approach takes the time $O\left(\min\left\{n \cdot m, n^r\right\}\right)$, where $r \leq \log_2 7$. The storage for SimFusion requires $O\left(n^2\right)$, whereas we use CSR representation for reducing the space requirement to $O\left(n+m\right)$ for sparse graphs. Moreover, our algorithm is I/O efficient, minimizing the bandwidth during the precomputation and has the faster convergence rate. Finally, some of the iterative matrix-analytic methods used in this work are surveyed in [14].

7 Conclusions

This paper investigated the optimization issues for SimRank computation. We first extended the SimRank transition matrix in our conference paper [13], and formalized the SimRank equation in matrix notations. A compressed storage scheme for sparse graphs is adopted for reducing the space requirement from $O\left(n^2\right)$ to $O\left(n+m\right)$, whereas a fast matrix multiplication for dense graph is used for improving the time complex from $O\left(n^2 \cdot d\right)$ to $O\left(\min\left\{n \cdot m, n^r\right\}\right)$, where $r \leq \log_2 7$. Then, for achieving the I/O efficiency of our algorithm, we developed a permuted SimRank iteration in combination of the extended Reversed Cuthill-McKee algorithm. We also devised a pruning technique for the similarity matrix to get rid of the impractical almost zero similarity values, keeping the sparseness of similarity matrix for each iteration. Finally, we have shown a successive over-relaxation method for computing SimRank to significantly speed up the convergence rate of the existing technique. Our experimental evaluations on synthetic and real-life data sets demonstrate that our algorithms have high performances in time and space, and can converge much faster than the existing approaches.

References

1. Jeh, G., Widom, J.: Simrank: a measure of structural-context similarity. In: KDD. (2002)
2. Pathak, A., Chakrabarti, S., Gupta, M.S.: Index design for dynamic personalized pagerank. In: ICDE. (2008)
3. Fogaras, D., Rácz, B.: Scaling link-based similarity search. In: WWW. (2005)
4. Fogaras, D., Racz, B.: A scalable randomized method to compute link-based similarity rank on the web graph. In: EDBT Workshops. (2004)
5. Lizorkin, D., Velikhov, P., Grinev, M., Turdakov, D.: Accuracy estimate and optimization techniques for simrank computation. PVLDB **1**(1) (2008)
6. Lizorkin, D., Velikhov, P., Grinev, M.N., Turdakov, D.: Accuracy estimate and optimization techniques for simrank computation. VLDB J. **19**(1) (2010)
7. Cai, Y., Li, P., Liu, H., He, J., Du, X.: S-simrank: Combining content and link information to cluster papers effectively and efficiently. In: ADMA. (2008)
8. Antonellis, I., Garcia-Molina, H., Chang, C.C.: Simrank++: query rewriting through link analysis of the click graph. PVLDB **1**(1) (2008)
9. Yu, W., Lin, X., Le, J.: Taming computational complexity: Efficient and parallel simrank optimizations on undirected graphs. In: WAIM. (2010)
10. Zhou, Y., Cheng, H., Yu, J.X.: Graph clustering based on structural/attribute similarities. PVLDB **2**(1) (2009)
11. 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)
12. He, G., Feng, H., Li, C., Chen, H.: Parallel simrank computation on large graphs with iterative aggregation. In: KDD. (2010)
13. Yu, W., Lin, X., Le, J.: A space and time efficient algorithm for simrank computation. In: APWeb. (2010)
14. Bhatia, R.: Matrix Analysis. Springer (1997)

15. D'Azevedo, E.F., Fahey, M.R., Mills, R.T.: Vectorized sparse matrix multiply for compressed row storage format. In: International Conference on Computational Science (1). (2005)
16. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symb. Comput. **9**(3) (1990)
17. Cohen, J., Roth, M.S.: On the implementation of strassen's fast multiplication algorithm. Acta Inf. **6** (1976)
18. Coppersmith, D., Winograd, S.: On the asymptotic complexity of matrix multiplication. SIAM J. Comput. **11**(3) (1982)
19. Chan, W.M., George, A.: A linear time implementation of the reverse cuthill-mckee algorithm. BIT **20**(1) (1980)
20. Lim, A., Rodrigues, B., Xiao, F.: Heuristics for matrix bandwidth reduction. European Journal of Operational Research **174**(1) (2006)
21. Quevedo, J.U., Huang, S.H.S.: Similarity among web pages based on their link structure. In: IKE. (2003)
22. Lawrence Page, Sergey brin, R.M., Winograd, T.: The pagerank citation ranking bringing order to the web (1998) Technial report.
23. Weinberg, B.H.: Bibliographic coupling: A review. Information Storage and Retrieval **10**(5-6) (1974)
24. 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)
25. Mendelzon, A.O.: Review - authoritative sources in a hyperlinked environment. ACM SIGMOD Digital Review **1** (2000)
26. 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)
27. Li, P., Cai, Y., Liu, H., He, J., Du, X.: Exploiting the block structure of link graph for efficient similarity computation. In: PAKDD. (2009)
28. Zhao, P., Han, J., Sun, Y.: P-rank: a comprehensive structural similarity measure over information networks. In: CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management. (2009)