2010 12th International Asia-Pacific Web Conference

A Space and Time Efficient Algorithm for SimRank Computation

Weiren Yu
Dept. of Computer Science & Technology
Donghua University
Shanghai, China
ywr0708@mail.dhu.edu.cn

Xuemin Lin
School of Computer Science & Engineering
University of New South Wales
NSW, Australia
lxue@cse.unsw.edu.au

Jiajin Le
Dept. of Computer Science & Technology
Donghua University
Shanghai, China
lejiajin@dhu.edu.cn

***Abstract*—SimRank has been proposed to rank web documents based on a graph model on hyperlinks. The existing techniques for conducting SimRank computation adopt an iteration computation paradigm. The most efficient technique has the time complexity $O\left(n^3\right)$ with the space requirement $O\left(n^2\right)$ in the worst case for each iteration where n is the number of nodes (web documents). In this paper, we propose novel optimization techniques such that each iteration takes the time $O\left(\min\left\{n\cdot m, n^r\right\}\right)$ and requires space $O\left(n+m\right)$ where m is the number of edges in a web-graph model and $r \leq \log_2 7$. We also show that our algorithm accelerates the convergence rate of the existing techniques. Moreover, our algorithm not only reduces the time and space complexity of the existing techniques but is also I/O efficient. 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;**

I. Introduction

Recently, the complex hyperlink-based similarity search has attracted considerable attention in the field of Information Retrieval. One of the promising measures is the SimRank similarity with applications to search engine ranking and document corpora clustering. 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 pages is defined recursively as the average similarity between their neighbours, along with the base case that a 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 the efficient SimRank computation, it is desirable to have optimization techniques that improve the time and space complexity of the SimRank algorithm. The idea of approximating SimRank scores has been studied in [3] based on *Monte Carlo method*. This computation model is inherently stochastic. However, with respect to the *non-probabilistic* SimRank iterative computation, little work has been done to establish a theoretical foundation of the optimization. The straightforward iterative SimRank computation has the time complexity $O\left(Kn^4\right)$ in the worst case and requires the space $O\left(n^2\right)$ [1]. The bottleneck mainly lies in high computational complexity. To the best of our knowledge, there is only one research work in [5] concerning *deterministic* methods for SimRank optimization. In that work SimRank computation takes the time $O\left(n^3\right)$ per iteration in the worst case with the space requirement $O\left(n^2\right)$, which has yet been regarded as the most efficient technique in *non-probabilistic* SimRank iteration.

Contributions: In this paper, we investigate the optimal algorithms that can further improve the efficiency of SimRank computation. We provide theoretical guarantee for our methods and present experimental results. The main contributions of this paper are summarized below:

- We introduce a matrix representation and storage schemes for SimRank model to reduce space requirement from $O\left(n^2\right)$ to $O\left(m+n\right)$ with 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 $r \leq \log_2 7$.
- We develop optimization techniques for minimizing the matrix bandwidth for SimRank computation, which may improve the I/O efficiency of SimRank iteration.
- We show a successive over-relaxation method for SimRank computation to significantly accelerate the convergence rate of the existing technique.

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

II. 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].

A. Problem Definition

Given a directed graph $\mathcal{G} = (V, E)$, where 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

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

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 $I_i(a) \quad (1 \leq i \leq |I(a)|)$, 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

978-0-7695-4012-2/10 $26.00 © 2010 IEEE
DOI 10.1109/APWeb.2010.42

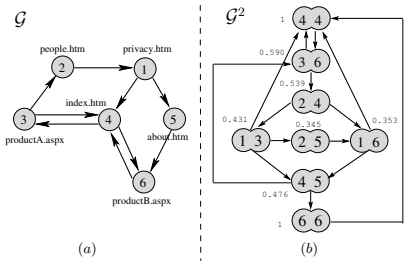


Figure 1. SimRank propagating similarity from pair to pair in $\mathcal{G}$ associated with the propagation from node to node in $\mathcal{G}^2$ with a decay factor $c = 0.8$

fixed-point, we can carry out the following iteration for SimRank computation.

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

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

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 I, 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 I
SYMBOLS AND NOTATIONS

Sym	Definition	Sym	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	error of 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

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

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. IV-B.

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. IV-C.

IV. Optimizations For SimRank Algorithms

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

A. Matrix Representations for SimRank Model

For an elaborate discussion on the subject, we first consider the SimRank similarity problem in matrix formulation. Let $\mathbf{S} = (s_{i,j}) \in \mathbb{R}^{n \times n}$ be a SimRank matrix of $\mathcal{G}$ whose entry $s_{i,j}$ equals the similarity score between page i and j, and $\mathbf{P} = (p_{i,j}) \in \mathbb{N}^{n \times n}$ 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

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

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

where we assume, without loss of generality, that $a \neq b$ (otherwise, $s_{a,a}^{(k)} \equiv 1 \quad (\forall k = 0,1,2,\cdots)$).

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

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

As we have seen in equation (6), the computational complexity is $O(n^3)$ per iteration with the space requirement $O(n^2)$ 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 $O(n^3)$ operations for all entries of $\mathbf{U} \in \mathbb{R}^{n \times n}$. In the following, two techniques are investigated to obtain the time and space efficient algorithms for SimRank computation. 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 $r \leq \log_2 7$.

1) 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 [6], 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 $k \in [row\_ptr(i), row\_ptr(i+1))$.

$$\mathbf{Q} = \begin{pmatrix} 0 & \boxed{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \boxed{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \boxed{1} & 0 & 0 \\ \boxed{\frac{1}{3}} & 0 & \boxed{\frac{1}{3}} & 0 & 0 & \boxed{\frac{1}{3}} \\ \boxed{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \boxed{\frac{1}{2}} & \boxed{\frac{1}{2}} & 0 \end{pmatrix}$$

index	1	2	3	4	5	6	7	8	9
val	1	1	1	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	1	$\frac{1}{2}$	$\frac{1}{2}$
col_idx	2	3	4	1	3	6	1	4	5

row_ptr	1	2	3	4	7	8	10

Figure 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 [6].

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

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 $\mathbf{U} = (u_{i,j}) \in \mathbb{R}^{n \times n} \leftarrow \mathbf{Q} \cdot \mathbf{S}$

```
begin
    Initialize U ← 0
    for i ← 1 : n do
        for j ← 1 : n do
            for k ← row_ptr_Q (j) : row_ptr_Q (j + 1) − 1 do
                Calculate U (j, i) ←
                U (j, i) + val_Q (k) × S (col_idx_Q (k), i)
    return U
```

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.

2) 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 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* [7] 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* [8]. The interested reader can refer to [9], [7], [8] 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 $O(\min\{n \cdot m, n^r\})$ 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.

B. Permuted SimRank Iterative Approach

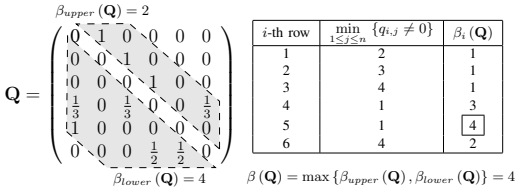
After the CSR storage scheme has been created for the sparse adjacency matrix $\mathbf{Q}$, the optimization technique suggested in this subsection allows improving I/O efficiency for SimRank computation. The main idea behind this optimization involves two steps: (a) *Reversed Cuthill-McKee(RCM) algorithm* [10] 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* [11].

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

$$\beta(\mathbf{Q}) \triangleq \max_{1 \leq i \leq n} \beta_i(\mathbf{Q})$$

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



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

Figure 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 [10].

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

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

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

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

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.

1) Reversed Cuthill-McKee (RCM) algorithm for directed graph: The RCM algorithm for directed graph [10] 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 [10] 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.

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 $\mathcal{G}_{\mathbf{Q}}$ and $\mathcal{G}_{\pi(\mathbf{Q})}$ have the identical structure and the different node labeling when we choose a new permutation π on both rows and columns of matrix $\mathbf{Q}$. Thus, the permutation π can be thought of as a bijection between the vertices of the labeled graph $\mathcal{G}_{\mathbf{Q}}$ and $\mathcal{G}_{\pi(\mathbf{Q})}$. And the bandwidth $\beta(\pi(\mathbf{Q}))$ is often no greater than $\beta(\mathbf{Q})$. 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 a given matrix. Observe that the popular *Reversed Cuthill-McKee(RCM) algorithm* [10] 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 a modification of RCM algorithm [10].

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.

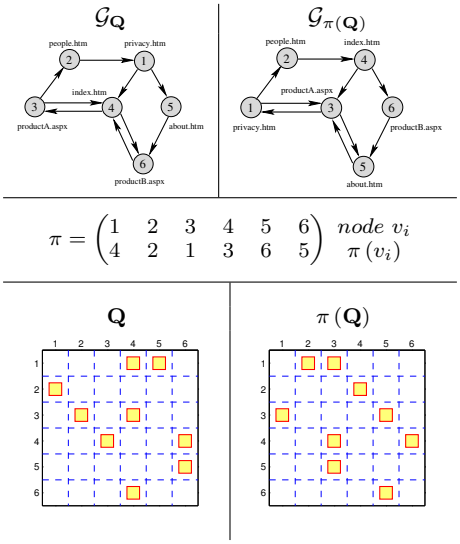


Figure 4. A simplified version of SimRank minimum bandwidth ordering problem with permutation π on the adjacency matrix $\mathbf{Q}$

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

Input : web graph $\mathcal{G}(V, E)$
Output: permutation array $\mathbf{\Pi}'$

begin
  /* 1. Initialization */
  $\mathbf{T}.empty()$ /* Create an empty queue $\mathbf{T}$ */
  $\mathbf{\Pi} \leftarrow \varnothing$ /* Create an permutation array $\mathbf{\Pi}$ */
  /* 2. Determination of a starting node with the minimum degree */
  foreach $a' \in \arg\min_{a \in V - \mathbf{\Pi}} |N(a)|$ **do**
    $\mathbf{\Pi} \leftarrow \mathbf{\Pi} \bigcup \{a'\}$
    Sort the nodes in $N(a')$ by degrees in ascending order
    $\mathbf{T}.enqueue(N(a'))$
    /* 3. Main loop */
    while $\mathbf{T} \neq \varnothing$ **do**
      $b \leftarrow \mathbf{T}.dequeue()$
      if $b \in \mathbf{\Pi}$ **then** continue
      $\mathbf{\Pi} \leftarrow \mathbf{\Pi} \bigcup \{b\}$
      Sort the nodes in $N(b)$ by degrees in ascending order
      $\mathbf{T}.enqueue(N(b))$
  /* 4. Reverse ordering */
  for $i \leftarrow 1 : n$ **do**
    $\mathbf{\Pi}'(n+1-i) \leftarrow \mathbf{\Pi}(i)$
  return $\mathbf{\Pi}'$

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

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

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

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

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

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

so that

$$\begin{cases} \mathbf{S} = \pi^{-1}\left(\hat{\mathbf{S}}\right) \\ \hat{\mathbf{S}} = c \cdot \pi(\mathbf{Q}) \cdot \hat{\mathbf{S}} \cdot \pi(\mathbf{Q})^T \bigvee \mathbf{I}_n \end{cases} \tag{7}$$

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.

For the SimRank computation to be I/O efficient, $\mathbf{Q}$ needs to be preordered during the precomputation. We first determine the permutation matrix $\mathbf{\Theta}$ produced by RCM Algorithm 2, for which $\pi(\mathbf{Q}) = \mathbf{\Theta} \cdot \mathbf{Q} \cdot \mathbf{\Theta}^T$ has a smaller bandwidth. Then based on equation (7), 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 $\mathbf{S}^{(k)} = \pi^{-1}\left(\hat{\mathbf{S}}^{(k)}\right) = \mathbf{\Theta}^T \cdot \hat{\mathbf{S}} \cdot \mathbf{\Theta}$.

C. SOR SimRank Algorithm

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 $\mathbf{Q} = (q_{i,j})_{n \times n}$, $\mathbf{S}^{(k)} = \left(s_1{}^{(k)}\ s_2{}^{(k)}\ \cdots\ s_n{}^{(k)}\right)$, and $s_i{}^{(k)}$ denotes the i-th column vector of matrix $\mathbf{S}^{(k)}$. For each $s_i{}^{(k)}$ $(i = 1, 2, \cdots, n)$, we can write (6) in 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

$$s_i^{GS\,(k+1)} = c \cdot \mathbf{Q} \cdot \left(\sum_{j<i} q_{i,j} \cdot s_j{}^{(k)} + \sum_{j>i} q_{i,j} \cdot s_j{}^{(k+1)}\right) \bigvee \mathbf{I}_n \tag{8}$$

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

$$\begin{aligned} s_i^{SOR\,(k+1)} &= s_i^{SOR\,(k)} + \omega\left(s_i^{GS\,(k+1)} - s_i^{SOR\,(k)}\right) \\ &= (1-\omega)\, s_i^{SOR\,(k)} + \omega \cdot s_i^{GS\,(k+1)} \end{aligned} \tag{9}$$

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 (8) back into the above equation to get

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

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 [12] 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. For our purpose, we take the optimal value $\omega = 1.3$, which gives a significant improvement in the convergence rate of the existing technique [5].

Algorithm 3: SOR_SimRank_Iteration $(\mathcal{G}_{\mathbf{Q}}, \varepsilon, c, \omega)$

Input : weh graph $\mathcal{G}_{\mathbf{Q}}$ with the transpose of column-normalized adjacency matrix in CSR format $\mathbf{Q} = \langle val_{\mathbf{Q}},\ col\_idx_{\mathbf{Q}},\ row\_ptr_{\mathbf{Q}} \rangle \in \mathbb{R}^{n \times n}$, error of accuracy ε, decay factor c, relaxation factor ω

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

1 **begin**
2 Initialize $\tilde{\mathbf{S}} \leftarrow \mathbf{0}, \quad \mathbf{S} \leftarrow \mathbf{I}, \quad k \leftarrow 0$
3 Initialize $\pi \leftarrow$ `ExtendedRCM` $\left(\mathcal{G}_{\mathbf{Q}+\mathbf{Q}^T}\right)$
4 **while** $\left(\left\|\hat{\mathbf{S}} - \tilde{\mathbf{S}}\right\| \geq \varepsilon\right)$ **do**
5 **for** $i \leftarrow 1 : n$ **do**
6 Initialize $\mathbf{v} \leftarrow \mathbf{0}$
7 Initialize $j \leftarrow row\_ptr_{\mathbf{Q}}\,(i)$
8 **while** $(j \leq row\_ptr_{\mathbf{Q}}\,(i+1) - 1$ && $col\_idx_{\mathbf{Q}}\,(j) \leq i)$ **do**
9 Set $\mathbf{v} \leftarrow \mathbf{v} + val_{\mathbf{Q}}\,(j) \cdot \tilde{\mathbf{S}}\left(:, col\_idx_{\mathbf{Q}}\,(j)\right)$
10 Set $j \leftarrow j + 1$
11 Set $r \leftarrow j$
12 **if** $i = 0$ **then**
13 Set $\hat{\mathbf{S}} \leftarrow (1-\omega) \cdot \hat{\mathbf{S}} + \omega \cdot \tilde{\mathbf{S}}$
14 **for** $j \leftarrow r : row\_ptr_{\mathbf{Q}}\,(i+1) - 1$ **do**
15 Set $\mathbf{v} \leftarrow \mathbf{v} + val_{\mathbf{Q}}\,(j) \cdot \hat{\mathbf{S}}\left(:, col\_idx_{\mathbf{Q}}\,(j)\right)$
16 Set $\tilde{\mathbf{S}}\,(:, i) \leftarrow \mathbf{0}$
17 **for** $m \leftarrow 1 : n$ **do**
18 **for** $n \leftarrow row\_ptr_{\mathbf{Q}}\,(m) : row\_ptr_{\mathbf{Q}}\,(m+1) - 1$ **do**
19 Set $\tilde{\mathbf{S}}\,(m, i) \leftarrow \tilde{\mathbf{S}}\,(m, i) + c \cdot val_{\mathbf{Q}}\,(n) \cdot \mathbf{v}\left(col\_idx_{\mathbf{Q}}\,(n)\right)$
20 Set $\tilde{\mathbf{S}}\,(i, i) \leftarrow 1$
21 Set $k \leftarrow k + 1$
22 Set $\mathbf{S} \leftarrow \pi^{-1}\left(\hat{\mathbf{S}}\right)$
23 return $\mathbf{S}, k$

Algorithm 3 describes the SOR technique applied to SimRank computation in combination with the CSR-styled matrix representation and the permuted SimRank equation proposed in the previous subsections.

- In Line 3, π can be calculated by Algorithm 2.
- In Line 8, the condition in the header of the while loop is justified by Algorithm 1.
- In Line 9-10, the iteration is justified by Equation 8.
- In Line 12-13, the expression is calculated by Equation 9.
- In Line 15-20, the iteration is justified by Equation 8.
- In Line 14,18, the condition in the header of the for loop is justified by Algorithm 1.

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 there is an inherent trade-off between computational time complexity and I/O efficiency. However, for our optimization techniques, we can achieve higher I/O efficiency while improving computational time complexity. The reason is that the I/O efficiency of our algorithm is greatly achieved by our extended RCM algorithm, which has the same time complexity as the naive RCM algorithm. According to [10], the time complexity of RCM algorithm (a) is $O(m)$ in the worst case and (b) even can be reduced to $O(n)$ with optimal implementation. In comparison, our SOR SimRank algorithm takes the time $O\left(\min\left\{n \cdot m, n^r\right\}\right)$ much greater than the RCM algorithm, so that the time consumption for achieving I/O efficiency can be ignored.

V. Experimental Evaluation

In this section, we present experimental results of our proposed algorithms. The primary purpose of the evaluation is to show that our algorithms do in fact improve the time, space and I/O efficiency of the existing technique [5]. The experiments also illustrate the effects of varying the parameters of our algorithms.

A. Experimental Setup

1) Hardware: The experiments were carried out on 2.0GHz Pentium(R) Dual-Core CPU with 2GB RAM and Windows Vista OS. We implemented our algorithms using Visual C++.

2) Data Sets: Both synthetic and real-life data sets were used in the evaluation: one for randomly generated graphs and the other for a subset of Wikipedia corpus.

Synthetic Data Sets: 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 dimensionality (web documents) from 1K to 10K and with ξ out-links on each row, where $\xi \sim$*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.

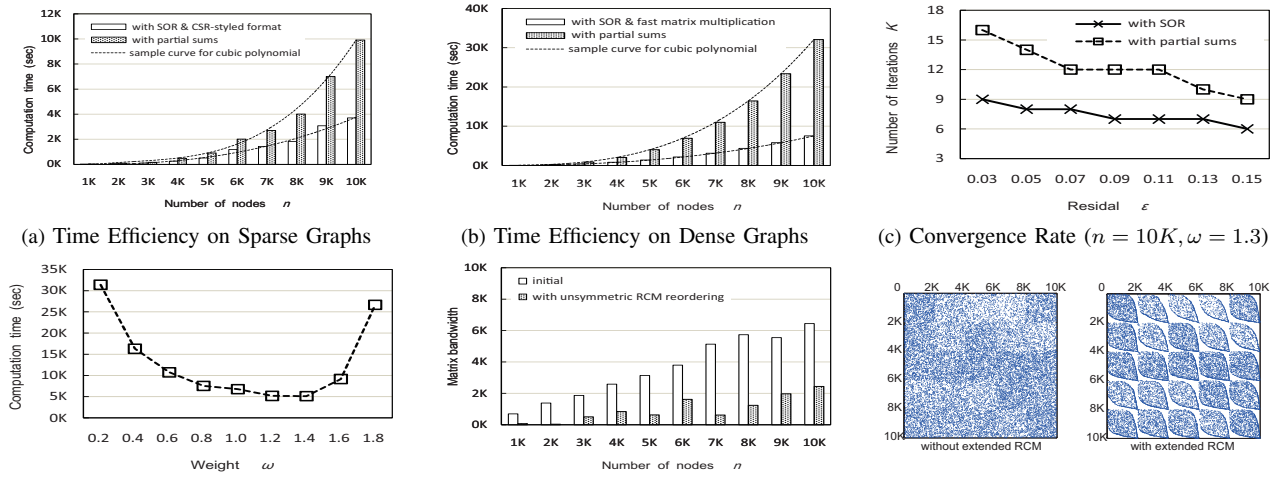
Real-life Data Sets: For practical applications, we tested our algorithms over the Wikipedia graph to investigate the relative improvement in SimRank computation time with respect to optimization techniques employed. Wikipedia is a popular online encyclopedia, containing 3.2M articles with 110M intra-wiki links in the English version (exported in October 2007). To build such a Wikipedia adjacency matrix, we chose the relationship "*a category contains an article to be a link from the category to the article*".

3) Parameter Settings: In our evaluations, for a correspondence with experiment conditions in [5], the following parameters were used as default values: the decay factor $c = 0.8$, the SOR weight value $\omega = 1.3$ and the accuracy $\varepsilon = 0.05$ (unless otherwise specified).

4) Performance Measures: For our optimization techniques, the efficiency was measured by the computation time complexity, the space requirement and the convergence rate needed to reach a certain desired SimRank accuracy.

B. Experimental Results

1) Time Efficiency: For the proposed optimization techniques, we first compare the computation time of our methods with that of the existing algorithm [5]. Figure 5(a) and 5(b) show the dynamics



(a) Time Efficiency on Sparse Graphs

(b) Time Efficiency on Dense Graphs

(c) Convergence Rate ($n = 10K, \omega = 1.3$)

(d) Relaxation Factor ($n = 10K, \varepsilon = 0.05$)

(e) I/O Efficiency on Sparse Graphs

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

Figure 5. Experimental results on SimRank computation

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 the existing technique with partial sums [5] for both sparse and dense graphs. Note that the different maximum value is chosen across the vertical axis in Figure 5(a) and 5(b). For sparse graphs, our method may reduce almost half of the computation 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] requires $O\left(n^2 \cdot d\right)$ per iteration, where d is the average node degree, resulting in $O\left(n^3\right)$ for dense graphs. For comparison, our approach requires $O\left(n \cdot m\right)$ 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 5(a), it is reasonable to see that for a given accuracy ε, our method has reduced just half of the time due to the SOR techniques accelerating the convergence rate and reducing the number of iterations to reach a desired accuracy. By contrast, for dense graphs, our method in Figure 5(b) has a significant improvement in computation time because the time consumption in [5] requires $O\left(n^3\right)$ in the dense case whilst our technique adopts the fast matrix multiplication for computing SimRank score, involving $O\left(n^r\right)$, where $r \leq \log_2 7$.

2) Convergence Rate: To investigate the correlation between the residual accuracy ε and the number of iterations K, we vary the accuracy ε from 0.03 to 0.15 over a 10K generated sparse graph. Figure 5(c) compares the convergence rate of our SOR SimRank method with the existing technique. In [5], for achieving accuracy ε, the existing algorithm requires $K = \lceil \log_c \varepsilon \rceil - 1$ iterations. For evaluating our algorithm, here we choose the relaxation factor $\omega = 1.3$ for SOR SimRank computation. 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].

Furthermore, to investigate how the computation time is influenced by the relaxation factor ω of our SOR method, we vary ω from 0 to 2. The results in Figure 5(c) indicate that the SOR computation 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 [12]. That is the reason why we choose $\omega = 1.3$ for achieving the best performance of our SOR SimRank algorithm.

3) I/O Efficiency: Now we show results of applying the extended RCM algorithms to the sparse matrix $\mathbf{Q} + \mathbf{Q}^T$ for the SimRank precomputation. Figure 5(e) presents the effect of applying our reordering Algorithm 2 to 10 generated sparse graphs. We can see that RCM does reduce the total bandwidths for these sparse matrices and can thus improve the I/O efficiency of our algorithm. In Figure 5(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.

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. We also use the CSR scheme for computing SimRank over the Wikipedia graph (a huge matrix with few non-zero elements) and it only takes up 846.6MB.

In the final experiment over the Wikipedia graph, we chose $c = 0.6, \varepsilon = 0.1$ corresponding with the evaluation conditions in [5]. We set the cache size of 128MB for Oracle Berkeley DB and kept the Wikipedia graph in the CSR format. Due to space constraints, we will focus only on the experimental result. From the time and convergence rate perspective, our evaluation shows that it takes nearly 35 hours with 5 iterations to complete the SimRank computation on one processor, whereas our method takes approximately 16 hours with only 2 iterations, thus saving almost half of the computation time. From the space perspective, our representation in CSR scheme requires 846.6MB storage space, whereas their scheme takes up nearly 2.7GB. Our results on a single machine demonstrate that our method is preferable as it demands less computation time and storage requirement, which agrees with our theoretical considerations addressed in Sect. IV.

VI. 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], [13], [14].

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* [15] and *co-citation* [16] 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 [17], PageRank [14], SimRank [1] and SimFusion [18] 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], [19], [20]. 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 work for computing SimRank determinsitically, the most efficient optimization techniques presented in [5] 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] 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] 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].

Antonellis et al. [20] 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. [18] 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:

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

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 $O(\min\{n \cdot m, n^r\})$, 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 $O(n+m)$ 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 [12].

VII. Conclusions

This paper investigated the optimization issues for SimRank computation. We first 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 $O(n+m)$, whereas a fast matrix multiplication for dense graph is used for improving the time complex from $O\left(n^2 \cdot d\right)$ to $O(\min\{n \cdot m, n^r\})$, 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. 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.

Acknowledgment

Part of the work done while the first author was a joint PhD candidate at Prof. Xuemin Lin's Research Groups and Laboratories. Prof. Xuemin Lin was supported by three ARC DPs (DP0666428, DP0881035, and DP0987557) and a Google research award.

References

[1] G. Jeh and J. Widom, "Simrank: a measure of structural-context similarity," in *KDD*, 2002.

[2] A. Pathak, S. Chakrabarti, and M. S. Gupta, "Index design for dynamic personalized pagerank," in *ICDE*, 2008.

[3] D. Fogaras and B. Rácz, "Scaling link-based similarity search," in *WWW*, 2005.

[4] D. Fogaras and B. Racz, "A scalable randomized method to compute link-based similarity rank on the web graph," in *EDBT Workshops*, 2004.

[5] D. Lizorkin, P. Velikhov, M. Grinev, and D. Turdakov, "Accuracy estimate and optimization techniques for simrank computation," *PVLDB*, vol. 1, no. 1, 2008.

[6] E. F. D'Azevedo, M. R. Fahey, and R. T. Mills, "Vectorized sparse matrix multiply for compressed row storage format," in *International Conference on Computational Science (1)*, 2005.

[7] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *J. Symb. Comput.*, vol. 9, no. 3, 1990.

[8] J. Cohen and M. S. Roth, "On the implementation of strassen's fast multiplication algorithm," *Acta Inf.*, vol. 6, 1976.

[9] D. Coppersmith and S. Winograd, "On the asymptotic complexity of matrix multiplication," *SIAM J. Comput.*, vol. 11, no. 3, 1982.

[10] W. M. Chan and A. George, "A linear time implementation of the reverse cuthill-mckee algorithm," *BIT*, vol. 20, no. 1, 1980.

[11] A. Lim, B. Rodrigues, and F. Xiao, "Heuristics for matrix bandwidth reduction," *European Journal of Operational Research*, vol. 174, no. 1, 2006.

[12] R. Bhatia, *Matrix Analysis*. Springer, 1997.

[13] J. U. Quevedo and S.-H. S. Huang, "Similarity among web pages based on their link structure," in *IKE*, 2003.

[14] R. M. Lawrence Page, Sergey brin and T. Winograd, "The pagerank citation ranking bringing order to the web," 1998, technial report.

[15] B. H. Weinberg, "Bibliographic coupling: A review," *Information Storage and Retrieval*, vol. 10, no. 5-6, 1974.

[16] D. T. Wijaya and S. Bressan, "Clustering web documents using co-citation, coupling, incoming, and outgoing hyperlinks: a comparative performance analysis of algorithms," *IJWIS*, vol. 2, no. 2, 2006.

[17] A. O. Mendelzon, "Review - authoritative sources in a hyperlinked environment," *ACM SIGMOD Digital Review*, vol. 1, 2000.

[18] W. Xi, E. A. Fox, W. Fan, B. Zhang, Z. Chen, J. Yan, and D. Zhuang, "Simfusion: measuring similarity using unified relationship matrix," in *SIGIR*, 2005.

[19] Y. Cai, P. Li, H. Liu, J. He, and X. Du, "S-simrank: Combining content and link information to cluster papers effectively and efficiently," in *ADMA*, 2008.

[20] I. Antonellis, H. Garcia-Molina, and C.-C. Chang, "Simrank++: query rewriting through link analysis of the click graph," *PVLDB*, vol. 1, no. 1, 2008.