On the Efficiency of Estimating Penetrating Rank on Large Graphs

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

[1] University of New South Wales & NICTA, Australia
{weirenyu,lxue,zhangw}@cse.unsw.edu.au
[2] Donghua University, China
lejiajin@dhu.edu.cn

Abstract. P-Rank (Penetrating Rank) has been suggested as a useful measure of structural similarity that takes account of both incoming and outgoing edges in ubiquitous networks. Existing work often utilizes memoization to compute P-Rank similarity in an iterative fashion, which requires *cubic* time in the worst case. Besides, previous methods mainly focus on the deterministic computation of P-Rank, but lack the probabilistic framework that scales well for large graphs. In this paper, we propose two efficient algorithms for computing P-Rank on large graphs. The first observation is that a large body of objects in a real graph usually share similar neighborhood structures. By merging such objects with an explicit low-rank factorization, we devise a *deterministic* algorithm to compute P-Rank in *quadratic* time. The second observation is that by converting the iterative form of P-Rank into a matrix power series form, we can leverage the random sampling approach to *probabilistically* compute P-Rank in *linear* time with provable accuracy guarantees. The empirical results on both real and synthetic datasets show that our approaches achieve high time efficiency with controlled error and outperform the baseline algorithms by at least one order of magnitude.

1 Introduction

Structural similarity search that ranks objects based on graph hyperlinks is a major tool in the fields of data mining. This problem is also known as link-based analysis, and it has become popularized in a plethora of applications, such as nearest neighbor search [26], graph clustering [27], and collaborative filtering [9]. For example, Figure 1 depicts a recommender system, in which person (A) and (B) purchase itemsets {egg, pancake, sugar} and {egg, pancake, flour}, respectively. We want to identify similar users and similar items.

Existing link-based approaches usually take advantage of graph structures to measure similarity between vertices. Each object (*e.g.,* person, or item) can be regarded as a vertex, and a hyperlink (*e.g.,* purchase relationship) as a directed edge in a graph. Then a scoring rule is defined to compute similarity between vertices. Consider the well-known SimRank scoring rule [18] "two vertices are similar if they are referenced (have incoming edges) from similar vertices" in Figure 1. We can see that the items sugar and egg are similar as they are purchased by the same person (A). In spite of its worldwide popularity [1, 4, 6, 18, 24, 27], SimRank has the "limited information problem" — it only takes incoming edges into account while ignoring outgoing links [26]. For instance, person (A) and (B) have the SimRank score zero as they have no incoming edges. This

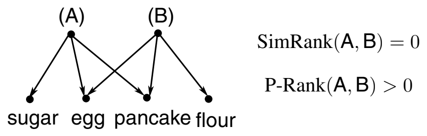


Fig. 1. Purchase Relationship in a Recommender System

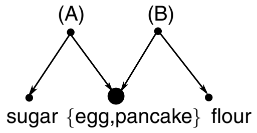


Fig. 2. Merge the nodes having the same neighborhood

is counter-intuitive since the similarity between person (A) and (B) also depends on the similarity of their purchased products. To address this issue, Zhao *et al.* [26] proposed to use P-Rank similarity to effectively incorporate both in- and out-links. Since then, P-Rank has attracted growing attention (*e.g.,* [2, 13, 17, 21]), and it can be widely used to any ubiquitous domain where SimRank is applicable, such as social graphs [2], and publication networks [13]. The intuition behind P-Rank is an improved version of SimRank: "two distinct vertices are similar if (a) they are referenced by similar vertices, and (b) they reference similar vertices". In contrast with SimRank, P-Rank is a general framework for exploiting structural similarity of the Web, and has the extra benefit of taking account of both incoming and outgoing links. As an example in Figure 1, person (A) and (B) are similar in the context of P-Rank.

Nonetheless, existing studies on P-Rank have the following problems. Firstly, it is rather time-consuming to iteratively compute P-Rank on large graphs. Previous methods [17, 26] deploy a fixed-point iterative paradigm for P-Rank computation. While these methods often attain good accuracy, they do not scale well for large graphs since they need to enumerate all n^2 vertex-pairs per iteration if there are n vertices in a graph. The most efficient existing technique using memoization for SimRank computation [18] can be applied to P-Rank in a similar fashion, but still needs $O(Kn^3)$ time. The recent dramatic increase in network scale highlights the need for a new method to handle large volumes of P-Rank computation with low time complexity and high accuracy.

Secondly, it is a big challenge to estimate the error when approximation approaches are leveraged for computing P-Rank. Zhao *et al.* [26] proposed the radius- and category-based pruning techniques to improve the computation of P-Rank to $O(Kd^2n^2)$, with d being the average degree in a graph. However, this heuristic method does not warrant the accuracy of pruning results. For certain applications like ad-hoc top-k nearest neighbor search, fast speed is far more important than accuracy; it is desirable to sacrifice a little accuracy (with controlled error) for accelerating the computation.

In this paper, we address the optimization issue of P-Rank. We have an observation that many real-world graphs are low rank and sparse, such as the Web [18], bibliographic network [9], and social graph [14]. Based on this, we devise two efficient algorithms (a) to deterministically compute P-Rank in an off-line fashion, and (b) to probabilistically estimate P-Rank with controlled error in an on-line fashion. For deterministic computation, we observe that a large body of vertices in a real graph usually have the similar neighborhood structures, and some may even share the same common neighborhoods (*e.g.,* we notice in Figure 1 that the products egg and pancake are purchased by the same users—their neighborhoods are identical. Therefore, we can merge egg and pancake into one vertex, as illustrated in Figure 2). Due to these redundancy, we have an opportunity to "merge" these similar vertices into one vertex. To this end, we utilize a low-rank factorization to eliminate such redundancy. However, it is hard to develop an efficient algorithm and give an error estimate for low-rank approximation. For probabilistic computation, we notice that the iterative form of P-Rank can be characterized as a matrix power series form. In light of this, we adopt a random sampling approach to further improve the computation of P-Rank in linear time with provable guarantee.

Contributions. In summary, we make the following contributions.

(1) We characterize the P-Rank similarity as two equivalent matrix forms: The matrix inversion form of P-Rank lays a foundation for deterministic optimization, and the power series form for probabilistic computation. (Section 3).

(2) We observe that many vertices in a real graph have neighborhood structure redundancy. By eliminating the redundancy, we devise an efficient deterministic algorithm based on the matrix inversion form of P-Rank to optimize the P-Rank computation, yielding quadratic-time in the number of vertices (Section 4).

(3) We base a sampling approach on the power series form of P-Rank to further speed up the computation of P-Rank probabilistically, achieving linear-time with controlled accuracy (Section 5).

(4) Using both real and synthetic datasets, we empirically show that (a) our deterministic algorithm outperforms the baseline algorithms by almost one order of magnitude in time and (b) our probabilistic algorithm runs much faster than the deterministic method with controlled error (Section 6).

2 Preliminaries

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a directed graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. For a vertex $u \in \mathcal{V}$, we denote by $\mathcal{I}(u)$ and $\mathcal{O}(u)$ the in-neighbor set and out-neighbor set of u respectively, $|\mathcal{I}(u)|$ and $|\mathcal{O}(u)|$ the cardinalities of $\mathcal{I}(u)$ and $\mathcal{O}(u)$ respectively.

The *P-Rank similarity* between vertices u and v, denoted by $s(u, v)$, is defined as (a) $s(u, u) = 1$; (b) when $u \neq v$,

$$s(u,v) = \underbrace{\frac{\lambda \cdot C_{\text{in}}}{|\mathcal{I}(u)|\,|\mathcal{I}(v)|} \sum_{i\in\mathcal{I}(u)} \sum_{j\in\mathcal{I}(v)} s(i,j)}_{\text{in-link part}} + \underbrace{\frac{(1-\lambda)\cdot C_{\text{out}}}{|\mathcal{O}(u)|\,|\mathcal{O}(v)|} \sum_{i\in\mathcal{O}(u)} \sum_{j\in\mathcal{O}(v)} s(i,j)}_{\text{out-link part}}, \quad (1)$$

where $\lambda \in [0, 1]$ is a *weighting factor* balancing the contribution of in- and out-links; C_{in} and $C_{\text{out}} \in (0, 1)$ are *damping factors* for in- and out-link directions, respectively.

Note that either $\mathcal{I}(\cdot)$ or $\mathcal{O}(\cdot)$ can be an empty set. To prevent division by zero, the definition in Eq.(1) also assumes that (a) in-link part$= 0$ if $\mathcal{I}(u)$ or $\mathcal{I}(v) = \varnothing$, and (b) out-link part$= 0$ if $\mathcal{O}(u)$ or $\mathcal{O}(v) = \varnothing$.

P-Rank Matrix Formula. Let $\mathbf{Q}$ be the backward transition matrix of $\mathcal{G}$, whose entry $q_{i,j} = 1/|\mathcal{I}(i)|$ if $\exists$ an edge $(j, i) \in \mathcal{E}$, and 0 otherwise; and let $\mathbf{P}$ be the forward transition matrix of $\mathcal{G}$, whose entry $p_{i,j} = 1/|\mathcal{O}(i)|$ if $\exists$ an edge $(i, j) \in \mathcal{E}$, and 0 otherwise. By virtue of our prior work [17], the P-Rank equation (1) then equivalently takes the simple form

$$\mathbf{S} = \lambda \cdot C_{\text{in}} \cdot \mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T + (1-\lambda) \cdot C_{\text{out}} \cdot \mathbf{P} \cdot \mathbf{S} \cdot \mathbf{P}^T + \mathbf{I}_n, \quad (2)$$

where $\mathbf{S}$ is the similarity matrix whose entry $s_{i,j}$ equals the P-Rank score $s(i,j)$, and $\mathbf{I}_n$ is the $n \times n$ identity matrix [1], ensuring that each vertex is maximally similar to itself.

3 Two Forms of P-Rank Solution

In this section, we present two closed-form expressions of the P-Rank similarity matrix $\mathbf{S}$, with the aim to optimize P-Rank computation in the next sections.

Our key observation is that the P-Rank matrix formula Eq.(2) is a linear equation. This linearity can be made more explicit by utilizing the matrix-to-vector operator that converts a matrix into a vector by staking its columns one by one. This operator, denoted vec, satisfies the basic property $vec(\mathbf{A} \cdot \mathbf{X} \cdot \mathbf{B}) = (\mathbf{B}^T \otimes \mathbf{A}) \cdot vec(\mathbf{X})$ in which $\otimes$ denotes the Kronecker product. (For a proof of this property, see Theorem 13.26 in [12, p.147].) Applying this property to Eq.(2) we immediately obtain $\mathbf{x} = \mathbf{M} \cdot \mathbf{x} + \mathbf{b}$, where $\mathbf{x} = vec(\mathbf{s})$, $\mathbf{M} = \lambda \cdot C_{\text{in}} \cdot (\mathbf{Q} \otimes \mathbf{Q}) + (1-\lambda) \cdot C_{\text{out}} \cdot (\mathbf{P} \otimes \mathbf{P})$, and $\mathbf{b} = vec(\mathbf{I}_n)$. The recursive form of $\mathbf{x}$ naturally leads itself into a power series form $\mathbf{x} = \sum_{i=0}^{\infty} \mathbf{M}^i \cdot \mathbf{b}$. Combining this observation with Eq.(2), we deduce the following lemma.

Lemma 1 (Power Series Form). *The P-Rank matrix formula Eq.*(2) *has the following algebraic solution*

$$vec(\mathbf{S}) = \sum_{i=0}^{\infty} \left[\lambda \cdot C_{in} \cdot (\mathbf{Q} \otimes \mathbf{Q}) + (1-\lambda) \cdot C_{out} \cdot (\mathbf{P} \otimes \mathbf{P})\right]^i \cdot vec(\mathbf{I}_n). \tag{3}$$

Lemma 1 describes the power series form of the P-Rank similarity. This result will be used to justify our random sampling approach for estimating P-Rank (in Section 5). One caveat is that the convergence of $\sum_i \mathbf{M}^i \cdot \mathbf{b}$ is guaranteed only if $\|\mathbf{M}\|_\infty < 1$ (see [7, p.301]), where $\| \star \|_\infty$ is the ∞-matrix norm [2]. This is true for Eq.(3) because $\lambda \in [0,1]$ and $C_{\text{in}}, C_{\text{out}} \in (0,1)$ imply that

$$\|\mathbf{M}\|_\infty \le \lambda \cdot \overbrace{C_{\text{in}}}^{<1} \cdot \overbrace{\|\mathbf{Q} \otimes \mathbf{Q}\|_\infty}^{=1} + (1-\lambda) \cdot \overbrace{C_{\text{out}}}^{<1} \cdot \overbrace{\|\mathbf{P} \otimes \mathbf{P}\|_\infty}^{=1} < \lambda + (1-\lambda) = 1.$$

Another explicit expression for the P-Rank similarity comes from the observation that $\sum_i \mathbf{M}^i = (\mathbf{I} - \mathbf{M})^{-1}$ whenever $\|\mathbf{M}\|_\infty < 1$. Applying this observation to Lemma 1 yields the matrix inversion form of the P-Rank similarity.

Lemma 2 (Matrix Inversion Form). *The P-Rank similarity matrix* $\mathbf{S}$ *in Eq.*(2) *can be rewritten as*

$$vec(\mathbf{S}) = \left[\mathbf{I}_{n^2} - \lambda C_{in} (\mathbf{Q} \otimes \mathbf{Q}) - (1-\lambda) C_{out} (\mathbf{P} \otimes \mathbf{P})\right]^{-1} \cdot vec(\mathbf{I}_n). \tag{4}$$

The utility of Lemma 2 lies in the observation that computing $\mathbf{S}$ can be converted into a matrix inversion computation. Due to the huge size, the straightforward way of computing such matrix inversion is prohibitively expensive; nevertheless, optimization techniques in the next section will significantly improve the computational efficiency.

[1] Throughout the paper, we denote by n the number of vertices in $\mathcal{G}$.

[2] The ∞-matrix norm is simply the maximum absolute row sum of the matrix.

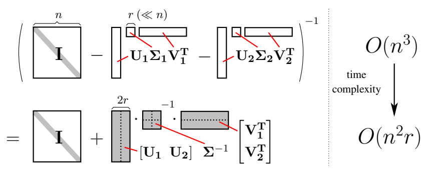


Fig. 3. Low-rank update of matrix inversion

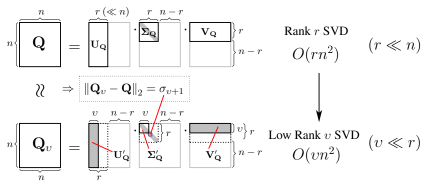


Fig. 4. Reduced SVD Process

4 An Algorithm for P-Rank Deterministic Computation

In light of the matrix inversion form in Lemma 2, we now focus on deterministic optimization of P-Rank computation. In this section, we show the following result.

Theorem 1. *For any graph $\mathcal{G}$, given a low-rank υ $(\leq r)$, it is in $O\left(\upsilon n^2 + \upsilon^6\right)$ time and $O(\upsilon \cdot \max\{\upsilon^3, n\})$ space to compute P-Rank similarity up to an additive error of $\epsilon_\upsilon \leq \frac{\lambda C_{in}\sigma_1\sigma_{\upsilon+1}+(1-\lambda)C_{out}\bar{\sigma}_1\bar{\sigma}_{\upsilon+1}}{1-\lambda C_{in}-(1-\lambda)C_{out}} r$, where r $(\ll n)$ is the rank of the adjacency matrix, σ_i and $\bar{\sigma}_i$ $(i = 1, \upsilon+1)$ are the i-th largest singular values of $\mathbf{Q}$ and $\mathbf{P}$ respectively.*

In particular, setting $\upsilon = r$ gives the following corollary.

Corollary 1. *The exact P-Rank similarity can be solvable in $O\left(rn^2 + r^6\right)$ time and $O(r \cdot \max\{r^3, n\})$ space.*

(A sketch proof of Theorem 1 and Corollary 1 will be provided after some discussions. See [23] for a full version of proof.)

The key observation behind P-Rank optimization is that vertices in a real graph usually have a large number of common neighborhoods (*e.g.,* many users often have the similar preferences in a recommender system). Hence, r is typically much smaller than n in practice. The main idea is (a) to devise a rank-r update formula for efficiently computing the matrix inversion in Eq.(4), and (b) to use a rank-r factorization for merging the vertices that have the same neighborhoods into one vertex.

To prove Theorem 1, we first devise a low-rank update formula of matrix inversion. We then present an algorithm for P-Rank computation with the desired properties.

Lemma 3. *Let $\mathbf{I}_n$ be an $n \times n$ identity matrix, $\mathbf{U}_i$ and $\mathbf{V}_i$ be $n \times r$ matrices $(r \ll n)$, and $\mathbf{\Sigma}_i$ be $r \times r$ matrices $(i = 1, 2)$. Then the following identity holds.*

$$\left(\mathbf{I}_n - \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T - \mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}_2^T\right)^{-1} = \mathbf{I}_n + \left(\mathbf{U}_1 \; \mathbf{U}_2\right)\begin{pmatrix} \mathbf{\Sigma}_1{}^{-1} - \mathbf{V}_1^T\mathbf{U}_1 & -\mathbf{V}_1^T\mathbf{U}_2 \\ -\mathbf{V}_2^T\mathbf{U}_1 & \mathbf{\Sigma}_2{}^{-1} - \mathbf{V}_2^T\mathbf{U}_2 \end{pmatrix}^{-1}\begin{pmatrix} \mathbf{V}_1^T \\ \mathbf{V}_2^T \end{pmatrix} \quad (5)$$

(For the interest of space, please refer to [23] for a detailed proof. Lemma 3 is an extension of *the Woodbury matrix identity* [12, p.48].)

As opposed to $O\left(n^3\right)$-time of the conventional matrix inversion [7], Lemma 3 provides an efficient way of computing $\left(\mathbf{I}_n - \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T - \mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}_2^T\right)^{-1}$ in $O(n^2r + r^2n + r^3)$ time $(r \ll n)$ via the RHS of Eq.(5). As depicted in Figure 3, the performance gain is achieved by the observation that $\mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T$ and $\mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}_2^T$ are low rank.

One immediate consequence of Lemma 3 is the optimization of the P-Rank matrix inversion form. We have an observation that most real graphs are low rank (*e.g.,* the web graph [18], bibliographic network [9], who-trusts-whom social network [14]). By applying a *reduced singular value decomposition* [19] [3] (as depicted in Figure 4), $\lambda C_{\text{in}} (\mathbf{Q} \otimes \mathbf{Q})$ and $(1-\lambda) C_{\text{out}} (\mathbf{P} \otimes \mathbf{P})$ in Eq.(4) can be factorized into the low-rank form of $\mathbf{U}_1 \mathbf{\Sigma}_1 \mathbf{V}_1^T$ and $\mathbf{U}_2 \mathbf{\Sigma}_2 \mathbf{V}_2^T$, respectively. Then combining Lemma 3, we have

$$vec(\mathbf{S}) = \begin{pmatrix} \tilde{\mathbf{U}}_{\mathbf{Q}} & \tilde{\mathbf{U}}_{\mathbf{P}} \end{pmatrix} \mathbf{\Sigma} \begin{pmatrix} \tilde{\mathbf{V}}_{\mathbf{Q}}^T \\ \tilde{\mathbf{V}}_{\mathbf{P}}^T \end{pmatrix} vec(\mathbf{I}_n) + vec(\mathbf{I}_n) \text{ with } \mathbf{\Sigma} = \begin{pmatrix} \frac{1}{\lambda C_{\text{in}}} \tilde{\mathbf{\Sigma}}_{\mathbf{Q}}^{-1} - \tilde{\mathbf{V}}_{\mathbf{Q}}^T \tilde{\mathbf{U}}_{\mathbf{Q}} & -\tilde{\mathbf{V}}_{\mathbf{Q}}^T \tilde{\mathbf{U}}_{\mathbf{P}} \\ -\tilde{\mathbf{V}}_{\mathbf{P}}^T \tilde{\mathbf{U}}_{\mathbf{Q}} & \frac{1}{(1-\lambda) C_{\text{out}}} \tilde{\mathbf{\Sigma}}_{\mathbf{P}}^{-1} - \tilde{\mathbf{V}}_{\mathbf{P}}^T \tilde{\mathbf{U}}_{\mathbf{P}} \end{pmatrix}^{-1},$$

where a tilde denotes the self-Kronecker product of a matrix, *e.g.,* $\tilde{\mathbf{U}}_{\mathbf{Q}} = \mathbf{U}_{\mathbf{Q}} \otimes \mathbf{U}_{\mathbf{Q}}$. Due to $\mathbf{\Sigma}$ small size, the efficiency of computing P-Rank can be greatly improved.

We next provide an algorithm for P-Rank computation, denoted by DE P-Rank.

Algorithm. In Algorithm 1, given $\mathcal{G}$, λ, C_{in} , C_{out}, and a low rank v (an optional parameter with a default value being the rank r of adjacency matrix), DE P-Rank outputs the exact $\mathbf{S}$ if $v = r$, or the approximate $\mathbf{S}$ with an error ϵ_v if $v < r$.

Some notations in the algorithm are elaborated below. (a) `RowNorm` $(\mathbf{A})$ returns a matrix by normalizing each nonzero row of $\mathbf{A}$. (b) `Rank` $(\mathbf{A})$ returns the rank of $\mathbf{A}$. (c) `RSVD` $(\mathbf{Q}, v)$ returns a low-rank v factorization of $\mathbf{Q}$ (see Figure 4). (d) `Reshape(`$\mathbf{v}, v$`)` returns an $v \times v$ matrix $\mathbf{V}$ such that $vec(\mathbf{V}) = \mathbf{v}$.

The algorithm works as follows. (a) It first initializes the adjacency matrix $\mathbf{A}$ (line 1), and computes $\mathbf{Q}$ and $\mathbf{P}$ (line 2). v is set to `Rank`$(\mathbf{A})$ if the low rank v is not specified (line 3). (b) It then utilizes `RSVD` () to decompose $\mathbf{Q}$ and $\mathbf{P}$ into $\mathbf{U}_{\mathbf{Q}} \mathbf{\Sigma}_{\mathbf{Q}} \mathbf{V}_{\mathbf{Q}}^T$ and $\mathbf{U}_{\mathbf{P}} \mathbf{\Sigma}_{\mathbf{P}} \mathbf{V}_{\mathbf{P}}^T$, respectively (line 4). In light of these matrices together with the self-Kronecker products, two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ can be obtained (lines 5-7). The error estimate ϵ_v is also computed if $v <$ `Rank`$(\mathbf{A})$ (line 8). (c) Utilizing $\mathbf{v}_1$ and $\mathbf{v}_2$, the matrix $\mathbf{S}$ can be derived, which is returned as the P-Rank similarity (lines 9-11).

Example. Figures 5(a) and 5(b) show how DE P-Rank computes P-Rank in a heterogenous graph $\mathcal{G}_0$ and a homogeneous $\mathcal{G}_1$, respectively. In $\mathcal{G}_0$, there are two types of entities : person (A) and (B) purchase the items sugar, egg, flour. In $\mathcal{G}_1$, each vertex denotes a paper, and each edge a citation. For these graphs, given $C_{\text{in}} = 0.4, C_{\text{out}} = 0.6, \lambda = 0.5$, DE P-Rank first computes $\mathbf{Q}$ and $\mathbf{P}$. Since v is not specified, it is set to `Rank`$(\mathbf{A})$. Then $\mathbf{Q}$ and $\mathbf{P}$ are decomposed into small matrices that can be used for computing $\left(\begin{smallmatrix} \mathbf{\Sigma}_{11} & \mathbf{\Sigma}_{12} \\ \mathbf{\Sigma}_{21} & \mathbf{\Sigma}_{22} \end{smallmatrix}\right)$ and $\mathbf{v}_1, \mathbf{v}_2$. Finally, DE P-Rank computes the exact $\mathbf{S}$.

To complete the proof of Theorem 1, we next show that the algorithm DE P-Rank (1) correctly computes the similarity values; (2) it has the time complexity bound stated in Theorem 1; (3) when $v \in [\frac{1}{2}r, r]$, the error ϵ_v (line 8) is acceptable in practice.

(Due to space limitations, please refer to [23] for detailed analysis.)

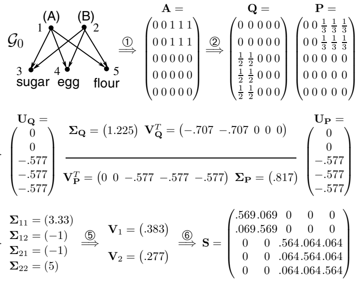
(1) Correctness. The algorithm returns *exactly* the same similarity as Eq.(4) when $v = r$; and it returns the low-rank v *approximate* similarity with an error ϵ_v stated in Theorem 1 when $v < r$.

(2) Running Time. The algorithm consists of three phases: pre-processing (lines 1-3), similarity computation (lines 4-8), and result collection (lines 9-11). One can verify

[3] Given an matrix $\mathbf{X}$ (with its rank r) and an integer v $(\leq r)$, the *reduced singular value decomposition* of $\mathbf{X}$ is the factorization $\mathbf{X}_v = \mathbf{U}_v \cdot \mathbf{\Sigma}_v \cdot \mathbf{V}_v{}^T$ *s.t.* $\|\mathbf{X} - \mathbf{X}_v\|_2 = \sigma_{v+1}$ is minimal, where $\mathbf{U}_v$ and $\mathbf{V}_v$ are $n \times v$ column orthonormal matrices, and $\mathbf{\Sigma} \triangleq diag(\sigma_1, \sigma_2, \cdots, \sigma_v)$ is an $v \times v$ diagonal matrix whose entries are the singular values of $\mathbf{X}$.

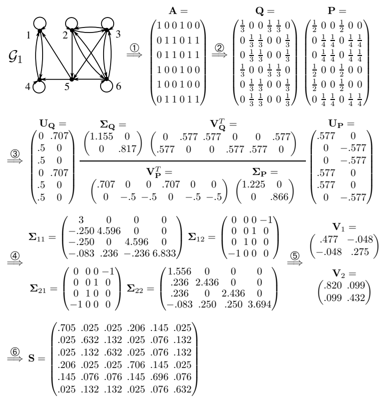
Algorithm 1: DE P-Rank

```
Input : web graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$,
        weighting factor $\lambda$,
        damping factors $C_{\text{in}}$ and $C_{\text{out}}$,
        low rank $v$.
Output: similarity matrix $\mathbf{S}$, and
        approximation error $\epsilon_v$.
1  initialize the adjacency matrix $\mathbf{A}$ of $\mathcal{G}$.
2  $\mathbf{Q} \leftarrow \texttt{RowNorm}(\mathbf{A}^T)$, $\mathbf{P} \leftarrow \texttt{RowNorm}(\mathbf{A})$.
3  if $v$ is empty then $v \leftarrow \texttt{Rank}(\mathbf{A})$.
4  $[\mathbf{U_Q}, \mathbf{\Sigma_Q}, \mathbf{V_Q}; \sigma_1, \sigma_{v+1}] \leftarrow \texttt{RSVD}(\mathbf{Q}, v)$,
   $[\mathbf{U_P}, \mathbf{\Sigma_P}, \mathbf{V_P}; \bar{\sigma}_1, \bar{\sigma}_{v+1}] \leftarrow \texttt{RSVD}(\mathbf{P}, v)$.
5  compute the small auxiliary matrices:
   $\mathbf{\Lambda_{Q,Q}} = \mathbf{V_Q}^T \cdot \mathbf{U_Q}$, $\mathbf{\Lambda_{P,P}} = \mathbf{V_P}^T \cdot \mathbf{U_P}$,
   $\mathbf{\Lambda_{P,Q}} = \mathbf{V_P}^T \cdot \mathbf{U_Q}$, $\mathbf{\Lambda_{Q,P}} = \mathbf{V_Q}^T \cdot \mathbf{U_P}$,
   $\mathbf{\Lambda_Q} = \mathbf{\Sigma_Q}^{-1}$, $\mathbf{\Lambda_P} = \mathbf{\Sigma_P}^{-1}$.
6  compute the four blocks of the matrix $\mathbf{\Sigma}$ :
   $\mathbf{\Sigma}_{11} \leftarrow \frac{1}{\lambda C_{\text{in}}} \mathbf{\Lambda_Q} \otimes \mathbf{\Lambda_Q} - \mathbf{\Lambda_{Q,Q}} \otimes \mathbf{\Lambda_{Q,Q}}$,
   $\mathbf{\Sigma}_{12} \leftarrow -\mathbf{\Lambda_{Q,P}} \otimes \mathbf{\Lambda_{Q,P}}$,
   $\mathbf{\Sigma}_{22} \leftarrow \frac{1}{(1-\lambda) C_{\text{out}}} \mathbf{\Lambda_P} \otimes \mathbf{\Lambda_P} - \mathbf{\Lambda_{P,P}} \otimes \mathbf{\Lambda_{P,P}}$,
   $\mathbf{\Sigma}_{21} \leftarrow -\mathbf{\Lambda_{P,Q}} \otimes \mathbf{\Lambda_{P,Q}}$.
7  compute the P-Rank similarity $\mathbf{S}$ :
   $\begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{pmatrix} \leftarrow \begin{pmatrix} \mathbf{\Sigma}_{11} & \mathbf{\Sigma}_{12} \\ \mathbf{\Sigma}_{21} & \mathbf{\Sigma}_{22} \end{pmatrix}^{-1} \begin{pmatrix} vec(\mathbf{V_Q}^T \mathbf{V_Q}) \\ vec(\mathbf{V_P}^T \mathbf{V_P}) \end{pmatrix}$.
8  if $v < \texttt{Rank}(\mathbf{A})$ then
   $\epsilon_v \leftarrow \frac{\lambda C_{\text{in}} \sigma_1 \sigma_{v+1} + (1-\lambda) C_{\text{out}} \bar{\sigma}_1 \bar{\sigma}_{v+1}}{1 - \lambda C_{\text{in}} - (1-\lambda) C_{\text{out}}} \texttt{Rank}(\mathbf{A})$
   else $\epsilon_v \leftarrow 0$.
9  $\mathbf{V}_1 \leftarrow \texttt{Reshape}(\mathbf{v}_1, v)$,
   $\mathbf{V}_2 \leftarrow \texttt{Reshape}(\mathbf{v}_2, v)$.
10 $\mathbf{S} \leftarrow (1 - \lambda C_{\text{in}} - (1-\lambda) C_{\text{out}}) \cdot$
       $(\mathbf{I}_n + \mathbf{U_Q} \mathbf{V}_1 \mathbf{U_Q}^T + \mathbf{U_P} \mathbf{V}_2 \mathbf{U_P}^T)$.
11 return $\mathbf{S}$ and $\epsilon_v$.
```



$\mathcal{G}_0$
(A)
(B)
sugar
egg
flour
A =
Q =
P =
$\mathbf{U_Q}$ =
$\mathbf{\Sigma_Q} = (1.225)$
$\mathbf{V_Q}^T = (-.707\ -.707\ 0\ 0\ 0)$
$\mathbf{V_P}^T = (0\ 0\ -.577\ -.577\ -.577)$
$\mathbf{\Sigma_P} = (.817)$
$\mathbf{U_P}$ =
$\mathbf{\Sigma}_{11} = (3.33)$
$\mathbf{\Sigma}_{12} = (-1)$
$\mathbf{\Sigma}_{21} = (-1)$
$\mathbf{\Sigma}_{22} = (5)$
$\mathbf{V}_1 = (.383)$
$\mathbf{V}_2 = (.277)$
S =

(a) Heterogenous Shopping Graph $\mathcal{G}_0$



(b) Homogeneous Scientific Paper Network $\mathcal{G}_1$

Fig. 5. How DE P-Rank computes similarity

that these phases take $O(m)$, $O\left(v^2 n + v n^2 + v^4 + v^6\right)$ and $O(v)$ time, respectively. Hence, the total time is bounded by $O\left(v n^2 + v^6\right)$ with $v \leq r$.

(3) Memory Space. (a) For pre-processing, it takes $O(n)$ space to compute $\mathbf{Q}$ and $\mathbf{P}$ (line 3). (b) For similarity computation, the memory consumption is dominated by $O(v \cdot \max\{v^3, n\})$, which includes $O(vn)$ space to decompose $\mathbf{Q}$ and $\mathbf{P}$ into low-rank matrices (line 4), and $O(v^4)$ for computing $\mathbf{\Sigma}^{-1}$ (line 7). (c) The result collection requires $O(v^2)$ space (line 10). Therefore, the total space can be bounded by $O(v \cdot \max\{v^3, n\})$ with $v \leq r$. (see Table 1 for a detailed analysis)

(4) Error Bound. The error ϵ_v is reasonably small in practice when $v \in [\frac{1}{2}r, r]$. Our experimental results in Section 6 show that for such v, the singular values σ_{v+1} and $\bar{\sigma}_{v+1}$ (in line 8) are almost zero, leading to a practically acceptable NDCG$_{30}$ (see Figure 13). As an extreme case of $v = r$, $\sigma_{v+1} = \bar{\sigma}_{v+1} = 0$, which implies that $\epsilon_v = 0$.

Table 1. Running Time & Memory Space for DE P-Rank in lines 2-9

#-line	time	memory	operation
2	$O(m)$	$O(n)$	row normalization of matrices
4	$O(vn^2 + v^2 n)$	$O(vn)$	low-rank v reduced SVD
5	$O(v^2 n + r)$	$O(v)$	matrix multiplications and inversions
6	$O(v^4 + v^2)$	$O(v^2)$	Kronecker products
7	$O(v^6 + v^4 + v^2 n)$	$O(v^4 + v^2)$	block matrix inversion
8	$O(1)$	$O(1)$	constant operations
9	$O(v)$	$O(v^2)$	reshape matrices

For instance, consider the WIKI (0715) data $(r = 15\text{K}, \sigma_1 = 1.12, \bar{\sigma}_1 = 1.08)$. Setting $C_{\text{in}} = 0.8, C_{\text{out}} = 0.6$, and $\lambda = 0.5$ will yield

$$\epsilon_v \leq \frac{0.5\times0.8\times1.12+0.5\times0.6\times1.08}{1-0.5\times0.8-0.5\times0.6} \times 10^{-7} \times 15\text{K} = 0.0039.$$

5 Probabilistic P-Rank Similarity Estimation

Although way better than cubic, the complexity bound of DE P-Rank is still too high to compute similarity in an on-line fashion. For ad-hoc (dynamic) queries on large graphs, the execution time is one of the most crucial metrics; it is worthwhile to drastically accelerate the P-Rank computation with a little sacrifice in accuracy.

This motivates us to study *the probabilistic P-Rank computation problem*. That is, given a graph $\mathcal{G}$, a query (u, v), and a desired probabilistic accuracy, it is to estimate the P-Rank similarity $s(u, v)$ in a scalable manner (*i.e.,* in worst-case linear time).

5.1 A Probabilistic P-Rank Model

In the light of the power series form of P-Rank in Lemma 1, our key observation is that P-Rank similarity can be viewed as a geometric sum of random walks, and its score $s\,(u, v)$ qualifies how soon two surfers are expected to meet at the same vertex if they start from vertices u and v and do random walks on a graph backwards and forwards.

The main idea is to utilize the first hitting time $\tau\,(u, v)$ of coalescing walks to estimate $s_l\,(u, v)$ of length l. The underlying rationale is that $\tau\,(u, v)$ can be represented in a compact way of storing only one integer (rather than a walk of length l) for each vertex-pair. It is far less costly to estimate $\tau\,(u, v)$ for $s(u, v)$ than to compute the entire similarity matrix $\mathbf{S}$. Specifically, we show the following result.

Theorem 2 (Probabilistic Model). *The P-Rank similarity score between vertices u and v, with damping factors C_{in} and C_{out} for in- and out-links, is equal to the weighted mean of their expected meeting distances with uniform independent walks, i.e.,*

$$s\,(u, v) = \mathbb{E}(\lambda \cdot C_{in}{}^{\tau_1(u,v)} + (1-\lambda) \cdot C_{out}{}^{\tau_2(u,v)}), \tag{6}$$

where $\mathbb{E}(\cdot)$ denotes the expectation of the random variables; and $\tau_i\,(u, v)\quad(i = 1, 2)$ are the first hitting time of the random surfers starting from the vertices u and v, and *following the links backwards* $(i = 1)$ *and forwards* $(i = 2)$*, respectively;* $\tau_i(u, v) = \infty$ *if they never hit; and* $\tau_i(u, v) = 0$ *if* $u = v$.

(A detailed proof of Theorem 2 will be provided after some discussions.)

Intuitively, Theorem 2 provides a stochastic model of P-Rank computation for interpreting the similarity score as the random walks of surfer pairs. From this perspective, the quality of similarity score hinges on whether the random surfers that start from two distinct vertices are close to a common "source" and meet within merely a few steps.

We first use *vertex-pair graph* $\mathcal{G}^2$ to formulate the hitting time of two surfers in $\mathcal{G}$. In $\mathcal{G}^2$, each vertex (u, v) represents a pair of vertices in $\mathcal{G}$, and each edge from (u, v) to (x, y) says that in $\mathcal{G}$, one surfer can move from u to x, and the other from v to y. Hence, in light of the power series form of P-Rank in Lemma 1, two surfers, in $\mathcal{G}$, starting from vertices u and v, following the links backwards (*resp.* forwards) and meeting within a few steps indicate that, in $\mathcal{G}^2$, there exists a path t from one singleton vertex (x, x) to (u, v) (*resp.* from (u, v) to (x, x)).

We then introduce the following notions to model the random surfers on $\mathcal{G}^2$.

(a) The *transformation* T in $\mathcal{G}^2$ is a mapping $T : t' \rightarrow t$ from one path t' into another t by adding (i) an edge $\langle (u, v), \mathcal{O}_i((u, v))\rangle$ to the beginning of t', or (ii) an edge $\langle \mathcal{I}_i((u, v)), (u, v)\rangle$ to the end of t'.

(b) The *length* of a path t, denoted by $l(t)$ is the number of edges in t.

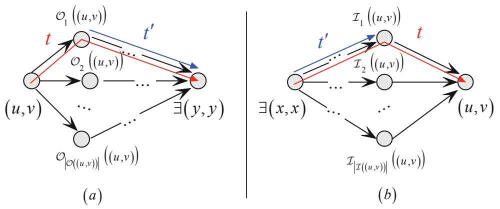


Fig. 6. Transformation from path t' into t

For one length of a random walk on $\mathcal{G}^2$, Figure 6 depicts the corresponding transformation from path t' to $t = T(t')$. Clearly,

$$l(t) = l(t') + 1. \tag{7}$$

(c) The *probability* of choosing a path $T(t')$ based on a path t', denoted by $p(T(t'))$, is defined to be

$$p(T(t')) = \begin{cases} \frac{1}{|\mathcal{I}((u,v))|} \cdot p(t'),\ t' : \exists(x,x) \rightarrow (u,v); \\ \frac{1}{|\mathcal{O}((u,v))|} \cdot p(t'),\ t' : (u,v) \rightarrow \exists(y,y). \end{cases} \tag{8}$$

We next complete the proof of Theorem 2 by showing that the probabilistic P-Rank model Eq.(6) is equivalent to the original model Eq.(1).

As the expectations in Eq.(6) can be rewirten as the sum *w.r.t.* probability distribution functions, it follows that

$$s(u, v) = \lambda \cdot \sum_{t:\exists(x,x)\rightarrow(u,v)} p(t) \cdot {C_{\text{in}}}^{l(t)} + (1 - \lambda) \cdot \sum_{t:(u,v)\rightarrow\exists(y,y)} p(t) \cdot {C_{\text{out}}}^{l(t)}.$$

Without loss of generality, we assume that $u \neq v$, and $\mathcal{I}(u)$, $\mathcal{I}(v)$, $\mathcal{O}(u)$, $\mathcal{O}(v) \neq \varnothing$. In the above equation, we split the sums *w.r.t.* one step of the path t. Combing this with Eqs.(7) and (8), we have

$$s(u,v) = \lambda \cdot \sum_{i=1}^{|\mathcal{I}((u,v))|} \sum_{\substack{t': \exists(x,x) \\ \to \mathcal{I}_i((u,v))}} \overbrace{p(T(t'))}^{= \frac{1}{|\mathcal{I}((u,v))|} \cdot p(t')} \cdot C_{\text{in}}{}^{\overbrace{l(T(t'))}^{=l(t')+1}} + (1-\lambda) \cdot \sum_{j=1}^{|\mathcal{O}((u,v))|} \sum_{\substack{t': \mathcal{O}_j((u,v)) \\ \to \exists(y,y)}} \overbrace{p(T(t'))}^{= \frac{1}{|\mathcal{O}((u,v))|} \cdot p(t')} \cdot C_{\text{out}}{}^{\overbrace{l(T(t'))}^{=l(t')+1}}$$

$$= \frac{\lambda \cdot C_{\text{in}}}{|\mathcal{I}(u)|\,|\mathcal{I}(v)|} \cdot \sum_{i=1}^{|\mathcal{I}((u,v))|} \sum_{t': \exists(x,x) \to \mathcal{I}_i((u,v))} p(t') \cdot C_{\text{in}}{}^{l(t')} + \frac{(1-\lambda) \cdot C_{\text{out}}}{|\mathcal{O}(u)|\,|\mathcal{O}(v)|} \cdot \sum_{j=1}^{|\mathcal{O}((u,v))|} \sum_{t': \mathcal{O}_j((u,v)) \to \exists(y,y)} p(t') \cdot C_{\text{out}}{}^{l(t')}$$

$$= \frac{\lambda \cdot C_{\text{in}}}{|\mathcal{I}(u)|\,|\mathcal{I}(v)|} \cdot \sum_{i=1}^{|\mathcal{I}(u)|} \sum_{j=1}^{|\mathcal{I}(v)|} s(\mathcal{I}_i(u), \mathcal{I}_j(v)) + \frac{(1-\lambda) \cdot C_{\text{out}}}{|\mathcal{O}(u)|\,|\mathcal{O}(v)|} \cdot \sum_{i=1}^{|\mathcal{O}(u)|} \sum_{j=1}^{|\mathcal{O}(v)|} s(\mathcal{O}_i(u), \mathcal{O}_j(v)).$$

Hence, the probabilistic model Eq.(6) agrees with the original model Eq.(1).

5.2 A Scalable Algorithm for P-Rank Estimation

In light of Theorem 2, we next devise a probabilistic algorithm for P-Rank estimation. The main result in this subsection is the following.

Theorem 3. *For any graph $\mathcal{G}$, the probabilistic P-Rank similarity can be solvable in $O(N \cdot n)$ time and $O(n+N)$ space, where N is the sample size.*

(The proof of Theorem 3 will be provided after a few discussions.)

As will be seen shortly, N is much smaller than n and affects the accuracy of estimation. This suggests that P-Rank can be solved in *linear* time with controlled probabilistic error, as opposed to the quadratic-time of its deterministic computation.

To prove Theorem 3, we first present the general idea of the P-Rank estimation. We then devise a randomized algorithm, followed by a complexity analysis.

The central idea is to use a sampling approach to estimate s from the first hitting time τ_1 and τ_2. (i) In the pre-computation phase, we utilize a tree index structure (instead of a low-rank factorization) to represent all the first hitting time for a set of coalescing walks in a compact way. (ii) In the query phase, we use two random surfers to estimate the P-Rank similarity by following the path that is a function of the first hitting time τ_1 and τ_2, which can be justified by Theorem 2.

Algorithm. The algorithm, referred to as PR P-Rank, is shown in Algorithm 2. It takes as input a graph $\mathcal{G}$, a query vertex-pair (u,v), a sample size N, a weighting factor λ, and two damping factors C_{in} and C_{out}, it returns the approximate similarity $\hat{s}_N(u,v)$.

The algorithm maintains the following data structures to ensure estimation quality. (a) A *(reversed) fingerprint tree* `FP` (*resp.* `RFP`) for vertices in $\mathcal{G}$. For each vertex u in the i-th samples $\texttt{FP}_i$ and $\texttt{RFP}_i$, $\texttt{RFP}_i(u,l)$ collects candidate vertices v in $\mathcal{G}$ such that each of the vertices u and v has an incoming directed path of length l that starts from some common vertex x; and $\texttt{FP}_i(u,l')$ is the set of vertices w in $\mathcal{G}$ such that each of the vertices u and w has an outcoming directed path of length l' that ends to some common vertex y. (b) A *random sample list* $\hat{s}_N^{(i)}$ of size N in $\mathcal{G}$ for estimating P-Rank similarity with $\hat{s}_N$ being the mean of N independent and identically distributed (*i.i.d.*) samples $\hat{s}_N^{(i)}$. (c) The *path length*, denoted by $\texttt{Len}(v/\cdots/u)$, from vertex v to u in $\mathcal{G}$. The path is *nonempty* if $\texttt{Len}(v/\cdots/u) \geq 1$.

The algorithm works as follows. (1) It first constructs two sample lists `RFP` and `FP` for $\mathcal{G}$ by invoking `Index Pre-processing` procedure (lines 1-6). For each vertex

```
Algorithm 2: PR P-Rank ($\mathcal{G}, (u,v), N, \lambda, C_{\text{in}}, C_{\text{out}}$)
Input  : web graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, query vertex pair $(u,v) \in \mathcal{V} \times \mathcal{V}$, sample size $N$,
         weighting factor $\lambda$, damping factors $C_{\text{in}}$ and $C_{\text{out}}$.
Output: P-Rank similarity score $\hat{s}_N(u,v)$.
INDEX PRE-PROCESSING
1  for $i \leftarrow 1, \cdots, N$ do
2      foreach vertex $u \in \mathcal{V}$ do
3          if $\exists v \in \mathcal{V} - \{u\}$ s.t. $u$ and $v$ meet at a common vertex $x$ along a chain of $l$
           in-links, and $\texttt{Len}\,(x/\cdots/v) = \texttt{Len}\,(x/\cdots/u) = l$ then
4              add $v$ to the reversed fingerprint tree $\texttt{RFP}_i\,(u,l)$ of $\mathcal{G}$.
5          else if $\exists w \in \mathcal{V} - \{u\}$ s.t. $u$ and $w$ meet at a common vertex $y$ along a chain of $l'$
           out-links, and $\texttt{Len}\,(v/\cdots/y) = \texttt{Len}\,(w/\cdots/y) = l'$ then
6              add $w$ to the fingerprint tree $\texttt{FP}_i\,(u,l')$ of $\mathcal{G}$.
QUERY $\hat{s}_N\,(u,v)$
7  for $i \leftarrow 1, \cdots, N$ do
8      if there exists a positive integer $l$ s.t. $\texttt{RFP}_i\,(u,l) = \texttt{RFP}_i\,(v,l)$ then
9          $l_0 \leftarrow \min_l \left\{ l \in \mathbf{Z}^+ \,|\texttt{RFP}_i\,(u,l) = \texttt{RFP}_i\,(v,l) \right\}$.
10     else if there exists a positive integer $l'$ s.t. $\texttt{FP}_i\,(u,l') = \texttt{FP}_i\,(v,l')$ then
11         $l'_0 \leftarrow \min_{l'} \left\{ l' \in \mathbf{Z}^+ \,|\texttt{FP}_i\,(u,l') = \texttt{FP}_i\,(v,l') \right\}$.
12     $\hat{s}_N^{(i)} \leftarrow \lambda \cdot {C_{\text{in}}}^{l_0} + (1-\lambda) \cdot {C_{\text{out}}}^{l'_0}$.
13 $\hat{s}_N \leftarrow \frac{1}{N} \cdot \sum_{i=1}^{N} \hat{s}_N^{(i)}$.
14 return $\hat{s}_N$.
```

u in $\mathcal{G}$, the i-th sample $\texttt{RFP}_i\,(u,l)$ (*resp.* $\texttt{FP}_i\,(u,l')$) collects the vertex v such that u and v have some common vertex x along a chain of l in-links (*resp.* l'out-links) in $\mathcal{G}$ (lines 4 and 6). (2) It then computes all the samples $\hat{s}_N^{(i)}$ by inspecting the vertices and path lengths collected in RFP and FP (lines 7-12). More concretely, for each sample $\hat{s}_N^{(i)}$, PR P-Rank identifies the minimum length l_0 (*resp.* ${l_0}'$) of the incoming (*resp.* outgoing) directed path, along which u and v may reach a common vertex (lines 9 and 11). Furthermore, utilizing l_0 and ${l_0}'$, it then computes $\hat{s}_N^{(i)}$ (line 12), which can be justified by Theorem 2. These $\hat{s}_N^{(i)}$ $(i = 1, \cdots, N)$ constitute a sequence of *i.i.d.* random samples. They are averaged to produce the final score $\hat{s}_N$ (lines 13-14).

To complete the proof of Theorem 3, we next show that (1) PR P-Rank has linear time complexity bound; (2) the memory requirement is bounded by $O(N+n)$; (3) the sample size $N \ll n$ in practice; (4) the error bounds are reasonably small; (5) the relative order of PR P-Rank scores is almost preserved.

(1) Running Time. PR P-Rank consists of three phases: (a) For pre-processing (lines 1-6), PR P-Rank invokes the randomized algorithm [5] for FPT indexing (lines 4 and 6), which is in $O\,(N \cdot n)$ time. (b) For on-line query (lines 7-12), PR P-Rank computes l_0 and ${l_0}'$ for each sample $\hat{s}_N^{(i)}$ in $O\,(n)$ time, being $O\,(N \cdot n)$ time for N samples. (c) For computing $\hat{s}_N$ (lines 13-14), it takes $O\,(N)$ time to collect all the samples.

Therefore, the total time of PR P-Rank is in $O\,(N \cdot n)$ time.

(2) Memory Space. The memory requirement is totally bounded by $O(N+n)$, comprising three phases: (a) In the precomputation phase, FPT indexing (lines 4 and 6) needs $O(n)$ space to maintain $\texttt{RFP}_i$ and $\texttt{FP}_i$ for every sample $\hat{s}_N^{(i)}$. (b) During the online query phase, it takes $O(n)$ space for finding the shortest meeting distance l_0 and l'_0 (lines 9 and 11), and $O(N)$ space for collecting all the similarity samples $\hat{s}_N^{(i)}$ (line 13). (c) Computing $\hat{s}_N$ on-the-fly requires $O(N)$ space.

(3) Sample Size N. (a) Choosing $N \geq -2\left\lceil(\sigma/\epsilon)^2 \log \alpha\right\rceil$ suffices to ensure that $\Pr\left(|\hat{s}_N - s| \geq \epsilon\right) < \alpha$ (where $\hat{s}_N(u,v)$ is the sample mean, and σ^2 the variance) , given any accuracy ϵ and confidence level $1-\alpha$ $(\alpha \in (0,1))$. This is because applying the *Bernstein's Theorem* [11] yields $\exp(-\frac{1}{2}(\epsilon\sqrt{N}/\sigma)^2) < \alpha$. (b) N is typically much smaller than n in practice, which can be verified by our empirical results in Section 6 (see Figure 14. For instance, consider DBLP (98-07) graph with $n = 10K$ vertices. Given $\epsilon = 0.15\sigma$ and $\alpha = 0.05$, we have $N \geq -2\left\lceil 0.15^{-2}\log(0.05)\right\rceil = 267$.

(4) Error Bound. We denoted by $Err \triangleq \sup_{N\geq 1}\Pr\left(|\hat{s}_N - s| \geq \epsilon\right)$.

(a) An upper bound can be obtained from *Bernstein's Theorem* [11], which gives

$$Err \leq \exp(-N\epsilon^2/(2\sigma^2)),$$

(b) A lower bound follows from the *Central Limit Theorem* [10], in which

$$Err \geq \Pr\left(|\hat{s}_N - s| \geq \epsilon\right) = \Pr\left(\left|\frac{1}{\sqrt{N}}\sum_{i=1}^{N}\left(\frac{\hat{s}_N^{(i)} - s}{\sigma}\right)\right| \geq \frac{\epsilon\sqrt{N}}{\sigma}\right) = 2 - 2\Phi\left(\frac{\epsilon\sqrt{N}}{\sigma}\right),$$

where $\Phi(\cdot)$ is the cumulative distribution function of normal distribution $\mathcal{N}(0,1)$.

(c) Both bounds of Err are reasonable because (i) $\exp(-N\epsilon^2/(2\sigma^2))$ is decreasing *w.r.t.* N, and (ii) $\Phi(\epsilon\sqrt{N}/\sigma)$ non-decreasingly approaches 1 as N increases. Hence,

$$\lim_{N\to\infty}\exp(-N\epsilon^2/(2\sigma^2)) = \lim_{N\to\infty} 2 - 2\Phi(\epsilon\sqrt{N}/\sigma) = 0.$$

According to the *Squeeze Principle* [10], we have $\hat{s}_N(u,v) \overset{a.s.}{\to} s(u,v)$ as $N \to \infty$.

(d) Err is typically small and acceptable in practice. For instance, Setting $\epsilon = 0.15\sigma$ and $N = 300$, we have $\exp\big(-\frac{N\epsilon^2}{2\sigma^2}\big) = \exp\big(-\frac{300\times(0.15\sigma)^2}{2\sigma^2}\big) \approx 0.034$ and $2 - 2\Phi\big(\frac{\epsilon\sqrt{N}}{\sigma}\big) = 2 - 2\Phi\big(0.15\times\sqrt{300}\big) \approx 0.0094$. This implies that only 0.94% (at most 3.4%) of the estimated scores $\hat{s}_N$ fall outside the interval $[s - 0.15\sigma, s + 0.15\sigma]$.

(5) Relative Order. The relative order of the similarity estimated by PR P-Rank is almost preserved with the deterministic result, as shown in the following theorem.

Theorem 4. *Let* $\hat{s}_N(\cdot,\cdot)$ *be the estimated similarity by PR P-Rank with* N *being the sample size, and* $s(\cdot,\cdot)$ *the exact similarity. If* $s(u,v) > s(u,w) + \epsilon$, *then*

$$\Pr(\hat{s}_N(u,v) - \hat{s}_N(u,w) > \epsilon) \leq \exp(-N\epsilon^2/2) \quad (\forall u,v,w \in \mathcal{V}).$$

(A detailed proof of Theorem 4 is provided in the Appendix.)

Our empirical results on DBLP will further verify that for $N \geq 350$, PR P-Rank can almost maintain the relative order of similarity (see Figure 12).

6 Experimental Evaluation

We conduct a comprehensive empirical study over several real and synthetic datasets to evaluate (1) the scalability, time and space efficiency of the proposed algorithms, (2) the approximability of DE P-Rank, and (3) the effectiveness of PR P-Rank.

6.1 Experimental Settings

Datasets. We used three real datasets (AMZN, DBLP, and WIKI) to evaluate the efficacy of our methods, and synthetic data (0.5M-3.5M RAND) to vary graph characteristics. The sizes of AMZN, WIKI and DBLP are shown in Tables 2-4.

Table 2. AMZN

	0505	0601
$\lvert\mathcal{V}\rvert$	410K	402K
$\lvert\mathcal{E}\rvert$	3,356K	3,387K

Table 3. DBLP

	98-99	98-01	98-03	98-05	98-07
$\lvert\mathcal{V}\rvert$	1,525	3,208	5,307	7,984	10,682
$\lvert\mathcal{E}\rvert$	5,929	13,441	24,762	39,399	54,844

Table 4. WIKI

	0715	0827	0919
$\lvert\mathcal{V}\rvert$	3,088K	3,102K	3,116K
$\lvert\mathcal{E}\rvert$	1,126K	1,134K	1,142K

(1) AMZN *data*[4] are based on *Customers Who Bought This Item Also Bought* feature of the Amazon website. Each node represents a product. There is a directed edge from node i to j if a product i is frequently co-purchased with product j. Two datasets were collected in May 5 2003, and June 1 2003.

(2) DBLP *data*[5] record co-authorships among scientists in the Bibliography. We extracted the 10-year (from 1998 to 2007) author-paper information, and singled out the publications on 6 conferences (ICDE, VLDB, SIGMOD, WWW, SIGIR, and KDD). Choosing every two years as a a time step, we built 5 co-authorship graphs.

(3) WIKI *data*[6] contain millions of encyclopedic articles on a vast array of topics to the latest scientific research. We built 3 graphs from the English WIKI dumps,where each vertex represents an article, and edges the relationship that "a category contains an article to be a link from the category to the article".

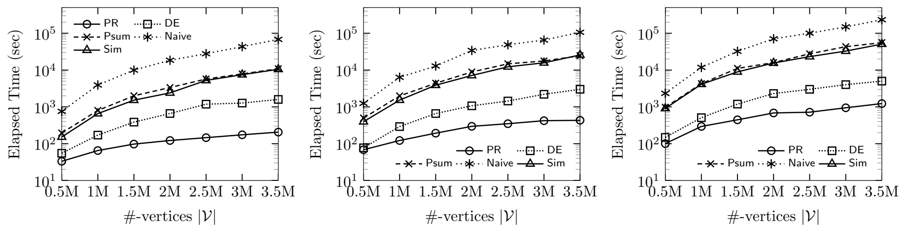
(4) RAND *data* were produced by C++ boost generator for digraphs, with 2 parameters: the number of vertices, and the number of edges.

Parameter Settings. To keep consistency with the experimental conditions in [26], we assigned each of the following parameters a default value.

Notation	Description	Default	Notation	Description	Default
C_{in}	in-link damping factor	0.8	λ	weighting factor	0.5
C_{out}	out-link damping factor	0.6	v	low rank	$50\% \times \texttt{Rank}$
ϵ	desired accuracy	0.001	N	sample size	350

Compared Algorithms. We have implemented the following algorithms. (1) DE and PR, *i.e.,* DE P-Rank and PR P-Rank; (2) Naive, a K-Medoids P-Rank iterative algorithm ($K = 10$) based on a radius-based pruning method [26]; (3) Psum, a variant

[4] http://snap.stanford.edu/data/index.html
[5] http://www.informatik.uni-trier.de/~ley/db/
[6] http://en.wikipedia.org/



(a) $|\mathcal{E}| = 2M$ (b) $|\mathcal{E}| = 4M$ (c) $|\mathcal{E}| = 6M$

Fig. 7. Scalability on Synthetic Datasets

of P-Rank, leveraging a partial sum function [18] to compute similarity; (4) Sim, an enhanced version of SimRank algorithm [1], which takes account of the evidence factor for incident vertices. These algorithms were implemented in C++, except that the MATLAB implementation [3] for calculating `RSVD` () and `Rank` ().

All experiments were run on a machine with a Pentium(R) Dual-Core (2.00GHz) CPU and 4GB RAM, using Windows Vista. Each experiment was repeated over 5 times, and the average is reported here.

6.2 Experimental Results

Scalability. We first evaluate the scalability of the five ranking algorithms, using synthetic data. In these experiments, PR fingerprint tree indexing is precomputed and shared by all vertex pairs in a given graph, and thus its cost is counted only once.

We randomly generate 7 graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with the edge size $|\mathcal{E}|$ varying from 2M to 6M. The results are reported in Figures 7(a), 7(b) and 7(c). We can notice that (1) DE is almost one order of magnitude faster than the other algorithms when $|\mathcal{V}|$ is increased from 0.5M to 3.5M. (2) Execution time for PR increases linearly with $|\mathcal{V}|$ due to the use of finger printed trees. Varying $|\mathcal{E}|$, we also see that the CPU time of DE is less sensitive to $|\mathcal{E}|$. This is because the time of DE mainly depends on the number of vertices having the similar neighbor structures. Hence, graph sparsity has not a large impact on DE. In light of this, DE scales well with $|\mathcal{E}|$, as expected.

Time & Space Efficiency. We next compare the CPU time and memory space of the five ranking algorithms on real datasets. The results are depicted in Figures 8 and 9. It can be seen that the time and space of PR clearly outperform the other approaches on AMZN, WIKI, and DBLP, *i.e.,* the use of Monte Carlo sampling approach is effective. In all the cases, DE runs faster than Psum, Naive and Sim with moderate memory requirements. This is because DE uses a singular value decomposition to cluster a large body of vertices having the similarity neighbor structures, and a low-rank approximation to eliminate the vertices of tiny singular values, which can save large amounts of memory space, and avoid repetitive calculations of "less important" vertices. Besides, with the increasing number of vertices on DBLP data, the upward trends of the time and space for DE and PR match our analysis in Sections 4 and 5.

Figure 10 depicts how the total computational time and memory space are amortized on the different phases of DE and PR, respectively, over AMZN data. We see from the results that the similarity calculation phase of DE is far more time and space consuming

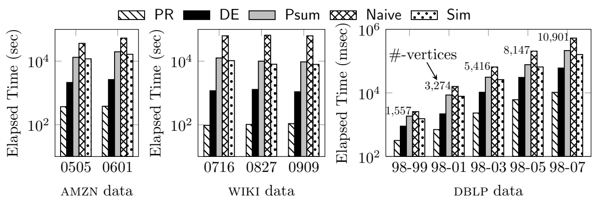


Fig. 8. Time Efficiency on Real Datasets

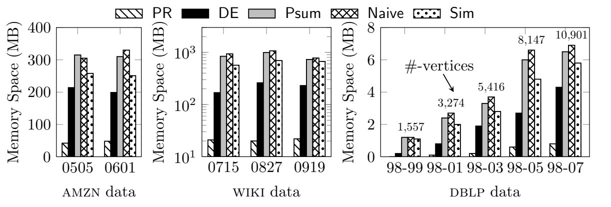


Fig. 9. Memory Space on Real Datasets

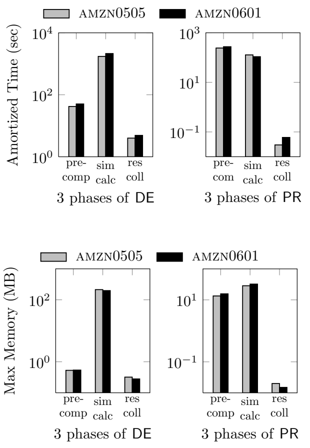


Fig. 10. Amortized Costs

(97.4% time and 99.59% space) than the other two phases (2.3% time and 0.27% space for preprocessing, and 0.3% time and 0.14% space for result collection), which is expected because factorizing $\mathbf{Q}$ and $\mathbf{P}$, and computing $\mathbf{\Sigma}^{-1}$ yield a considerable amount of complexity. We also notice that the total cost of PR is well balanced between off-line pre-indexing and on-line query phases, both of which take high proportions of CPU time and memory usage, *i.e.,* almost 71.6% total time and 32.6% space are leveraged on indexing phase, and 28.3% time and 67.3% space on query phase. This tells that the use of finger printed trees can effectively reduce the overhead costs of PR.

Accuracy. We now evaluate the accuracy of the five algorithms on real data. The *Normalized Discounted Cumulative Gain* (NDCG) measure [8] is adopted. The NDCG at a rank position p is defined as $\text{NDCG}_p = \frac{1}{\text{IDCG}_p} \sum_{i=1}^{p} (2^{\text{rank}_i} - 1)/(\log_2(1+i))$, where rank_i is the average similarity at rank position i judged by the human experts, and IDCG_p is the normalization factor to guarantee that NDCG of a perfect ranking at position p equals 1.

Figure 11 compares the accuracy of DE and PR with that of Naive, Psum and Sim returned by NDCG_{30} on AMZN, WIKI and DBLP, respectively. It can be seen that DE always achieves higher accuracy than PR. The accuracy of PR is not that good because some valid finger printed trees may be neglected with certain probability by PR sampling. The results on DBLP also show that the accuracy of DE and PR is insensitive to $|\mathcal{V}|$. Hence, adding vertices does not affect the error in estimation, as expected.

We further evaluate the ground truth calculated by DE and PR on DBLP (98-07) dataset to retrieve the top-k most similar authors for a given query u. Interestingly, Figure 12 depicts the top-10 ranked results for the query “Jennifer Widom” according to the similarity scores returned by PR, DE and Naive, respectively. These members were frequent co-authors of the 6 major conference papers with “Jennifer Widom” from 1998 to 2007. It can be noticed that the ranked results for different ranking algorithms on DBLP (98-07) are practically acceptable and obey our common sense pretty well.

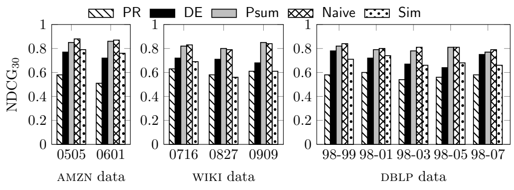


Fig. 11. Accuracy on Real Datasets

Rank	PR	DE	Naive
1	Shivnath Babu	Shivnath Babu	Shivnath Babu
2	Chris Olston	Yingwei Cui	Yingwei Cui
3	Jun Yang	Chris Olston	Chris Olston
4	Yingwei Cui	Jun Yang	Jun Yang
5	Rajeev Motwani	Arvind Arasu	Rajeev Motwani
6	Arvind Arasu	Rajeev Motwani	Arvind Arasu
7	David J. DeWitt	Alon Y. Halevy	Utkarsh Srivastava
8	Glen Jeh	Anish Das Sarma	David J. DeWitt
9	Utkarsh Srivastava	Omar Benjelloun	Omar Benjelloun
10	Omar Benjelloun	David J. DeWitt	Alon Y. Halevy

Fig. 12. Top-10 Co-authors of Jennifer Widom on DBLP

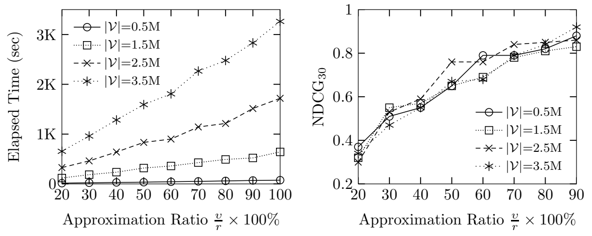


Fig. 13. Effects of v for DE

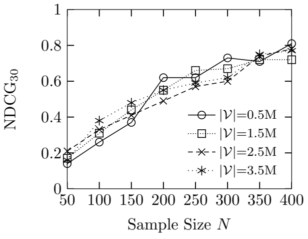


Fig. 14. Effects of N for PR

The similarities calculated by DE and PR almost preserve the relative order of Naive. Hence, both DE and PR can be effectively used for P-Rank similarity estimation in top-k nearest neighbor search on real networks.

Effects of v. For DE algorithm, we next investigate the impact of approximation rank v and adjacency matrix rank r on similarity estimation, using synthetic data.

We use 4 web graphs with the size $|\mathcal{V}|$ of the set of vertices ranging from 0.5M to 3.5M, $2|\mathcal{V}|$ edges. We consider various approximation ranks v for a given graph $\mathcal{G}$. We fix $|\mathcal{V}|$ while varying v from $10\% \times r$ to $90\% \times r$ with r being the rank of adjacency matrix for $\mathcal{G}$. The results are reported in Figure 13, which visualizes the low-rank v as a speed-accuracy trade-off. When v becomes increasingly close to r (*i.e.,* the radio $\frac{v}{r}$ approaches 1), high accuracy (NDCG$_{30}$) could be expected, but more running time needs to be consumed. This tells that adding approximation rank v will induce smaller errors for similarity estimation, but it will increase the complexity of computation up to a point of the rank r when no extra approximation errors can be reduced.

Effects of N. For PR algorithm, we evaluate the impact of the sample size N of the finger printed trees on similarity quality.

We consider 4 web graphs $\mathcal{G}$ with the size $|\mathcal{V}|$ $(= \frac{1}{2}|\mathcal{E}|)$ ranged from 0.5M to 3.5M. In Figure 14, we fixed $|\mathcal{V}|$ while varying N from 50 to 400. In all the cases, when the sample size is larger ($N > 300$), higher accuracy could be attained (NDCG$_{30} > 0.6$), irrelevant to the size $|\mathcal{V}|$ of graph. The result reveals that adding samples of finger printed trees reduces errors in estimation, and hence improves the effectiveness of PR.

Summary. We find the following. (1) DE and PR can scale well with the large size of graphs, whereas Naive, Psum, and Sim fail to run with an acceptable time. (2) DE significantly outperforms Psum and Sim by almost one order of magnitude with error guarantees (a drop 10% in NDCG). (3) PR may run an order of magnitude faster than DE with a little sacrifice in accuracy (5% relative error), which is practically acceptable for ad-hoc query performed in an on-line fashion.

7 Related Work

P-Rank has become an appealing measure of similarity used in a variety of areas, such as publication network [26], top-k nearest neighbor search [13], and social graph [2,17]. The traditional method leverages a fixed-point iteration to compute P-Rank, yielding $O(Kn^4)$ time in the worst case. Due to the high time complexity, Zhao *et al.* [26] further propose the radius- and category-based pruning techniques to improve the computation of P-Rank to $O(Kd^2n^2)$ at the expense of reduced accuracy, where d is the average degree in a graph. However, their heuristic methods can not guarantee the similarity accuracy. In contrast, our methods are based on two matrix forms for optimizing P-Rank computation with fast speed and provable accuracy.

There has also been work on other similarity optimization (*e.g.,* [5,6,15,18,20,22,24, 25]). Lizorkin *et al.* [18] proposed an interesting memoization approach to improve the computation of SimRank from $O(Kn^4)$ to $O(Kn^3)$. The idea of memoization can be applied to P-Rank computation in the same way. A notion of the weighted and evidence-based SimRank is proposed by Antonellis *et al.* [1], yielding better query rewrites for sponsored search. He *et al.* [6] and Yu *et al.* [24] show interesting approaches to paralleling the computation of SimRank.

Closer to this work are [15, 17, 27]. Our prior work [17] focuses on P-Rank computations on undirected graphs by showing an $O(n^3)$-time deterministic algorithm; however the optimization techniques in [17] rely mainly on the symmetry of the adjacency matrix. In comparison, this work further studies the general approaches to optimizing P-Rank on directed graphs, achieving quadratic-time for deterministic computation, and linear-time for probabilistic estimation. Extensions of SimRank are studied in [27] for structure- and attribute-based graph clustering, but the time complexity is still cubic in the number of vertices. Recent work by Li *et al.* shows an incremental algorithm for dynamically computing SimRank; however it is not clear that extending to the P-Rank model is possible. Besides, it seems hard to obtain an error bound for computing SimRank on directed graphs as the error bound in [15] is only limited to undirected graphs. In contrast, the error bounds in our work may well suit digraphs.

In comparison to the work on deterministic SimRank computation, the work on probabilistic computation is limited. Li *et al.* [16] exploit the block structure of linkage patterns for SimRank estimation, which is in $O(n^{4/3})$ time. Fogaras *et al.* [4, 5] utilize a random permutation method in conjunction with Monte Carlo Simulation to estimate SimRank in linear time. As opposed to our probabilistic methods, (a) these algorithms are merely based on ingoing links; it seems hard to observe global structural connectivity while maintaining linear time, by using only a finger printed tree structure. (b) The theoretical guarantee of choosing a moderate sample size is not mentioned in [4, 5] as these studies ignore the central limit property of the finger printed tree by and large.

8 Conclusion

In this paper, we have studied the optimization problem of P-Rank computation. We proposed two equivalent matrix forms to characterize the P-Rank similarity. (i) Based on the matrix inversion form of P-Rank, a deterministic algorithm was devised to reduce the computational time of P-Rank from cubic to quadratic in the number of vertices; the error estimate was given as a by-product when the low rank approximation was deployed. (ii) Based on the matrix power series form of P-Rank, a probabilistic algorithm was also suggested for further speeding up the computation of P-Rank in linear time with controlled accuracy. The experimental results on both real and synthetic datasets have demonstrated the efficiency and effectiveness of our methods.

Acknowledgements. We greatly appreciate the constructive comments from the anonymous reviewers. We also thank National ICT Australia Ltd (NICTA) for their sponsorship.

References

1. Antonellis, I., Garcia-Molina, H., Chang, C.-C.: SimRank++: query rewriting through link analysis of the click graph. PVLDB 1(1) (2008)
2. Cai, Y., Zhang, M., Ding, C.H.Q., Chakravarthy, S.: Closed form solution of similarity algorithms. In: SIGIR, pp. 709–710 (2010)
3. Cowell, W.R. (ed.): Sources and Development of Mathematical Software. Prentice-Hall Series in Computational Mathematics, Cleve Moler, Advisor (1984)
4. Fogaras, D., Rácz, B.: A Scalable Randomized Method to Compute Link-Based Similarity Rank on the Web Graph. In: Lindner, W., Fischer, F., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) EDBT 2004. LNCS, vol. 3268, pp. 557–567. Springer, Heidelberg (2004)
5. Fogaras, D., Rácz, B.: Scaling link-based similarity search. In: WWW (2005)
6. He, G., Feng, H., Li, C., Chen, H.: Parallel SimRank computation on large graphs with iterative aggregation. In: KDD (2010)
7. Horn, R.A., Johnson, C.R.: Matrix Analysis. Cambridge University Press (February 1990)
8. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of IR techniques. ACM Trans. Inf. Syst. 20, 422–446 (2002)
9. Jeh, G., Widom, J.: SimRank: a measure of structural-context similarity. In: KDD, pp. 538–543 (2002)
10. Kallenberg, O.: Foundations of Modern Probability. Springer (January 2002)
11. Latuszynski, K., Miasojedow, B., Niemiro, W.: Nonasymptotic bounds on the estimation error for regenerative MCMC algorithms. Technical report (2009)
12. Laub, A.J.: Matrix Analysis For Scientists And Engineers. Society for Industrial and Applied Mathematics, Philadelphia (2004)
13. Lee, P., Lakshmanan, L.V.S., Yu, J.X.: On top-k structural similarity search. In: ICDE (2012)
14. Leskovec, J., Huttenlocher, D.P., Kleinberg, J.M.: Signed networks in social media. In: CHI, pp. 1361–1370 (2010)
15. 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)
16. Li, P., Cai, Y., Liu, H., He, J., Du, X.: Exploiting the Block Structure of Link Graph for Efficient Similarity Computation. In: Theeramunkong, T., Kijsirikul, B., Cercone, N., Ho, T.-B. (eds.) PAKDD 2009. LNCS, vol. 5476, pp. 389–400. Springer, Heidelberg (2009)

17. Li, X., Yu, W., Yang, B., Le, J.: ASAP: Towards Accurate, Stable and Accelerative Penetrating-Rank Estimation on Large Graphs. In: Wang, H., Li, S., Oyama, S., Hu, X., Qian, T. (eds.) WAIM 2011. LNCS, vol. 6897, pp. 415–429. Springer, Heidelberg (2011)
18. Lizorkin, D., Velikhov, P., Grinev, M.N., Turdakov, D.: Accuracy estimate and optimization techniques for SimRank computation. VLDB J. 19(1) (2010)
19. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. Society for Industrial and Applied Mathematics (April 2003)
20. Sarma, A.D., Gollapudi, S., Panigrahy, R.: Estimating PageRank on graph streams. In: PODS, pp. 69–78 (2008)
21. Tsatsaronis, G., Varlamis, I., Nørvåg, K.: An Experimental Study on Unsupervised Graph-based Word Sense Disambiguation. In: Gelbukh, A. (ed.) CICLing 2010. LNCS, vol. 6008, pp. 184–198. Springer, Heidelberg (2010)
22. 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)
23. Yu, W., Le, J., Lin, X., Zhang, W.: On the Efficiency of Estimating Penetrating-Rank on Large Graphs. In: Ailamaki, A., Bowers, S. (eds.) SSDBM 2012. LNCS, vol. 7338, pp. 231–249. Springer, Heidelberg (2012), `http://www.cse.unsw.edu.au/~weirenyu/yu-tr-ssdbm2012.pdf`
24. Yu, W., Lin, X., Le, J.: Taming Computational Complexity: Efficient and Parallel SimRank Optimizations on Undirected Graphs. In: Chen, L., Tang, C., Yang, J., Gao, Y. (eds.) WAIM 2010. LNCS, vol. 6184, pp. 280–296. Springer, Heidelberg (2010)
25. Yu, W., Zhang, W., Lin, X., Zhang, Q., Le, J.: A space and time efficient algorithm for SimRank computation. World Wide Web 15(3), 327–353 (2012)
26. Zhao, P., Han, J., Sun, Y.: P-Rank: a comprehensive structural similarity measure over information networks. In: CIKM (2009)
27. Zhou, Y., Cheng, H., Yu, J.X.: Graph clustering based on structural/attribute similarities. PVLDB 2(1) (2009)

Appendix: Proof of Theorem 4

Proof. Let $A \triangleq \{|\hat{s}_N(u,v) - s(u,v)| \geq \epsilon\}$, and $B \triangleq \{|\hat{s}_N(u,w) - s(u,w)| \geq \epsilon\}$. We first find an upper bound of variance σ^2 for any sample $\hat{s}_N^{(i)} \in [0,1]$.

$$\sigma^2 = \mathbb{E}[(\hat{s}_N^{(i)})^2] - \mathbb{E}[\hat{s}_N^{(i)}]^2 \leq \mathbb{E}[\hat{s}_N^{(i)}] - \mathbb{E}[\hat{s}_N^{(i)}]^2 = -(\mathbb{E}[\hat{s}_N^{(i)}] - 1/2)^2 + 1/4 \leq 1/4.$$

Then, using the Bernstein's inequality, we have

$$\Pr(A \cap B) \leq \Pr(A) \leq \exp(-N\epsilon^2/(2\sigma^2)) \leq \exp(-2N\epsilon^2).$$

Since $s(u,v) > s(u,w) + \epsilon$, we have

$$\begin{aligned} A \cap B \supseteq \{&\hat{s}_N(u,v) - \hat{s}_N(u,w) > \hat{s}_N(u,v) - \hat{s}_N(u,w) - \overbrace{(s(u,v) - s(u,w))}^{>\epsilon(>0)} \\ &= \underbrace{(\hat{s}_N(u,v) - s(u,v))}_{\geq\epsilon} - \underbrace{(\hat{s}_N(u,w) - s(u,w))}_{<-\epsilon} > 2\epsilon\} \end{aligned}$$

Hence, $\Pr\{\hat{s}_N(u,v) - \hat{s}_N(u,w) > \epsilon\} \leq \exp(-N\epsilon^2/2)$.