Estimating Walk-Based Similarities Using Random Walk

Shogo Murai*
The University of Tokyo
Bunkyo-ku, Tokyo, Japan
s.murai@is.s.u-tokyo.ac.jp

Yuichi Yoshida†
National Institute of Informatics
Chiyoda-ku, Tokyo, Japan
yyoshida@nii.ac.jp

ABSTRACT

Measuring similarities between vertices is an important task in network analysis, which has numerous applications. One major approach to define a similarity between vertices is by accumulating weights of walks between them that encompasses personalized PageRank (PPR) and Katz similarity. Although many effective methods for PPR based on efficient simulation of random walks have been proposed, these techniques cannot be applied to other walk-based similarity notions because the random walk interpretation is only valid for PPR.

In this paper, we propose a *random-walk reduction* method that reduces the computation of any walk-based similarity to the computation of a stationary distribution of a random walk. With this reduction, we can exploit techniques for PPR to compute other walk-based similarities. As a concrete application, we design an indexing method for walk-based similarities with which we can quickly estimate the similarity value of queried pairs of vertices, and theoretically analyze its approximation error. Our experimental results demonstrate that the instantiation of our method for Katz similarity is two orders of magnitude faster than existing methods on large real networks, without any deterioration in solution quality.

CCS CONCEPTS

• **Information systems** → **Web mining**; • **Mathematics of computing** → *Graph algorithms.*

KEYWORDS

Walk-based similarity, Katz similarity, random walk

ACM Reference Format:
Shogo Murai and Yuichi Yoshida. 2019. Estimating Walk-Based Similarities Using Random Walk. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3308558.3313421

*Supported by JST ERATO Grant Number JPMJER1305
†Supported by JSPS KAKENHI Grant Number JP17H04676.

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313421

1 INTRODUCTION

In network analysis, measuring similarities between vertices is an important task that has numerous applications such as recommendation [12, 15] and link prediction [20, 25].

One major class of similarity measures is *walk-based similarities* [14]. Here, we first define the weight of a walk. Then, we define the similarity value between two vertices as the sum of the weights of all the walks connecting them. By changing the definition of the weight of a walk, we can express various similarity notions, including personalized PageRank (PPR) [22], Katz similarity [13], and adsorption [3]. Among those, PPR has drawn much attention and several efficient methods have been proposed [1, 2, 4, 10, 17, 18, 24]. In contrast, there are only a few efficient methods that can be applied to *general* walk-based similarities [14, 30], even though they include other practically important similarity notions.

In this paper, we propose a *random-walk reduction* method to address the issue above. Given any walk-based similarity notion, this method constructs a weighted graph on which we can perform a random walk. Then, the original similarity values can be easily recovered from the stationary distribution of a random walk on the weighted graph. With this reduction, we can apply any algorithm that was originally designed to compute PPR (such as the Monte Carlo method [10], FAST-PPR [18] and BiPPR [17]) to any other walk-based similarity notion.

As an application of our random-walk reduction, for any walk-based similarity, we transform BiPPR [17], originally designed to compute the PPR of a target vertex from a source vertex, to a method for the similarity. We also provide a theoretical analysis on its approximation error. The experiment shows that the transformed method instantiated by Katz similarity is two orders of magnitude faster than existing methods without deteriorating the solution quality.

This paper is structured as follows. Section 2 gives an overview of related works. Section 3 provides a basic explanation of our problem. Section 4 explains previous methods that we rely on and outlines our method. We present our method in detail in Sections 5 and 6. We theoretically guarantee the approximation error of our method in Section 7. In Section 8, we present experimental results on one-to-one Katz similarity estimation and show that our method is two orders of magnitude faster than existing methods on large real networks.

2 RELATED WORK

Considering the importance of vertex similarities, several walk-based similarity metrics have been proposed, including PPR [22], Katz similarity [13], and adsorption [3].

Although they are all computable by solving certain linear equations, this approach is time-consuming. To address this issue, various other methods have been proposed to approximately compute the values of these metrics and to compute *top-k* vertices from a given vertex with regard to the given metric.

PPR has been intensively studied in the literature. Monte Carlo approximation algorithms were proposed in [10]. Andersen *et al.* [1, 2] proposed algorithms that iteratively update the estimated value by checking the local neighborhood of each vertex. Lofgren *et al.* [18] proposed bidirectional algorithms for PPR. Their algorithms combine Monte Carlo simulation (*forward work*) and backward local push algorithm (*backward work*) to obtain better estimations in shorter time. For any *threshold* $\delta > 0$, their algorithm for one-to-one similarity estimation can estimate PPR values larger than δ with a theoretical guarantee in average $O(\sqrt{\bar{d}/\delta})$ time, where $\bar{d}$ is the average degree of the graph. These algorithms were later simplified and improved in [17]. Their works are also extended to estimation of probabilities in Markov models [4] and single element of linear systems of form $A^\ell \boldsymbol{z}$ [24].

Esfandiar *et al.* [9] proposed algorithms for approximating Katz similarities and for computing the top-k vertices from the given vertex with theoretical guarantees.

However, there are few methods that can handle *general* walk-based similarity metrics. The only works we are aware of are [14] and [30]. Both methods considered computing top-k vertices by locally updating estimated values and maintaining their upper and lower bounds, which cannot efficiently handle one-to-one similarity estimation.

3 PRELIMINARIES

For a positive integer $n \in \mathbb{N}$, let $[n]$ denote the set $\{1, 2, \ldots, n\}$. We denote by $\langle \cdot, \cdot \rangle$ the standard inner product.

Let $G = (V, E)$ be a directed graph with n vertices and m edges. For a vertex $v \in V$, let $N^+(v)$ denote the set of out-neighbors of $v \in V$ and $d^+(v) := |N^+(v)|$ denote the out-degree of v. Similarly, we define $N^-(v)$ and $d^-(v)$ for in-neighbors.. For a vertex $v \in V$, let $\boldsymbol{e}_v \in \mathbb{R}^V$ be the indicator vector of v, that is, the all-zero vector except the v-th element being one.

3.1 Walk-based similarity

Let $G = (V, E)$ be a directed graph. A *walk* on G is a sequence $(v_0, v_1, \ldots, v_\ell)$ of vertices such that $v_{i-1}v_i \in E$ for every $i \in [\ell]$. We note that $v_0, v_1, \ldots, v_\ell$ are not necessarily distinct; when they are distinct, it is called a *path*.

A *walk-based similarity* [14] is parameterized by a weight function $w \colon E \to \mathbb{R}_+$ on edges and $c \in \mathbb{R}_+$. For a walk $W = (v_0, v_1 \ldots, v_\ell)$ of length ℓ, we define its weight $w_c(W)$ as

$$w_c(W) = c \prod_{i=1}^{\ell} w(v_{i-1}v_i)$$

For two vertices $s, t \in V$, let $\mathrm{walk}(s, t)$ be the (infinite) set of walks from s to t. When $s = t$, we include the path of length 0, which consists only of s, to $\mathrm{walk}(s, s)$.

The similarity $\sigma_{c,w}(s, t)$ from a vertex $s \in V$ to another vertex $t \in V$ is defined as

$$\sigma_{c,w}(s, t) = \sum_{W \in \mathrm{walk}(s,t)} w_c(W),$$

when the sum converges. We omit the subscripts c and w when they are clear from the context.

Although it is difficult to compute $\sigma_{c,w}(s, t)$ based on the definition, we can rephrase it in linear algebraic terms [14]. We define the *transition weight matrix* $H \in \mathbb{R}^{V \times V}$ as $H_{uv} = w(uv)$. If $\sigma_{c,w}(u, v)$ is well-defined for every $(u, v) \in V^2$, then the spectral radius $\rho(H)$ of H must be less than one and $\sigma_{c,w}(s, t)$ can be represented as

$$\sigma_{c,w}(s, t) = \left\langle \boldsymbol{e}_s, c \sum_{\ell=0}^{\infty} H^\ell \boldsymbol{e}_t \right\rangle,$$

where $\sum_{\ell=0}^{\infty} H^\ell = (I - H)^{-1}$ [8]. Because every component of H is nonnegative, it is guaranteed that $\rho(H)$ equals the largest real eigenvalue of H by the Perron-Frobenius theorem.

By changing the weight function $w \colon E \to \mathbb{R}_+$ and the scalar coefficient $c \in \mathbb{R}_+$, we can express various similarity notions.

Example 3.1 (Personalized Pagerank [22]). *Let $T = AD^{-1}$ be the transition matrix, where $A \in \mathbb{R}^{V \times V}$ is the adjacency matrix of G and $D \in \mathbb{R}^{V \times V}$ is the diagonal matrix such that $D_{vv} = d^+(v)$ for every $v \in V$. For $\alpha \in (0, 1)$, the personalized PageRank $\sigma_{\mathrm{PPR}}(s, t)$ from a vertex $s \in V$ to another vertex $t \in V$ is*

$$\sigma_{\mathrm{PPR}}(s, t) = \left\langle \boldsymbol{e}_s, \alpha \sum_{\ell=0}^{\infty} (1 - \alpha)^\ell T^\ell \boldsymbol{e}_t \right\rangle,$$

which equals the probability of staying at t in the stationary distribution of the following random walk process: Starting with s, we move to a random neighbor of the current vertex with probability $1 - \alpha$ and jump back to s with the complement probability α. We can easily check that σ_{PPR} is the walk-based similarity with $c = \alpha$ and $w(uv) = 1/d^+(u)$.

Example 3.2 (Katz Similarity [13]). *For $\beta \in (0, 1)$, the Katz similarity from a source vertex $s \in V$ to another vertex $t \in V$ is*

$$\sigma_{\mathrm{Katz}}(s, t) = \left\langle \boldsymbol{e}_s, \sum_{\ell=0}^{\infty} \beta^\ell A^\ell \boldsymbol{e}_t \right\rangle,$$

where $A \in \mathbb{R}^{V \times V}$ is the adjacency matrix. The sum converges when β is smaller than the reciprocal of the largest eigenvalue of A. Here, too, we can confirm that σ_{Katz} is the walk-based similarity with $c = 1$ and $w(uv) = \beta$.

We note that any walk-based similarity can be seen as a Katz similarity on a weighted graph, that is, $A \in \mathbb{R}^{V \times V}$ is a general non-negative matrix.

In this work, we primarily consider the following *one-to-one* similarity estimation problem:

Definition 3.3 (One-to-one similarity estimation problem). *The one-to-one similarity estimation problem is specified by a graph $G = (V, E)$ with a weight function $w \colon E \to \mathbb{R}_+$ on edges and a scalar coefficient $c \in \mathbb{R}_+$. The goal is to construct an index with which, given a pair of vertices $(s, t) \in V^2$, a threshold $\delta \in (0, 1)$, an accuracy parameter $\epsilon \in (0, 1)$, and a failure probability $p \in (0, 1)$ as a query,*

we can compute $\hat{\sigma}(s,t)$ such that $|\hat{\sigma}(s,t) - \sigma_{c,w}(s,t)| < \epsilon \cdot \sigma_{c,w}(s,t)$ with probability at least $1-p$ if $\sigma_{c,w}(s,t) \geq \delta$. If $\hat{\sigma}(s,t)$ satisfies this, we call $\hat{\sigma}(s,t)$ an (ϵ,δ,p)-estimation of $\sigma_{c,w}(s,t)$.

Because approximating small $\sigma_{c,w}(s,t)$ is quite difficult [18], introducing the threshold parameter δ is crucial.

4 PREVIOUS METHODS AND OVERVIEW OF OUR METHOD

4.1 Previous methods

In this section, we review previous methods used for the one-to-one similarity estimation problem.

Iterative method. Iterative method is a basic approach for solving linear systems of the form $A\boldsymbol{x} = \boldsymbol{b}$, where $A \in \mathbb{R}^{n\times n}$ is a matrix with $\rho(I-A) < 1$ and $\boldsymbol{b} \in \mathbb{R}^n$ is a vector. The Jacobi method is a popular method that iteratively computes

$$\boldsymbol{x}_0 = 0, \quad \text{and} \quad \boldsymbol{x}_i = (I-A)\boldsymbol{x}_{i-1} + \boldsymbol{b} \ (i \geq 1).$$

Then, $\boldsymbol{x}_i$ converges to $\boldsymbol{x} = A^{-1}\boldsymbol{b}$ (Note that A is nonsingular from the assumption $\rho(I-A) < 1$). This method can be directly applied to compute walk-based similarity vector from a vertex $s \in V$ by setting $A = I - H$ (from the convergence condition, $\rho(I-A) = \rho(H) < 1$ is guaranteed) and $\boldsymbol{b} = \boldsymbol{e}_t$.

The time complexity of this algorithm is $O(m \cdot i)$, where i is the number of iterations. Moreover, the error decreases as $O(\rho(I-A)^i)$ [21]. This algorithm can compute similarity values from all the vertices to a target vertex simultaneously. (Note that we can also compute similarity values from a source vertex to all the vertices at once by setting $A = I - H^\top, \boldsymbol{b} = \boldsymbol{e}_s$.) However, this method can be rather inefficient to solve the one-to-one similarity estimation problem because the target vertex (or the source vertex) varies with queries.

Forward and backward push methods. The forward push method was proposed in [2] to compute PPR and then was applied to compute Katz similarities [9]. The forward push method approximates the similarity values from a source vertex $s \in V$ by maintaining two vectors; the *reserve vector* $\boldsymbol{p}_s \in \mathbb{R}^V$ and the *residue vector* $\boldsymbol{r}_s \in \mathbb{R}^V$. Initially, we set $\boldsymbol{p}_s = 0$ and $\boldsymbol{r}_s(v) = \boldsymbol{e}_s$. Then, these vectors are updated so that the following invariant is maintained:

$$\sigma_{c,w}(s,v) = p_s(v) + \sum_{u\in V} r_s(u)\sigma_{c,w}(u,v). \tag{1}$$

A pseudocode of this method is given in Algorithm 1.

To the extent of our knowledge, there is no approximation guarantee for the forward push method. As for time complexity, Andersen *et al.* [2] showed that Algorithm 1 runs in $O\left(\frac{1}{\alpha\delta}\right)$ time for PPR if we use the termination condition $\max_{v\in V} \frac{r_s(v)}{d^+(v)} < \delta$ instead of $r_s(u) > \delta$.

The backward push method [1] is similar to the forward push method, except that it maintains a *reverse similarity vector* $\boldsymbol{q}_t \in \mathbb{R}^V$ for which $q_t(v)$ is supposed to approximate $\sigma_{c,w}(v,t)$. The backward push method also maintains a residue vector $\boldsymbol{r}_t \in \mathbb{R}^V$. The invariant for this method is the following:

$$\sigma_{c,w}(v,t) = q_t(v) + \sum_{u\in V} \sigma_{c,w}(v,u) r_t(u). \tag{2}$$

A pseudocode of this method is given in Algorithm 2.

Algorithm 1 Forward push

Input: source vertex $s \in V$, threshold $\delta > 0$

$p_s(u) \leftarrow 0 \ (u \in V), r_s(s) \leftarrow 1, r_s(u) \leftarrow 0 \ (u \in V \setminus \{s\})$
while $\exists u \in V, r_s(u) > \delta$ **do**
  for $v \in N^+(u)$ **do**
    $r_s(v) \leftarrow r_s(v) + w(uv) r_s(u)$.
  end for
  $p_s(u) \leftarrow p_s(u) + r_s(u), r_s(u) \leftarrow 0$.
end while

Algorithm 2 Backward push

Input: target vertex $t \in V$, threshold $\delta > 0$

$q_t(u) \leftarrow 0 \ (u \in V), r_t(t) \leftarrow 1, r_t(u) \leftarrow 0 \ (u \in V \setminus \{t\})$.
while $\exists v, r_t(v) > \delta$ **do**
  for $u \in N^-(v)$ **do**
    $r_t(u) \leftarrow r_t(v) + w(uv) r_t(v)$.
  end for
  $q_t(v) \leftarrow q_t(v) + r_t(v), r_t(v) \leftarrow 0$.
end while

As opposed to the forward push method, the backward push method on PPR has an approximation guarantee. Andersen *et al.* [1] showed that Algorithm 2 returns a reverse similarity vector $\boldsymbol{q}_t$ such that $|q_t(v) - \sigma_{\mathrm{PPR}}(v,t)| \leq \delta$ for every $v \in V$. Lofgren and Goel [19] showed that, for uniformly chosen target vertex $t \in V$, Algorithm 2 runs in expected $O\left(\frac{1}{\alpha\delta} \cdot \frac{|E|}{|V|}\right)$ time.

Monte Carlo method. Since the PPR of a vertex indicate the probability of staying at the vertex in the stationary distribution of a random walk, it can be approximated with a Monte Carlo method [10]. It should be noted that this approach is directly applicable only to PPR, because the transition weight matrix H is not a stochastic matrix for other similarity notions (even $\|H\|_\infty := \max_{\boldsymbol{v}} \|H\boldsymbol{v}\|_\infty / \|\boldsymbol{v}\|_\infty > 1$ may hold). An extension of the Monte Carlo method to general similarity notions and its shortcomings are discussed later.

The Monte Carlo method for PPR first generates N random walks starting from the source vertex $s \in V$ and then, for each $v \in V$, computes the number of random walks N_v that terminate at v. Then, the similarity $\sigma_{\mathrm{PPR}}(s,v)$ is estimated as $\frac{N_v}{N}$. Since the probability that a random walk terminates at $v \in V$ equals $\sigma_{\mathrm{PPR}}(s,v)$, this estimator is unbiased.

Bidirectional methods. The first bidirectional algorithm for computing the PPR between two vertices, called *FAST-PPR*, was proposed in [18] and then its improved version, called *BiPPR*, was proposed in [17]. We describe the latter in detail since our method is built on it.

BiPPR carefully combines the Monte Carlo method and the backward push method. Suppose that we have performed backward pushes to obtain the reverse similarity vector $\boldsymbol{q}_t \in \mathbb{R}^V$ and the residue vector $\boldsymbol{r}_t \in \mathbb{R}^V$ for the target vertex t. From (2), if we have

estimates $\hat{\sigma}_{c,w}(s,v)$ of $\sigma_{c,w}(s,v)$ for each $v \in V$, we can approximate $\sigma_{c,w}(s,t)$ by

$$\hat{\sigma}'_{c,w}(s,t) := q_t(s) + \sum_{v \in V} \hat{\sigma}_{c,w}(s,v) r_t(v).$$

Now, we use the Monte Carlo method to obtain $\hat{\sigma}_{c,w}(s,v)$. It is reported in [17] that BiPPR is significantly faster than both the Monte Carlo method and the backward push method.

4.2 Limitation of Monte Carlo based methods on general similarity notions

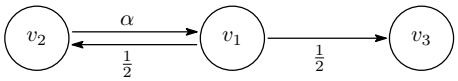


Figure 1: A weighted directed graph for which the variance of the naive Monte Carlo estimator may be infinity

As explained before, the Monte Carlo method cannot be directly used to compute general similarity notions. However, with a slight modification to the Monte Carlo method, we can construct an unbiased estimator for any similarity notion. Unfortunately, its performance is not generally good: The variance of the estimator can be infinite. Below, we explain how we can construct an unbiased estimator and its performance limitation more in detail.

In the case of the PPR, the matrix H itself was used as a transition matrix of a random walk. For a general simliarity notion, we introduce a transition probability matrix P and generate random walks based on P, where P satisfies $\|P\|_\infty \leq 1$ and $P(u,v) > 0$ if and only if $uv \in E$ [7, 23, 29]. Using P, A random walk $W = (v_0, v_1, \ldots, v_\ell)$ starting at vertex u is generated as follows:

(1) $\ell \leftarrow 0, v_0 \leftarrow u$.
(2) Repeat the following:
   - With probability $1 - \sum_{w \in V} P(v_\ell, w)$, terminate the repetition. For every $w \in V$, with probability $P(v_\ell, w)$, $\ell \leftarrow \ell + 1$ and $v_\ell \leftarrow w$.

In general, the distribution of w_ℓ generated this way is not equal to $\sigma_{c,w}(s,v)$. To address this issue, we define *correction weight* $\gamma(W)$ for each walk $W = (v_0, v_1, \ldots, v_\ell)$ as

$$\gamma(W) = \prod_{i=1}^{\ell} \frac{w(v_{i-1}v_i)}{P(v_{i-1}, v_i)}. \tag{3}$$

Suppose that we have generated N walks $W_1, \ldots, W_N$ all starting at s. Then, the Monte Carlo estimation of $\sigma_{c,w}(s,v)$ for every $v \in V$ is given by

$$\hat{\sigma}_{c,w}(s,v) = \frac{1}{N} \sum_{i \in \{1,\ldots,N\}: W_i \text{ ends at } v} \gamma(W_i).$$

We can check that $\hat{\sigma}_{c,w}(s,v)$ is an unbiased estimator of $\hat{\sigma}_{c,w}(s,v)$ regardless of the choice of P.

To use this algorithm, we need to choose P. A naive approach will be to set $P(u,v)$ as the *row-normalized* value of $w(uv)$, that is:

$$P(u,v) = \frac{w(uv)}{c + \sum_{v' \in V, uv' \in E} w(uv')}.$$

However, the variance of the estimator by the Monte Carlo method with this P can be infinite. To see this, we consider the weighted directed graph shown in Figure 1. In Figure 1, the value near the edge from v_i to v_j corresponds to $w(v_iv_j)$. If $\alpha < 2$, the similarity $\sigma_{c,w}(v_1, v_3)$ is well-defined and $\sigma_{c,w}(v_1, v_3) = \frac{c}{2-\alpha}$. We can also see $\hat{\sigma}_{c,w}(v_1, v_3) = \frac{c}{2-\alpha}$. However, $\mathrm{E}[\sigma_{c,w}(v_1, v_3)^2]$ on $N = 1$ is $\mathrm{E}[\sigma_{c,w}(v_1, v_3)^2] = \sum_{i=0}^{\infty} \left(\frac{\alpha(1+c)(\alpha+c)}{2}\right)^i \cdot \frac{(1+c)c^2}{2}$, where i corresponds to the number of visits of a walk to v_2. If $\alpha > \sqrt{2}$, this does not converge regardless of the value of $c > 0$. This also implies that the variance of $\sigma_{c,w}(v_1, v_3)$ is infinite, and the estimation quality cannot be improved by increasing N. Thus, we cannot rely on this estimator for the Monte Carlo estimation of similarity values.

4.3 Overview of our method

Our goal in this study is to obtain efficient methods for the one-to-one similarity estimation problem. To this end, we extend BiPPR [17] to general walk-based similarity notions. As explained, BiPPR combines the Monte Carlo method and the backward push method. Although the backward push method is applicable to general walk-based similarities, the Monte Carlo method cannot be directly applied to general similarities because it relies on the fact that the weights of edges form a transition matrix of a random walk.

To address this problem, we first generalize our definition of walk-based similarities so that different vertices can have different *termination weight*. Then, we introduce a *random-walk reduction* method: From any weight function $w : E \to \mathbb{R}_+$ and scalar coefficient $c \in \mathbb{R}$, this method constructs a pair of a weight function $w' : E \to \mathbb{R}_+$ and a *termination function* $\tau' : V \to \mathbb{R}_+$. The pair (τ', w') is designed so that (i) it forms a transition matrix of a random walk, and (ii) we can easily recover the original similarity value $\sigma_{c,w}(u,v)$ from the similarity value $\sigma_{\tau',w'}(u,v)$ defined by (τ', w').

This reduction method enables us to sample a vertex v with probability proportional to $\sigma_{c,w}(s,v)$ for the source vertex $s \in V$. Thus, we can replace random walks in the Monte Carlo part of BiPPR with this sampling method. Formally, the ability of our reduction method can be stated as follows:

Theorem 4.1. *Let (c,w) be a walk-based similarity on a graph $G = (V,E)$. Then, after preprocessing in $O(m \cdot i)$ time, we can construct an oracle of $O(m)$ space with which, for any given vertex $u \in V$, we can sample a vertex $v \in V$ with probability* almost *proportional to $\sigma_{c,w}(u,v)$. Here, i is the number of iterations in power iteration. The probability with which v is sampled can be made arbitrarily close to $\frac{\sigma_{c,w}(u,v)}{\sum_{v' \in V} \sigma_{c,w}(u,v')}$ by taking a sufficiently large i.*

The idea of buliding an oracle for Monte Carlo estimation of the solution of a linear system $(I - H)\boldsymbol{x} = \boldsymbol{b}$ with $\rho(H) < 1$ but $\|H\|_\infty > 1$ resembles [29]. The algorithm in [29] maintains multiple probability distributions for random walks, while ours require just one. Also, while the algorithm in [29] assigns non-uniform correction weights (similar to (3)), ours enables us to sample walks with probability (almost) proportional to their weights. Thus we can expect that ours have a wider range of applications.

In Section 5, we discuss how to construct (τ', w') from (c,w), and in Section 6, we give a detailed explanation of our algorithm.

5 REDUCTION TO RANDOM WALK

In this section, we describe our generic method that reduces the computation of walk-based similarities to the computation of the stationary distribution of a random walk.

To this end, we slightly generalize the notion of walk-based similarity. For a termination function $\tau\colon V \to \mathbb{R}_+$ and a weight function $w\colon E \to \mathbb{R}_+$, we define the weight of a walk $W = (v_0, \dots, v_\ell)$ as

$$w_\tau(W) = \tau(v_\ell) \prod_{i=1}^{\ell} w(v_{i-1}v_i).$$

Then, we define the similarity $\sigma_{\tau,w}(s,t)$ from a vertex $s \in V$ to a vertex $t \in V$ as

$$\sigma_{\tau,w}(s,t) = \sum_{W \in \mathrm{walk}(s,t)} w_\tau(W).$$

Note that we no longer use the scalar coefficient $c \in \mathbb{R}_+$ because it can be included in τ.

Our goal is to construct other termination function $\tau'\colon V \to \mathbb{R}_+$ and weight function $w'\colon E \to \mathbb{R}_+$ such that

(i) $\sigma_{\tau',w'}(s,\cdot)$ is given by a stationary distribution of a random walk from $s \in V$.
(ii) We can efficiently compute $\sigma_{\tau,w}$ from $\sigma_{\tau',w'}$.

To satisfy (i), τ' and w' must define a transition of a random walk. We first ask for the following condition:

Definition 5.1 (Random walkability). *For functions $\tau\colon V \to \mathbb{R}_+$ and $w\colon E \to \mathbb{R}_+$, we call the pair (τ, w)* randomly walkable *if*

- $\tau(v) \in [0,1]$ *for every* $v \in V$,
- $w(uv) \in [0,1]$ *for every* $uv \in E$, *and*
- $\tau(u) + \sum_{v \in N^+(u)} w(uv) = 1$ *for every* $u \in V$.

Now we discuss (ii). Note that sometimes it is impossible to satisfy $\sigma_{\tau,w}(u,v) = \sigma_{\tau',w'}(u,v)$: If it were possible, then we have $\sum_{v\in V} \sigma_{\tau,w}(u,v) = \sum_{v\in V} \sigma_{\tau',w'}(u,v) = 1$, where the last equality holds because $\sigma_{\tau',w'}$ is given by a stationary distribution of a random walk. However, we may not have $\sum_{v\in V} \sigma_{\tau,w}(u,v) = 1$.

To address the issue above, we consider the following unary quantity

$$c_{\tau,w}(u) = \sum_{v \in V} \sigma_{\tau,w}(u,v), \tag{4}$$

which can be seen as the closeness centrality [5, 6] defined via $\sigma_{\tau,w}$, and we impose the following relaxed condition:

Definition 5.2 (Recoverability). *We say that a pair (τ', w')* recovers *a pair (τ, w) if*

$$\sigma_{\tau',w'}(u,v) = \frac{\sigma_{\tau,w}(u,v)}{c_{\tau,w}(u)} \tag{5}$$

holds for every $u, v \in V$.

The main theorem of this section is as follows:

Theorem 5.3. *For any pair (τ, w), there exists a random-walkable pair (τ', w') that recovers (τ, w).*

We note that, to recover $\sigma_{\tau,w}(u,v)$, we need $c_{\tau,w}(u)$ along with $\sigma_{\tau',w'}(u,v)$. However, computing (an approximation to) $c_{\tau,w}(u)$ is an easier task than directly computing $\sigma_{\tau,w}(u,v)$. We discuss this issue in further detail in Section 6.

Next, for notational simplicity, we write $\sigma(u,v)$ and $\sigma'(u,v)$ to denote $\sigma_{\tau,w}(u,v)$ and $\sigma_{\tau',w'}(u,v)$, respectively. Similarly, we write $c(u)$ and $c'(u)$ to denote $c_{\tau,w}(u)$ and $c_{\tau',w'}(u)$, respectively. The following lemma is useful to guarantee recoverability.

Lemma 5.4. *Suppose that a pair (τ', w') satisfies*

$$w'(W) = \frac{w(W)}{c(u)} \tag{6}$$

for any $u \in V$ and any walk W starting from u. Then, (τ', w') recovers (τ, w).

Proof. The claim holds because

$$\sigma'(u,v) = \sum_{W \in \mathrm{walk}(u,v)} w'(W) = \sum_{W \in \mathrm{walk}(u,v)} \frac{w(W)}{c(u)} = \frac{\sigma(u,v)}{c(u)}.$$

□

Now, we derive the exact form of τ' and w' from the recoverability condition. Considering a walk W consisting of a single vertex $u \in V$, we must have

$$\tau'(u) = \frac{w(W)}{c(u)} = \frac{\tau(u)}{c(u)} \tag{7}$$

Considering walks $W = (v_0, v_1, \dots, v_\ell)$ and $W' = (v_1, \dots, v_\ell)$, we must have

$$w'(v_0v_1) = \frac{w'(W)}{w'(W')} = \frac{w(W)}{c(v_0)} \cdot \frac{c(v_1)}{w(W')} = w(v_0v_1) \cdot \frac{c(v_1)}{c(v_0)} \tag{8}$$

We now show that the pair (τ', w') satisfies the desired conditions.

Lemma 5.5. *The pair (τ', w') satisfies random walkability.*

Proof. We have

$$\tau'(u) + \sum_{v \in N^+(u)} w'(uv) = \frac{\tau(u) + \sum_{v\in N^+(u)} w(uv)c(v)}{c(u)} = 1$$

for every $u \in V$. As $\tau'(u) \geq 0$ and $w'(uv) \geq 0$, it is clear that $\tau'(u) \in [0,1]$ and $w'(uv) \in [0,1]$ hold. □

Lemma 5.6. *The pair (τ', w') recovers (τ, w).*

Proof. By Lemma 5.4, it suffices to check the condition (6). Let $W = (v_0, \dots, v_\ell)$ be an arbitrary walk. Then, we have

$$w'(W) = \prod_{i=1}^{\ell} \left(w(v_{i-1}v_i) \cdot \frac{c(v_i)}{c(v_{i-1})} \right) \cdot \frac{\tau(v_\ell)}{c(v_\ell)} = \frac{w(W)}{c(v_0)}.$$

□

Theorem 5.3 is established by combining Lemmas 5.5 and 5.6.

6 ALGORITHMS

In Section 5, we have seen that computing walk-based similarities can be reduced to computing the stationary distribution of a random walk. To perform random walk, we need to be able to quickly sample random neighbors of a vertex in the weighted graph obtained by the reduction. In Section 6.1, we explain how to implement an oracle with which we can randomly sample neighbors of a vertex, given that we can afford to compute the exact values of $c_{\tau,w}(u)$ (Refer to (4) for the definition). In Section 6.2, we discuss how to implement the oracle when we only have approximations to $c_{\tau,w}(u)$. Finally, in Section 6.3, we describe our algorithm for the one-to-one similarity estimation problem. As in Section 5, we use symbols $\sigma(u,v)$, $\sigma'(u,v)$, $c(u)$, and $c'(u)$ for notational simplicity.

6.1 Random neighbor oracle

In this section, we present the following:

Theorem 6.1. *Let (τ', w') be the pair given in Theorem 5.3. Then, after preprocessing in $O(|V|^3 + |E|)$ time, we can provide an oracle such that, given a query vertex $u \in V$, it returns in $O(1)$ time*

- *a special symbol $\perp$ with probability $\tau'(u)$, and*
- *a vertex $v \in N^+(u)$ with probability $w'(uv)$.*

With the oracle provided in Theorem 6.1, we can perform random walk with respect to the pair (τ', w').

Proof. To obtain (τ', w'), we need to compute $c(u)$ for every $u \in V$. Because the similarity $\sigma(u, v)$ is defined as

$$\sigma(u,v) = \left\langle \boldsymbol{e}_u, \sum_{\ell=0}^{\infty} H^\ell \tau(v) \boldsymbol{e}_v \right\rangle = \left\langle \boldsymbol{e}_u, (I-H)^{-1}\tau(v)\boldsymbol{e}_v \right\rangle,$$

where H is the transition weight matrix defined in Section 3.1. (note that $\sum_{\ell=0}^{\infty} H^\ell \tau(v)\boldsymbol{e}_v$ converges if $\sigma_{\tau,w}(u,v)$ is well defined for every $(u,v) \in V^2$), $c(u)$ can be rephrased as

$$\begin{aligned} c(u) &= \sum_{v \in V} \left\langle \boldsymbol{e}_u, (I-H)^{-1}\tau(v)\boldsymbol{e}_v \right\rangle \\ &= \left\langle \boldsymbol{e}_u, (I-H)^{-1} \sum_{v \in V} \tau(v)\boldsymbol{e}_v \right\rangle = \left\langle \boldsymbol{e}_u, (I-H)^{-1}\boldsymbol{\tau} \right\rangle, \end{aligned}$$

where $\boldsymbol{\tau} \in \mathbb{R}^V$ is a vector with the v-th element being $\tau(v)$. Hence, we can compute $c(u)$ for every $u \in V$ in $O(|V|^3)$ time. Then, we can compute τ' and w' in $O(|V| + |E|)$ time.

Now, we can use Walker's alias method [26] to provide the desired oracle. We need $O(d^+(u))$ time to preprocess each vertex $u \in V$. Hence, the total preprocessing time is $O(|V|^3 + |E|)$. □

6.2 Random neighbor oracle with inexact centrality values

In Section 6.1, we have seen that we can implement a random neighbor oracle by computing $c(u)$. However, this requires $O(|V|^3)$ preprocessing time, which is impractical. To reduce the preprocessing time, in this section, we consider how to implement the random neighbor oracle using approximations $\hat{c}(u)$ to $c(u)$, computed by, for example, the power iteration method.

Considering the proof of Lemma 5.5, the following property was crucial to guarantee for (τ', w') satisfying random walkability:

$$c(u) = \tau(u) + \sum_{v \in N^+(u)} w(uv)c(v) \quad (u \in V). \tag{9}$$

This indicates that we cannot guarantee random walkability if we use approximations $\hat{c}$, instead of c, to define the pair (τ', w').

Therefore, we define another random-walkable pair $(\hat{\tau}', \hat{w}')$ using $\hat{c}$ by slightly changing the recovery condition. First, we define $\hat{c}_R(u)$ by

$$\hat{c}_R(u) = \tau(u) + \sum_{v \in N^+(u)} w(uv)\hat{c}(v) \quad (u \in V).$$

Using this quantity, we define a pair $(\hat{\tau}', \hat{w}')$ as

$$\hat{\tau}'(u) = \frac{\tau(u)}{\hat{c}_R(u)} \quad \text{and} \quad \hat{w}'(uv) = w(uv) \cdot \frac{\hat{c}(v)}{\hat{c}_R(u)}. \tag{10}$$

We can easily confirm the following:

Algorithm 3 Oracle construction

Input: weight function w, termination function τ

1: Compute $\hat{c}$, an approximation of $(I-H)^{-1}\boldsymbol{\tau}$
2: **for** $u \in V$ **do**
3: $\quad \hat{c}_R(u) \leftarrow \tau(u) + \sum_{v \in N^+(u)} w(uv)\hat{c}(u)$
4: $\quad f_u \leftarrow \left\{ \left(v, \frac{w(uv)\hat{c}(v)}{\hat{c}_R(v)} \right) \mid v \in N(u) \right\} \cup \left\{ \left(\perp, \frac{\tau(u)}{\hat{c}_R(v)} \right) \right\}$
5: $\quad \Omega_u \leftarrow$ random sampler that returns v with probability p for every $(v,p) \in f_u$
6: **end for**
7: Return $(\hat{c}, \{\Omega_u\}_{u \in V})$

Algorithm 4 Sampling using the oracle by Algorithm 3

Input: source vertex u, $(\hat{c}, \{\Omega_u\}_{u \in V})$ given by Algorithm 3

1: **loop**
2: $\quad v \leftarrow$ return value of an oracle call to Ω_u
3: $\quad$ Return u if $v = \perp$. Otherwise, $u \leftarrow v$.
4: **end loop**

Proposition 6.2. *The pair $(\hat{\tau}', \hat{w}')$ satisfies random walkability.*

Following the proof of Theorem 6.1, we obtain the following:

Theorem 6.3. *Suppose that we are given a vector $\hat{\boldsymbol{c}} \in \mathbb{R}^V$. Let $(\hat{\tau}', \hat{w}')$ be the pair given by (10). Then, after preprocessing in $O(|V| + |E|)$ time, we can provide an oracle such that, given a query vertex $u \in V$, it returns in $O(1)$ time*

- *a special symbol $\perp$ with probability $\hat{\tau}'(u)$, and*
- *a vertex $v \in N^+(u)$ with probability $\hat{w}'(uv)$.*

The pseudocode of the modified oracle is given in Algorithm 3. The procedure for sampling vertices using the random neighbor oracle is shown in Algorithm 4. It is guaranteed that Algorithm 4 returns a vertex $v \in V$ with probability $\hat{\sigma}'(u,v)$, where $\hat{\sigma}'(u,v) = \sigma_{\hat{\tau}', \hat{w}'}(u,v)$. In Section 7, we discuss the error in using $\hat{\sigma}'(u,v)$ to approximate $\sigma(u,v)$.

6.3 Similarity estimation

We can estimate walk-based similarities by simulating the Monte Carlo part of BiPPR [17] using the random neighbor oracle described in Sections 6.1 and 6.2. However, the transition weights produced by our reduction have different properties from that of PPR:

- Monte Carlo itself only estimates $\hat{\sigma}'(u,v)$. We need to multiply this value by $\hat{c}(u)$ in order to get an estimate of $\sigma(u,v)$, but this may affect the accuracy of the estimation.
- Termination probabilities may be different among vertices. Therefore, the efficiency of push algorithms largely depends on the source vertex.

Hence, we modify the original BiPPR as follows:

Threshold depending on $\hat{c}(u)$. As we see later in Theorem 7.1, $\hat{\sigma}'(u,v)\hat{c}(u)$ is a good approximation to $\sigma(u,v)$. Assume that the difference between $\hat{\sigma}'(u,v)\hat{c}(u)$ and $\sigma(u,v)$ is small enough to be ignored. Then, if $\hat{\sigma}'(u,v)$ is an (ϵ, δ', p)-estimation of $\sigma'(u,v)$, then $\hat{\sigma}'(u,v)\hat{c}(u)$ will be an $(\epsilon, \delta'\hat{c}(u), p)$-estimation of $\sigma(u,v)$. Therefore, if we want to have an (ϵ, δ, p)-estimation of $\sigma_{\tau,w}(u,v)$, we

should set $\delta' = \frac{\delta}{\hat{c}(u)}$. We will use $\frac{\delta}{\hat{c}(u)}$ as the threshold for estimation of $\hat{\sigma}'(u, v)$.

Adaptive thresholding. As indicated in [17, 18], we can dynamically change the threshold for terminating backward pushes. We take a balance between the running time of the backward push part and that of the Monte Carlo part. This technique is especially important for the walk-based similarity estimation, because the cost of backward pushes is difficult to predict and may largely depend on the target vertex. We can estimate the (average) running time of the Monte Carlo part as the product of the average walk time and the number of required walks.

We note that the average walk length may vary depending on the starting vertex. Therefore, we computed the average walk time (using Monte Carlo sampling) from the source vertex at the beginning of each query computation. We also note that the walks obtained in this step can be reused in later steps.

Integrating forward push into Monte Carlo. Instead of generating walks from the source vertex, we can perform forward pushes and then generate walks from each vertex with positive residuals [28].

Suppose we want to generate N walks in the Monte Carlo part for estimation of $\hat{\sigma}'$. Let $\boldsymbol{p}_s$ and $\boldsymbol{r}_s$ be the reserve and residue vectors after running forward push algorithm, respectively. From the invariant of forward push (1), if we have unbiased estimations of $\hat{\sigma}'(u, v)$ for every $u \in V$, then $\hat{\sigma}(s, v)$ defined by the following formula is also an unbiased estimation of $\hat{\sigma}'(s, v)$:

$$\hat{\sigma}(s, v) = p_s(v) + \sum_{u \in V} r_s(u)\hat{\sigma}'(u, v).$$

Here, $\hat{\sigma}'(u, v)$ can be obtained by Monte Carlo estimation from vertex u. If we use at least $\lceil r_s(u)N \rceil$ walks for this estimation for every $u \in V$, we can show that the variance of $\hat{\sigma}(s, v)$ defined above is at most that of the estimator by naive Monte Carlo with N walks. With this technique, we can reduce the cost of tracking many identical walks in the Monte Carlo part.

Final algorithm. Combining the techniques discussed above, we obtain the final algorithm for similarity estimation. This algorithm is shown in Algorithm 5. Here γ is a parameter for controlling accuracy and ι is the number of trials of generating walks for estimating the walk time. Note that this requires the random neighbor oracle generated by Algorithm 3. We can precompute this oracle and use it for answering each query to reduce the computation cost.

In Algorithm 5, we used the terminate condition $N_0 \max_u r_s(u) < 1$. This means that we terminate when the number of walks to be generated from every vertex is less than 1. As this condition may cause too many forward pushes, in Section 8, we also consider a heuristic that uses a terminate condition depending on $\hat{c}(s)$.

As with the original BiPPR [17], we can expect that Algorithm 5 significantly improves the efficiency of similarity estimation. However, as we are considering general walk-based similarity, it is difficult to give upper bounds for running time of this algorithm. Analyzing the time complexity will be a future work.

Algorithm 5 Similarity estimation

Input: source vertex s, target vertex t, threshold δ, parameter γ and ι, $(\hat{c}, \{\Omega_u\}_{u \in V})$ given by Algorithm 3

$\delta' \leftarrow \frac{\delta}{\hat{c}(s)}$
$\tau_{\mathrm{walk}} \leftarrow$ (Time to generate ι walks from s)$/\iota$
Perform Algorithm 2 with the following terminate condition:
  Let τ_{push} be the time required for backward push so far. Terminate if $\frac{\gamma \max_v r_t(v)}{\delta'} \cdot \tau_{\mathrm{walk}} < \tau_{\mathrm{push}}$
$N_0 \leftarrow \lceil \frac{\gamma \max_v r_t(v)}{\delta'} \rceil$
$\hat{\sigma} \leftarrow q_t(s)$
Perform Algorithm 1 using r'_s instead of r_s with the terminate condition of $N_0 \max_u r'_s(u) < 1$
for $u \in V, p_s(u) > 0$ **do**
  $\hat{\sigma} \leftarrow \hat{\sigma} + p_s(u) r_t(u)$
end for
for $u \in V, r'_s(u) > 0$ **do**
  $\omega \leftarrow \frac{\hat{c}(u)}{\hat{c}(s)} N_0 r'_s(u),\ N \leftarrow \lceil \omega \rceil$
  for $i = 1, 2, \ldots, N$ **do**
    $v \leftarrow$ vertex returned by a call to Algorithm 4 with source s and oracle $(\hat{c}, \{\Omega_u\}_{u \in V})$
    $\hat{\sigma} \leftarrow \hat{\sigma} + \frac{\omega}{N} \cdot \frac{1}{N_0} \hat{c}(s) r_t(v)$
  end for
end for
Return $\hat{\sigma}$ as an estimation of $\sigma_{\tau, w}(s, t)$

7 ERROR ANALYSIS

In this section, we provide theoretical guarantees of our algorithm.

First, we analyze the approximation error caused by using the inexact random neighbor oracle given in Section 6.2. We borrow notations from Section 6. Then, we show that $\hat{\sigma}'(u, v)$ and $\sigma(u, v)$ are close:

Theorem 7.1. *Suppose that*

$$\frac{1}{1+\kappa} \hat{c}(u) \le \hat{c}_R(u) \le (1+\kappa)\hat{c}(u) \quad (u \in V) \tag{11}$$

holds for some $\kappa \in (0, 1)$ *and that* $\lambda := \min_{u \in V}(1 - \hat{\tau}'(u)) < \frac{1}{1+\kappa}$ *holds. Then, we have*

$$\underline{\alpha}(\sigma(u, v))^{1+\beta} \le \hat{\sigma}'(u, v)\hat{c}(u) \le \overline{\alpha}(\sigma(u, v))^{1-\beta}$$

where

$$\underline{\alpha} = \frac{(1-\lambda)^{1+\beta}}{(1+\kappa) - \lambda},\ \overline{\alpha} = \frac{(1-\lambda)^{1-\beta}}{(1+\kappa)^{-1} - \lambda},\ \textit{and } \beta = \log_{\frac{1}{\lambda}}(1+\kappa).$$

Before proving Theorem 7.1, we discuss its implication. Theorem 7.1 claims that the relative error of $s_{\hat{\tau}', \hat{w}'}(u, v)\hat{c}_R(u)$ is controlled by κ. The following proposition states that we can make κ arbitrarily small if we use the power iteration method to compute $\hat{c}_R(u)$.

Proposition 7.2. *Let* $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots$ *be a sequence that converges to* $\boldsymbol{x} = (I - H)^{-1}\boldsymbol{\tau}$, *where* $H \in \mathbb{R}^{V \times V}$ *is a transition weight matrix with* $\rho(H) < 1$. *Then, for any* $\kappa > 0$, *there exists* i *such that*

$$\frac{1}{1+\kappa} x_i(u) \le \tau(u) + \sum_v H(u, v) x_i(v) \le (1+\kappa) x_i(u) \tag{12}$$

holds for every $u \in V$ *with* $x(u) > 0$.

Proof. Let $\mu = \min\{x(u) \mid u \in V,\ x(u) \neq 0\}$. From the convergence of $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots$, there exists i such that $\|\boldsymbol{x}_i - \boldsymbol{x}\|_\infty < \delta'$ where $\delta' = \frac{\delta\mu}{2(1+\|H\|_\infty)}$. Then, for $u \in V$ with $x(u) > 0$, we have

$$\begin{aligned}
&\tau(u) + \sum_v H(u,v)x_i(v) \\
&= \tau(u) + \sum_v H(u,v)x(v) + \sum_v H(u,v)(x_i(v) - x(v)) \\
&\geq x(u) - \|H\|_\infty\delta' \geq x_i(u) - (1 + \|H\|_\infty)\delta' \\
&\geq \frac{1}{1+\delta}x_i(u)
\end{aligned}$$

as desired. The second inequality of (12) follows analogously. □

If $\boldsymbol{x}_i$ satisfies (12) and we choose $\hat{\boldsymbol{c}} = \boldsymbol{x}_i$, then (11) is satisfied and the approximation guarantee given in Theorem 7.1 follows. Hence, we can use (12) as a terminate condition in the power iteration.

To prove Theorem 7.1, we need the following technical lemma, whose proof is deferred to the full version:

Lemma 7.3. *Let $\lambda, \kappa \in (0,1)$ be such that $\lambda(1+\kappa) < 1$. Consider two sequences $\{a_i\}_{i\geq 0}, \{b_i\}_{i\geq 0}$ such that $a_i \in [0, \lambda^i]$ and $b_i \in [a_i(1+\kappa)^{-i}, a_i(1+\kappa)^i]$ for every $i \geq 0$. Let $s = \sum_{i=0}^\infty a_i < \infty$. Then, we have*

$$\frac{(1-\lambda)^{1-\eta}}{1-\lambda(1+\kappa)^{-1}}s^{1-\eta} \leq \sum_{i=0}^\infty b_i \leq \frac{(1-\lambda)^{1+\eta}}{1-\lambda(1+\kappa)}s^{1+\eta}. \tag{13}$$

for $\eta = \log_\lambda(1+\kappa)$.

Proof of Theorem 7.1. Let $\sigma_\ell(u,v)$ denote the sum of weights of walks from u to v of length ℓ, that is,

$$\sigma_\ell(u,v) = \sum_{W \in \mathrm{walk}_\ell(u,v)} w_\tau(W).$$

Then, clearly $\sigma(u,v) = \sum_{\ell=0}^\infty \sigma_\ell(u,v)$ holds. Similarly, we define $\sigma_\ell(u) = \sum_{v\in V}\sigma_\ell(u,v)$.

First, we analyze the relation between $w'(W) := w_{\tau'}(W)$ (which is obtained using $\boldsymbol{s}$) and $\hat{w}'(W) := w_{\hat{\tau}'}(W)$ for a walk $W = (v_0, v_1, \ldots, v_\ell)$. By the definition of $\hat{w}'(u,v)$ and $(\hat{\tau}', \hat{w}')$, we have

$$\begin{aligned}
\hat{w}'(W) &= \hat{\tau}'(v_\ell)\prod_{i=1}^\ell \hat{w}'(v_{i-1}v_i) = \frac{\tau(v_\ell)}{\hat{c}_R(v_\ell)}\prod_{i=1}^\ell\left(w(v_{i-1}v_i)\cdot\frac{\hat{c}(v_i)}{\hat{c}_R(v_{i-1})}\right) \\
&= \left(\frac{\tau(v_\ell)}{\hat{c}_R(v_0)}\prod_{i=1}^\ell w(v_{i-1}v_i)\right)\cdot\prod_{i=1}^\ell\frac{\hat{c}(v_i)}{\hat{c}_R(v_i)} = \frac{w_\tau(W)}{\hat{c}_R(v_0)}\cdot\prod_{i=1}^\ell\frac{\hat{c}(v_i)}{\hat{c}_R(v_i)}.
\end{aligned}$$

Taking the sum of $\hat{w}'(W)$ over all walks W from u to v of length ℓ, from the assumption on $\hat{c}_R(u)$, we have

$$\sigma_\ell(u,v)(1+\kappa)^{-\ell} \leq \hat{\sigma}'_\ell(u,v)\hat{c}_R(u) \leq \sigma_\ell(u,v)(1+\kappa)^\ell.$$

From random walkability, we have

$$0 \leq \hat{\sigma}'_\ell(u,v) \leq \hat{c}'_\ell(u) \leq \lambda^\ell \text{ and } \sum_{\ell=0}^\infty \hat{c}'_\ell(u) = \hat{c}'(u) = 1.$$

Therefore, using Lemma 7.3 and (11), we obtain

$$\underline{\alpha}\cdot\sigma(u,v)^{1+\beta} \leq \hat{\sigma}'(u,v)\hat{c}(u) \leq \overline{\alpha}\cdot\sigma(u,v)^{1-\beta}.$$ □

Next, we give a guarantee on the estimation quality of Algorithm 5.

Theorem 7.4. *Suppose that (11) holds for a sufficiently small κ. Then, Algorithm 5 returns an (ϵ,δ,p)-estimation of $\sigma(s,t)$ if we set $\gamma = \Omega(\epsilon^{-2}\log(1/p))$.*

We show this theorem using the following lemma:

Lemma 7.5. *Suppose that $\hat{c} = c$ holds. Then, Algorithm 5 returns an (ϵ,δ,p)-estimation of $\sigma(s,t)$ if we set $\gamma = \Omega(\epsilon^{-2}\log(1/p))$*

Proof. First, we show the claim holds if we modify Algorithm 5 so that we do not perform forward pushes, and instead we use $p_s(u) = 0$ $(u \in V)$ and $r_s(u) = \begin{cases} 1 & u = s \\ 0 & u \neq s \end{cases}$. In this case, the proof is almost identical to that of the original BiPPR given in [17]. Let $r_{\max} = \max_v r_t(v)$, v_i be the vertex chosen in the i-th iteration and $X_i = \frac{r_t(v_i)}{r_{\max}}$. Then,

$$\hat{\sigma} = q_t(s) + \frac{c(s)}{N_0}\sum_{i=1}^{N_0} r_t(v_i) = q_t(s) + \frac{c(s)r_{\max}}{N_0}\sum_{i=1}^{N_0} X_i.$$

We can analyze the concentration of $\frac{1}{N_0}\sum_{i=1}^{N_0} X_i$ using the Chernoff bound and thus obtain the desired statement.

For the complete proof for Algorithm 5, we return to the proof of the Chernoff bound (see, e.g., [11]). The Chernoff bound can be proven by bounding $\Pr[e^{\lambda X} > e^{\lambda a}]$ $(\lambda > 0)$ using Markov inequality

$$\Pr[e^{\lambda X} > e^{\lambda a}] < \frac{\mathrm{E}[e^{\lambda X}]}{e^{\lambda a}}$$

and optimizing λ. Therefore, if two random variables X, Y satisfy $\mathrm{E}[e^{\lambda X}] \leq \mathrm{E}[e^{\lambda Y}]$ for any $\lambda > 0$, Chernoff bounds satisfied by Y are all satisfied by X.

Let $v(s)$ be a random vertex that is chosen with probability equal to the similarity from s and

$$\Phi = \prod_{u\in V} e^{\lambda p_s(u)\frac{c(u)}{c(s)}\frac{r_t(u)}{r_{\max}}}\cdot\mathrm{E}\left[e^{\lambda\frac{c(u)}{c(s)}\frac{r_t(v(u))}{r_{\max}}}\right]^{N_0 r'_s(u)}$$

$$\Phi' = \prod_{u\in V} e^{\lambda p_s(u)\frac{c(u)}{c(s)}\frac{r_t(u)}{r_{\max}}}\cdot\mathrm{E}\left[e^{\lambda\frac{c(u)}{c(s)}\frac{r_t(v(u))}{r_{\max}}\frac{N_0 r'_s(u)}{\lceil N_0 r'_s(u)\rceil}}\right]^{\lceil N_0 r'_s(u)\rceil}.$$

It can be verified that $\Phi' = \mathrm{E}[e^{\lambda(\hat{\sigma}-q_t(s))}]$. Before performing forward pushes, $\Phi = \Phi_0 = \mathrm{E}\left[e^{\lambda\sum_{i=1}^{N_0}X_i}\right]$. From the independence of $X_1, X_2, \ldots$, we have $\Phi_0 = \left(\mathrm{E}\left[e^{\lambda X}\right]\right)^{N_0}$, where $X = \frac{r_t(v(s))}{r_{\max}}$. We can show that Φ is non-increasing in a forward push operation by the convexity of e^x. Furthermore, $\Phi' \leq \Phi$ can be proven using Hölder's inequality. Finally, we have $\Phi' \leq \Phi_0$ and thus we obtain the desired statement for Algorithm 5 with forward push when $\hat{c} = c$. □

Proof of Theorem 7.4. For simplicity, we modify the line 5 of Algorithm 5 to $N_0 \leftarrow \frac{\gamma\max_v r_t(v)}{\delta'}$ in the analysis.

Suppose the vertices pushed in the execution of forward push in Algorithm 5 are $s = u_1, u_2, \ldots, u_k$ in this order. Then, we construct a graph $G' = (V', E')$ from $G = (V, E)$ in the following way:

(1) We let $V' \leftarrow V, E' \leftarrow E$ and define a mapping $f : V \to V'$ by $f(u) = u$.

Table 1: Tested networks and precomputation time

Name	Type	n	m	precomputation time (s)
com-dblp	Undirected	317K	2.10M	1.289
web-Stanford	Directed	282K	2.31M	1.060
com-youtube	Undirected	1.13M	5.98M	4.193
soc-pokec	Directed	1.63M	30.6M	16.517
soc-LiveJournal	Directed	4.85M	69.0M	40.555
com-orkut	Undirected	3.07M	234M	107.122

(2) For each $i = k, k-1, \ldots, 1$, we create a fresh vertex x_i. Then, we let $V' \leftarrow V' \cup \{x_i\}$, $E' \leftarrow E' \cup \{(x_i, f(v)) \mid v \in N^+(u_i)\}$, $\hat{c}(x_i) \leftarrow \tau(u_i) + \sum_{v \in N^+(u)} w(u_i v)\hat{c}(f(v))$, $\hat{\tau}'(x_i) = \frac{\tau(u_i)}{\hat{c}(x_i)}$, $w(x_i v) = \hat{w}'(x_i v) = w(u_i v) \cdot \frac{\hat{c}(v)}{\hat{c}(x_i)}$ and finally, $f(u_i) \leftarrow x_i$.

Note that this construction of $\hat{c}, \hat{\tau}'$ and $\hat{w}'$ is well-defined since $\hat{c}(x_i)$ depends only on $x_{i+1}, \ldots, x_k$ and $u \in V$.

Suppose we run Algorithm 5 on G' instead of G, with the sequences of pushed vertices as follows:

- Suppose pushed vertices in the forward push of the run for G are $v_1, \ldots, v_l$ and let $f(v) = \{v\} \cap \{x_i \mid u_i = v\}$. Then, those for G' are $f(v_1), \ldots, f(v_l)$. Here, vertices in $f(v_i)$ for one i are pushed *at the same time.*
- Pushed vertices in the backward push are $x_1, \ldots, x_k$ in this order.

Then, the distribution of $\hat{\sigma}$ is identical for the runs for G and G'.

We can analyze the concentration of Algorithm 5 for G' (and the sequence of pushed vertices as described above) using the same way as in the proof of Lemma 7.5, because the invariant (1) is maintained in forward pushes for this setting. Thus we can argue that, if we set $\gamma = \Omega(\epsilon^{-2} \log(1/p))$, Algorithm 5 returns an (ϵ, δ, p)-estimation of $\mathrm{E}[\hat{\sigma}]$. Therefore, it suffices to show that $\mathrm{E}[\hat{\sigma}]$ can be made arbitrarily close to $\sigma(s,t)$ by choosing sufficiently small κ. Since we can obtain the same bounds for $\mathrm{E}[\hat{\sigma}]$ as those for $\hat{c}_R(u)$ in Theorem 7.1, the claim holds. □

8 EXPERIMENTS

We experimentally evaluated our algorithms for one-to-one walk-based similarity estimation. The primary objective of this experiment is to show that our algorithms are efficient in the following extreme condition, which previous algorithms are difficult to handle: $\max_i \sum_j H(i,j)$ and $\max_j \sum_i H(i,j)$ are significantly larger than 1, where H is the transition weight matrix defined in Section 3.1.

Katz similarity with β close to $\frac{1}{\rho(A)}$ satisfies the above condition. Also, recall that Katz similarity can express any walk-based similarity by appropriately setting weights on edges. Hence, in this experiment, we used Katz similarity as a representative similarity notion and set $\beta = \frac{1}{1.1\rho(A)}$. Datasets used, all taken from SNAP [16], are shown in Table 1. Our codes are provided in the supplementary material.

Query generation. For each tested network, one-to-one similarity queries were generated as follows. First, we randomly chose the source vertex s uniformly or with probability proportional to Katz similarity values. Then, we computed similarity vector from s using power iteration. Finally, we chose the target vertex $t \neq s$ uniformly at random such that $\sigma_{\mathrm{Katz}}(s,t) \geq \delta_{\mathrm{index}} := \frac{1}{10n}$. If no such t was found, we returned to the first step and chose another s. We repeated these procedures until 1000 queries were obtained.

Tested algorithms. We implemented the following algorithms;

- Power iteration (PI).
- Iterative method with upper/lower bound guarantees (IB) [9].
- Forward push (FP).
- BiPPR with naive Monte Carlo for general path-based similarities (described in Section 4.2) (NBP).
- Bidirectional algorithm (BI, ours): Algorithm 5 with forward push disabled.
- BI with forward push (BIF, ours): Algorithm 5.
- BIF with a modified terminate condition (BIF', ours): Algorithm 5. We used $N_0 \max_u r_s(u) < \hat{c}(s)$ as the terminate condition in Line 7 of Algorithm 5.

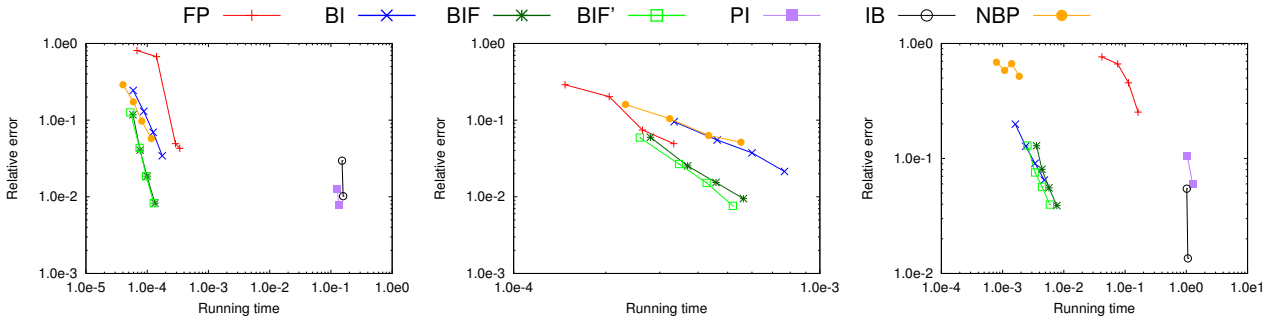
We implemented these algorithms in C++11 and ran them on a Linux (Ubuntu 16.04) desktop with Intel Core i7-6700 processor (3.4GHz) and 16 GiB of RAM.

Note that IB is applicable only to undirected networks. Furthermore, because PI and IB require iterations over all edges and thus take long execution time, we only ran them on com-dblp and com-youtube, which are relatively small.

To examine the relationships between execution time and estimation quality, we executed these algorithms with varying termination parameters δ. For BI, BIF, BIF' and NBP, we used $\gamma = 7$ as the original BiPPR [17]. Note that changing δ to $\frac{\delta}{c}$ is equivalent to changing γ to $c\gamma$. Therefore, using $\delta \neq \delta_{\mathrm{index}}$ is effectively equivalent to use $\delta = \delta_{\mathrm{index}}$ with different values of ϵ and p from the specified ones. For PI, the termination condition was set to $\|\boldsymbol{x}_i - \boldsymbol{x}_{i-1}\|_\infty < \delta$. For IB, we terminated iterations once the difference between upper and lower bounds on the similarity value became less than δ. For BI and BIF, we precomputed to provide the random neighbor oracle and used it in the query time.

Results. First, we report the precomputation time for generating the random neighbor oracle required for BI and BIF (see Table 1). The precomputation time is less than 2 min even for a graph with more than 200M edges (com-orkut), which is small, given that precomputation is required only once for each graph. Note that existing methods for similarity estimation such as BiPPR [17] and HubPPR [27] also use precomputation. In particular, it is reported that the latter method required more than 6,000 seconds for preprocessing com-orkut [27].

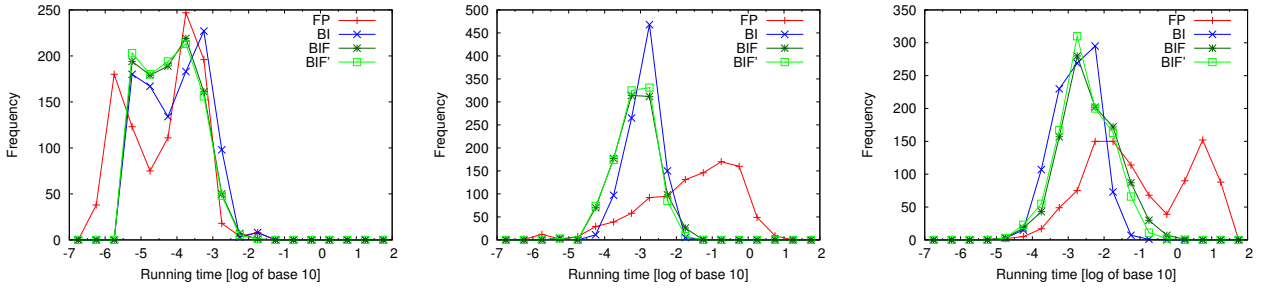
In Figure 2, we report the trade-off between the average relative error and the average running time for each instance. Except for web-Stanford, our proposed algorithms (BI, BIF, and BIF') outperform other baseline methods. The improvement is prominent in larger networks such as soc-pokec and com-orkut. In particular, for com-orkut, our algorithms achieved approximately 100 times faster running time and better relative errors than FP. BIF and BIF' also showed better error-time trade-offs than FP for web-Stanford. Additionally, BIF' consistently shows better performance than BIF. This suggests that controlling the amount of forward pushes is effective to improve performance. Finally, the performance of NBP significantly varies on different networks. Specifically, in com-youtube,



(a) com-dblp (b) web-Stanford (c) com-youtube

(d) soc-pokec (e) soc-LiveJournal (f) com-orkut

Figure 2: Average relative error vs. Average running time



(a) web-Stanford (b) soc-pokec (c) com-orkut

Figure 3: Distribution of running time

soc-pokec and com-orkut, increased running time (corresponding to smaller γ) does not necessarily improve the relative error. This implies that the performance of NBP can be unstable.

Figure 3 shows the distribution of query processing times. Parameters may be different among algorithms to equalize the relative error as much as possible. In this figure, a query with processing time t (in seconds) is assigned to the bin of width 0.5 containing the index $\log_{10} t$. We can observe that the processing time largely varies depending on the queries. As we have discussed before, vertices with larger centrality values tend to require more iterations. The query processing times of BI, BIF, and BIF' are more concentrated than that of FP except for BI in Figure 3a. This means that our algorithms are significantly faster than FP for difficult queries.

9 CONCLUSION

We proposed a *random-walk reduction* method that reduces the computation of any walk-based similarity to the computation of a stationary distribution of a random walk. With this reduction, we can exploit techniques for PPR to compute other walk-based similarities. As a concrete application, we designed an indexing method for walk-based similarities with which we can quickly estimate the similarity value of queried pairs of vertices, and theoretically analyze its approximation error. Our experimental results demonstrate that the instantiation of our method for Katz similarity is two orders of magnitude faster than existing methods on large real networks, without any deterioration in solution quality.

ACKNOWLEDGMENTS

S. Murai would like to thank Hiroshi Imai for offering insightful comments and discussions.

REFERENCES

[1] Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. 2007. Local Computation of PageRank Contributions. In *Proceedings of the 5th International Workshop on Algorithms and Models for the Web-Graph (WAW)*. 150–165.

[2] R. Andersen, F. Chung, and K. Lang. 2006. Local Graph Partitioning using PageRank Vectors. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 475–486.

[3] Shumeet Baluja, Rohan Seth, D Sivakumar, Yushi Jing, Jay Yagnik, Shankar Kumar, Deepak Ravichandran, and Mohamed Aly. 2008. Video suggestion and discovery for youtube: taking random walks through the view graph. In *Proceeding of the 17th International Conference on World Wide Web (WWW)*. 895–904.

[4] Siddhartha Banerjee and Peter Lofgren. 2015. Fast Bidirectional Probability Estimation in Markov Models. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems (NIPS)*. 1423–1431.

[5] Alex Bavelas. 1950. Communication Patterns in TaskâĂŘOriented Groups. *The Journal of the Acoustical Society of America* 22, 6 (1950), 725–730.

[6] Murray A. Beauchamp. 1965. An improved index of centrality. *Behavioral Science* 10, 2 (1965), 161–163.

[7] Michele Benzi, Thomas M Evans, Steven P Hamilton, Massimiliano Lupo Pasini, and Stuart R Slattery. 2017. Analysis of Monte Carlo accelerated iterative methods for sparse linear systems. *Numerical Linear Algebra with Applications* 24, 3 (2017), e2088.

[8] Ivan Dimov, Sylvain Maire, and Jean Michel Sellier. 2015. A new Walk on Equations Monte Carlo method for solving systems of linear algebraic equations. *Applied Mathematical Modelling* 39, 15 (2015), 4494–4510.

[9] Pooya Esfandiar, Francesco Bonchi, David F. Gleich, Chen Greif, Laks V. S. Lakshmanan, and Byung-Won On. 2010. Fast Katz and Commuters: Efficient Estimation of Social Relatedness in Large Networks. In *Proceedings of the 7th International Workshop on Algorithms and Models for the Web-Graph (WAW)*. 132–145.

[10] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. 2005. Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments. *Internet Mathematics* 2, 3 (2005), 333–358.

[11] Michel Goemans. 2014. MIT Mathematics 18.310, Lecture Notes: Chernoff bounds, and some applications. https://ocw.mit.edu/courses/mathematics/18-310-principles-of-discrete-applied-mathematics-fall-2013/lecture-notes/MIT18_310F13_Ch4.pdf.

[12] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. 1992. Using collaborative filtering to weave an information tapestry. *Commun. ACM* 35, 12 (1992), 61–70.

[13] Leo Katz. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18, 1 (1953), 39–43.

[14] Samamon Khemmarat and Lixin Gao. 2016. Fast Top-K Path-Based Relevance Query on Massive Graphs. *IEEE Transactions on Knowledge and Data Engineering* 28, 5 (2016), 1189–1202.

[15] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. 1997. GroupLens: applying collaborative filtering to Usenet news. *Commun. ACM* 40, 3 (1997), 77–87.

[16] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[17] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2016. Personalized PageRank Estimation and Search: A Bidirectional Approach. In *Proceedings of the 9th ACM International Conference on Web Search and Data Mining (WSDM)*. 163–172.

[18] Peter Lofgren, Siddhartha Banerjee, Ashish Goel, and Seshadhri Comandur. 2014. FAST-PPR: scaling personalized pagerank estimation for large graphs. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1436–1445.

[19] Peter Lofgren and Ashish Goel. 2013. Personalized PageRank to a Target Node. *CoRR* abs/1304.4658 (2013).

[20] Linyuan Lu and Tao Zhou. 2010. Link Prediction in Complex Networks: A Survey. *CoRR* abs/1010.0725 (2010).

[21] Takanori Maehara, Takuya Akiba, Yoichi Iwata, and Ken-ichi Kawarabayashi. 2014. Computing Personalized PageRank Quickly by Exploiting Graph Structures. *PVLDB* 7, 12 (2014), 1023–1034.

[22] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report. Stanford InfoLab.

[23] Natalia Rosca. 2006. Monte carlo methods for systems of linear equations. *Studia Univ. BABES-BOLYAI Mathematica* 51 (2006).

[24] Nitin Shyamkumar, Siddhartha Banerjee, and Peter Lofgren. 2016. Sublinear estimation of a single element in sparse linear systems. In *Proceedings of the 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. 856–860.

[25] Han Hee Song, Tae Won Cho, Vacha Dave, Yin Zhang, and Lili Qiu. 2009. Scalable proximity estimation and link prediction in online social networks. In *Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference (IMC)*. 322–335.

[26] Alastair J. Walker. 1977. An Efficient Method for Generating Discrete Random Variables with General Distributions. *ACM Trans. Math. Softw.* 3, 3 (Sept. 1977), 253–256.

[27] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. HubPPR - Effective Indexing for Approximate Personalized PageRank. *PVLDB* 10, 3 (2016), 205–216.

[28] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 505–514.

[29] Tao Wu and David F Gleich. 2016. Multi-way Monte Carlo Method for Linear Systems. *arXiv preprint arXiv:1608.04361* (2016).

[30] Yubao Wu, Ruoming Jin, and Xiang Zhang. 2016. Efficient and Exact Local Search for Random Walk Based Top-K Proximity Query in Large Graphs. *IEEE Transactions on Knowledge and Data Engineering* 28, 5 (2016), 1160–1174.