The VLDB Journal (2018) 27:127–152
https://doi.org/10.1007/s00778-017-0490-5

REGULAR PAPER

Second-order random walk-based proximity measures in graph analysis: formulations and algorithms

Yubao Wu[1] · Xiang Zhang[2] · Yuchen Bian[2] · Zhipeng Cai[1] · Xiang Lian[3] · Xueting Liao[1] · Fengpan Zhao[1]

Received: 8 May 2017 / Revised: 19 September 2017 / Accepted: 31 October 2017 / Published online: 1 December 2017

Abstract Measuring the proximity between different nodes is a fundamental problem in graph analysis. Random walk-based proximity measures have been shown to be effective and widely used. Most existing random walk measures are based on the first-order Markov model, i.e., they assume that the next step of the random surfer only depends on the current node. However, this assumption neither holds in many real-life applications nor captures the clustering structure in the graph. To address the limitation of the existing first-order measures, in this paper, we study the second-order random walk measures, which take the previously visited node into consideration. While the existing first-order measures are built on node-to-node transition probabilities, in the second-order random walk, we need to consider the edge-to-edge transition probabilities. Using incidence matrices, we develop simple and elegant matrix representations for the second-order proximity measures. A desirable property of the developed measures is that they degenerate to their original first-order forms when the effect of the previous step is zero. We further develop Monte Carlo methods to efficiently compute the second-order measures and provide theoretical performance guarantees. Experimental results show that in a variety of applications, the second-order measures can dramatically improve the performance compared to their first-order counterparts.

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00778-017-0490-5) contains supplementary material, which is available to authorized users.

✉ Yubao Wu
ywu28@gsu.edu

Xiang Zhang
xzhang@ist.psu.edu

Yuchen Bian
yub31@ist.psu.edu

Zhipeng Cai
zcai@gsu.edu

Xiang Lian
xlian@kent.edu

Xueting Liao
xliao3@student.gsu.edu

Fengpan Zhao
fzhao6@student.gsu.edu

1 Department of Computer Science, Georgia State University, Atlanta, GA, USA

2 College Information Sciences and Technology, The Pennsylvania State University, State College, PA, USA

3 Department of Computer Science, Kent State University, Kent, OH, USA

Keywords Second-order random walk · Proximity measure · Graph mining · PageRank · SimRank

1 Introduction

A fundamental problem in graph analysis is to measure the *proximity* (or closeness) between different nodes. It serves as the basis of many advanced tasks such as ranking and querying [18,34,40,46], community detection [2,42], link prediction [28,30], and graph-based semi-supervised learning [48,49].

Designing effective proximity measures is a challenging task. The simplest notation of proximity is based on the shortest path or the network flow between two nodes [8]. Random walk-based measures have recently been shown to be effective and widely used in various applications. The basic idea is to allow a surfer to randomly explore the graph. The probabilities of the nodes being visited by the random surfer are used to measure the importance of the nodes or the proximity

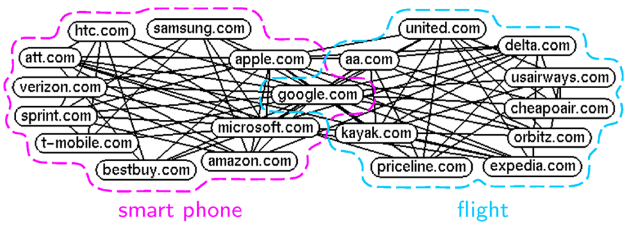

Fig. 1 An example of the web domain graph

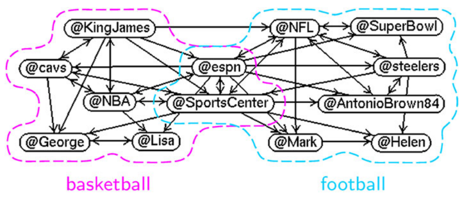

Fig. 2 An example of the Twitter follower network

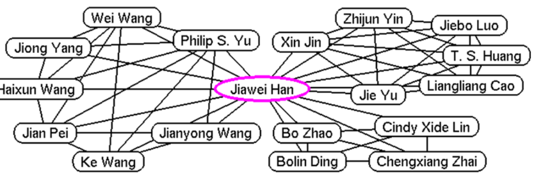

Fig. 3 An example of the research collaboration network

between different nodes. The most commonly used random walk-based proximity measures include PageRank [34], personalized PageRank [40], and SimRank [18].

Most existing random walk measures are based on the first-order Markov model [24], i.e., they assume the next node to be visited only depends on the current node and is independent of the previous step. However, this assumption does not hold in many real-life applications. For example, consider the clickstream data which records the sequences of web domains visited by individual users [5]. The existing first-order random walk measures assume that the next page a user will visit only depends on the current page and is independent on the previous page the user has visited. This is clearly not true.

Figure 1 shows a subgraph of the real-life web domain graph [26].[1] Each node in the graph represents a domain, and two domains share an edge if there are hyperlinks between them. The domains in the graph form two communities. The domains in the left community are about smart phones, and those in the right community are about flights. Suppose the random surfer is currently on google.com and the previously visited node is apple.com, i.e., the surfer came from the smart phone community. The existing first-order random walk measures do not consider where the surfer came from and the transition probability only depends on the edges incident to the current node. Based on this assumption and the graph topology, in the next step, the probabilities to visit att.com and delta.com are 2.4×10^{-5} and 3.1×10^{-5}, respectively. That is, the surfer has a higher probability to visit a domain about flight even though she just visited a smart phone domain. However, using the real-life clickstream data (collected from comScore Inc.), given that the previous node is apple.com, the probabilities to visit att.com and delta.com are 8.5×10^{-4} and 3.7×10^{-6}, respectively. That is, the probability to visit a smart phone domain is more than 200 times higher than the probability to visit a flight domain.

As another example, consider the Twitter follower network. Figure 2 shows a subgraph of the real Twitter follower network. The users form two communities, the basketball community on the left and the football community on the right. If the NBA player LeBron James (@KingJames) posts a tweet, it is likely to be propagated among the users in the basketball community instead of the football community. Similarly, if the Pittsburgh Steelers (@steelers) post a tweet, it is likely to be propagated in the football community. The real tweet cascade data support this intuition: Given that the tweet is from @KingJames, the transition probabilities from @espn to @NBA and from @espn to @NFL are 5.6×10^{-2} and 2.1×10^{-4}, respectively. However, using the first-order random walk, the transition probabilities are both 4.5×10^{-3}. That is, the probabilities to go to both communities are similar.

In the examples above, the visiting sequences are recorded in the network flow data. When such data are not available, it is still important to know where the surfer came from. Consider the local community detection problem, whose goal is to find a community nearby a given query node [2,42]. Using the DBLP data,[2] Fig. 3 shows three different research communities involving Prof. Jiawei Han at UIUC. The authors in the left community are senior researchers in the core data mining research areas. The authors in the upper right community have published many works on social media mining. The authors in the lower right community mostly collaborate on information retrieval. Suppose that the random surfer came from the upper right community, e.g., from Prof. T. S. Huang, and is currently at node Prof. Jiawei Han. Intuitively, in the next step, the surfer should walk to a node in the upper right community, since the authors in this community are more similar to Prof. T. S. Huang. However, using the first-order random walk model, the probability of the surfer walking into the left community is higher than that of walking into the upper right community.

To address the limitation of the existing first-order random walk-based proximity measures, in this paper, we investigate the second-order random walk measures, which take the previously visited node into consideration. We systemat-

[1] The entire graph is publicly available at http://webdatacommons.org.

[2] The data are publicly available at http://dblp.uni-trier.de/xml/.

ically study the theoretical foundations of the second-order measures. Specifically, the existing first-order measures are all built on the node-to-node transition probabilities, which can be defined using the adjacency matrix of the graph. To take the previous step into consideration, in the second-order random walk, we need to consider edge-to-edge transition probabilities. We show that such probabilities can be conveniently represented by incidence matrices of the graph [24]. Based on these mathematical tools, we develop simple and elegant matrix representations for the second-order measures including PageRank [34], personalized PageRank [40], SimRank [18], and SimRank* [46], which are among the most widely used proximity measures. A desirable property of the developed second-order measures is that they can degenerate to their original first-order forms when the effect of the previous step is zero. Furthermore, to efficiently compute the second-order measures, we design Monte Carlo algorithms, which effectively simulate paths of the random surfer and estimate proximity values. We formally prove that the estimated proximity value is sharply concentrated around the exact value and converges to the exact value when the sample size is large. We perform extensive experiments to evaluate the effectiveness of the developed second-order measures and the efficiency of the Monte Carlo algorithms using both real and synthetic networks.

The main contributions of this paper are as follows.

1. We provide simple and rigorous theoretical foundation for the second-order measures and develop elegant matrix representations.
2. We devise efficient and accurate Monte Carlo methods to compute the developed second-order measures.
3. We perform comprehensive experimental evaluations using a variety of real-world datasets and applications, which demonstrate the advantage of the second-order measures.

Compared to the previous conference version [41], several significant improvements have been made in the current journal version. In particular, we provide more theoretical details in the matrix representation part. We provide the derivation for the second-order SimRank*. Moreover, in the computing part, we develop single-source algorithms for SimRank and SimRank*. The single-source algorithms are used as the baseline methods for evaluating the Monte Carlo methods. In the computing part, we further provide more technical details for the Monte Carlo methods for computing SimRank and SimRank*. We also add additional experimental results to further evaluate the effectiveness and efficiency of the methods.

2 Related work

Designing effective proximity measures is challenging and has attracted intensive research interests. Shortest path and network flow are two simple proximity measures [8]. Shortest path considers the length of a single path but cannot identify multiple paths between two nodes. On the other hand, network flow uses multiple paths between two nodes but does not consider path lengths. Katz score considers both path lengths and multiple paths [8,20]. It is defined as a weighted sum of lengths of all paths. The more paths between two nodes and the shorter these paths are, the closer the two nodes. Common neighbors and Jaccard's coefficient are two proximity measures defined based on direct neighbors [28]. Common neighbors are defined as the number of neighbors that two nodes have in common. A hub node may have large number of common neighbors with many other nodes. Therefore, Jaccard's coefficient is proposed to reduce this bias and it is defined as the ratio between the number of common neighbors and the number of all neighbors. Jaccard's coefficient only considers the direct neighbors and is zero if two nodes are more than two hops away from each other.

Random walk-based measures have been shown to be effective in many applications. In the first-order random walk, a random surfer explores the graph according to the node-to-node transition probabilities determined by the graph topology. Various random walk-based proximity measures have been developed. Examples include hitting time [32], commute time [12], discounted hitting time [39], truncated hitting time [38], penalized hitting probability [43,47], personalized PageRank [40], RoundTripRank [9], SimRank [18], SimRank* [46], and CoSimRank [37]. Even their definitions are diverse, some are shown to have simple relationships with other measures [39,44]. For example, discounted hitting time and penalized hitting probability give the same ranking results. Even Katz score is defined by using paths instead of random walks, it has a simple relationship with a variant of penalized hitting probability [44]. Among these random walk-based measures, personalized PageRank [40], SimRank [18], and SimRank* [46] have gained significant popularity and been extensively studied.

We briefly introduce each random walk-based measure here. Hitting time is the expected number of hops to hit a node starting at another node [32]. Commute time is the expected number of hops to hit a node and come back to the starting node [12]. Discounted hitting time is similar to hitting time, but at each time point the random surfer can stop with a constant probability. Truncated hitting time only considers paths of length less than a threshold [38]. Similar to discounted hitting time, penalized hitting probability is the probability instead of expected number of hops to hit a node when the random surfer starts from another node and can stop with a constant probability at each time point [43,47]. Discounted hitting time and penalized hitting probability give the same ranking results [44]. Personalized PageRank is the query-biased version of PageRank [40]. In personalized PageRank, at each time point, the surfer has a constant probability to jump to the query node. The personalized PageRank value of a node is defined as the probability of visiting that node. Similar to commute time, RoundTripRank considers round trips between two nodes, but the difference is that in RoundTripRank, the random surfer can stop at each time point with a constant probability [9]. SimRank is based on the intuition that two nodes are similar if their neighbors are similar. The SimRank value between two nodes measures the expected number of steps required before two surfers, one starting from each node, meet at the same node if they walk in lock-step. Here the "lock-step" means that the two random surfers walk exactly one step at each time point; thus, their paces are synchronized. SimRank* is a variant of SimRank, which allows the two surfers not to walk in lock-step. CoSimRank is another variant of SimRank [37]. CoSimRank computes the similarities between the probability distributions of the two random surfers, who start from two nodes, respectively, and walk independently. The matrix form of CoSimRank is very similar to that of SimRank [37]. CoSimRank is more efficient and more accurate than both SimRank or personalized PageRank for similarity measurement [37].

PageRank is a classic method for ranking the nodes [23,24,34] and has been widely used in many real-life applications [14]. PageRank can be explained by using random walk on graphs. At each time point, the random surfer can either walk to out-neighbors by following the transition probabilities or randomly jump to any node in the graph. Recently, two works try to improve PageRank by using the second-order random walk model [15,36]. In [36], the authors study memory-based PageRank, which considers the previously visited node. However, the developed measure does not degenerate to the original PageRank when the effect from the previous node is zero. Along the same line, multilinear PageRank [15] also tries to generalize PageRank to the second order. It approximates the probability of visiting an edge by the product of the probabilities of visiting its two end nodes. This may not be reasonable, e.g., the probability of visiting a nonexistent edge would be nonzero. Both methods are specifically designed for PageRank and do not apply to other measures. The multilinear PageRank is further used in clustering higher-order network structures such as triangles and cycles [3,4].

These random walk-based measures are used by the state-of-the-art methods for many advanced tasks, including ranking [24], querying [19], link prediction [30], local community detection [42], and graph-based semi-supervised learning [49]. In ranking, the task is to rank the nodes based on their importance. PageRank has been shown to be effective and is widely used [14]. Querying aims at finding the nodes that are most close to a given query node. Numerous proximity measures have been proposed to measure the closeness between nodes [8]. The random walk measures, such as personalized PageRank [19] and SimRank [18], have been demonstrated to be effective. In link prediction, the task is to predict new links in the graph [28,30]. Intuitively, two nodes are likely to be connected in the future if they are close to each other in a graph. Since the random walk-based proximity measures are effective in capturing the closeness between nodes, they are used by the state-of-the-art methods for link prediction [28,30]. In local community detection, the goal is to find the community near a given query node [2,42]. Intuitively, a node should be in the target local community if it is close to the query node. The PageRank-Nibble method first computes the personalized PageRank proximity value of each node with regard to the query node and then greedily selects the nodes with large proximity values to form the local community [2]. The query-biased densest connected subgraph method also selects nodes with large proximity values based on random walk measures [42]. In graph-based semi-supervised learning, a graph is constructed to connect similar data objects [49]. The goal is to predict the unknown class labels using the partially labeled data. The label of the nearest neighbor is used as the predicted class label for unlabeled nodes. Since the random walk measures are effective in capturing the closeness between nodes, they are used by the state-of-the-art methods in graph-based semi-supervised learning [48,49].

Although random walk measures are effective, efficiently computing them is a challenging task. Since many random walk measures can be represented as linear systems, the power iteration method can be applied to compute them. In SimRank, however, the power method needs to compute all-pairs proximity matrix and the time and space complexities are very high [18]. A single-source algorithm is developed to compute a single column of the proximity matrix for SimRank [31]. To further accelerate the computation, randomized Monte Carlo methods are proposed [10,11]. These methods can provide a trade-off between efficiency and accuracy.

3 The second-order random walk

In this section, we study the foundation of the second-order random walk. The first-order random walk is based on node-to-node transition probabilities. In the second-order random walk, we need to consider edge-to-edge transition probabilities. The main symbols used in this paper and their definitions are listed in Table 1.

Table 1 Main symbols

Symbols	Definitions
$G(V, E)$	Directed graph G with node set V and edge set E
I_i, O_i	Set of in-/out-neighbor nodes of node i
n, m, σ	Number of nodes; number of edges; $\sigma = \sum_{i \in V} \lvert I_i \rvert \cdot \lvert O_i \rvert$
$\mathbf{B}$	$n \times m$ incidence matrix, $[\mathbf{B}]_{i,u} = 1 : u$ is an out-edge of i
$\mathbf{E}$	$m \times n$ incidence matrix, $[\mathbf{E}]_{u,i} = 1 : u$ is an in-edge of i
$w_{i,j}, w_i$	Weight of edge (i, j); out-degree of i, $w_i = \sum_{j \in O_i} w_{i,j}$
$\mathbf{W}$	$m \times m$ diagonal matrix, $[\mathbf{W}]_{u,u} = w_{i,j}$ if edge $u = (i, j)$
$\mathbf{D}$	$n \times n$ diagonal matrix, $[\mathbf{D}]_{i,i} = w_i$
$p_{i,j}$	Transition probability from node i to j
$p_{i,j,k}$	Transit. prob. from j to k if the surfer came from i
$p_{u,v}$	Transition prob. from edge u to v, $p_{(i,j),(j,k)} = p_{i,j,k}$
$\mathbf{P}$	$n \times n$ node-to-node transition matrix, $[\mathbf{P}]_{i,j} = p_{i,j}$
$\mathbf{H}$	$n \times m$ node-to-edge transition matrix, $[\mathbf{H}]_{i,(i,j)} = p_{i,j}$
$\mathbf{M}$	$m \times m$ edge-to-edge transition matrix, $[\mathbf{M}]_{u,v} = p_{u,v}$
$r_{i,j}, r_i$	Proximity value of node i w.r.t. node j; $r_i = r_{i,q}$
$\mathbf{r}, \mathbf{R}$	$\mathbf{r} : n \times 1$ vector, $\mathbf{r}_i = r_i$; $\mathbf{R} : n \times n$ matrix, $[\mathbf{R}]_{i,j} = r_{i,j}$
$s_u, s_{(i,j)}$	Proximity value of edge u or (i, j) w.r.t. query node q
$s_{u,v}$	Proximity value between edges u and v
$\mathbf{s}, \mathbf{S}$	$\mathbf{s} : m \times 1$ vector, $\mathbf{s}_u = s_u$; $\mathbf{S} : m \times m$ matrix, $[\mathbf{S}]_{u,v} = s_{u,v}$

3.1 The edge-to-edge transition probability

Consider the first-order random walk, where a surfer walks from node i to j with probability $p_{i,j}$. Let $\mathbb{X}_t$ be a random variable representing the node visited by the surfer at time point t. The node-to-node transition probability $p_{i,j}$ can be represented as a conditional probability $\mathbb{P}[\mathbb{X}_t = j | \mathbb{X}_{t-1} = i]$. Let $r_j^t = \mathbb{P}[\mathbb{X}_t = j]$ represent the probability of the surfer visiting node j at time t. We have $r_j^t = \sum_{i \in I_j} p_{i,j} \cdot r_i^{t-1}$, where I_j is the set of in-neighbors of j.

Now consider the second-order random walk. We need to consider where the surfer came from, i.e., the node visited before the current node. We use $p_{i,j,k}$ to represent the transition probability from node j to k given that the previous step was from node i to j, i.e., $p_{i,j,k} = \mathbb{P}[\mathbb{X}_{t+1} = k | \mathbb{X}_{t-1} = i, \mathbb{X}_t = j] = \mathbb{P}[\mathbb{X}_t = j, \mathbb{X}_{t+1} = k | \mathbb{X}_{t-1} = i, \mathbb{X}_t = j]$.

Let $\mathbb{Y}_t = (i, j)$ represent the joint event $(\mathbb{X}_{t-1} = i, \mathbb{X}_t = j)$, i.e., the surfer is at node i at time $(t-1)$ and at node j at time t. Then, the second-order transition probability can be written as $p_{i,j,k} = \mathbb{P}[\mathbb{Y}_{t+1} = (j, k) | \mathbb{Y}_t = (i, j)]$, which can be interpreted as the transition probability between edges: Let $u = (i, j)$ be the edge from node i to j, and $v = (j, k)$ be the edge from node j to k, we can rewrite $p_{i,j,k}$ as $p_{u,v}$.

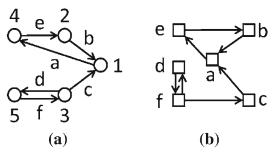

Fig. 4 An example graph and its line graph. **a** A toy graph, **b** the line graph

Probability $p_{i,j,k}$ can be treated as the node-to-node transition probability in the line graph of the original graph. For example, Fig. 4 shows an example graph and its line graph. The second-order transition probability $p_{4,2,1}$ in Fig. 4a is the same as the first-order transition probability $p_{\mathrm{e,b}}$ in Fig. 4b.

Let $s_{(i,j)}^t = \mathbb{P}[\mathbb{Y}_t = (i, j)]$ denote the probability of visiting edge (i, j) between time $(t-1)$ and t. We have

$$s_{(j,k)}^{t+1} = \sum_{i \in I_j} p_{i,j,k} \cdot s_{(i,j)}^t \tag{1}$$

In the following, we introduce incidence matrices, which will be used as building blocks in the second-order random walk measures.

3.2 Incidence matrices as the basic tool

A graph can be represented by its adjacency matrix $\mathbf{A}$, whose element $[\mathbf{A}]_{i,j}$ represents the weight of edge (i, j). Let $\mathbf{D}$

denote the diagonal matrix with $[\mathbf{D}]_{i,i}$ being the out-degree of node i. In the first-order random walk, the node-to-node transition matrix can be represented as $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$. In the second-order random walk, instead of using the adjacency matrix, we will use incidence matrices [24].

The incidence matrices $\mathbf{B}$ and $\mathbf{E}$ represent the out-edges and in-edges of the nodes, respectively. In matrix $\mathbf{B}$, each row represents a node and each column represents an edge. In matrix $\mathbf{E}$, each row represents an edge and each column represents a node. The elements in matrices $\mathbf{B}$ and $\mathbf{E}$ are defined as follows.

$$[\mathbf{B}]_{i,u} = \begin{cases} 1, \text{ if edge } u \text{ is an out-edge of node } i, \\ 0, \text{ otherwise.} \end{cases}$$

$$[\mathbf{E}]_{u,i} = \begin{cases} 1, \text{ if edge } u \text{ is an in-edge of node } i, \\ 0, \text{ otherwise.} \end{cases}$$

For example, the incidence matrices of the graph in Fig. 4a are

$$\mathbf{B} = \begin{array}{c|cccccc} & a & b & c & d & e & f \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 1 & 1 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}, \text{ and } \mathbf{E}^{\mathsf{T}} = \begin{array}{c|cccccc} & a & b & c & d & e & f \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 1 \\ 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}.$$

Note that in the above definitions, the orders of nodes and edges are consistent in $\mathbf{B}$ and $\mathbf{E}$.

The incidence matrices can be conveniently used to reconstruct other commonly used matrices in graph analytics. For example, let $\mathbf{W}$ be a diagonal matrix with $[\mathbf{W}]_{u,u}$ being the weight of edge u, then the adjacency matrix can be represented by the incidence matrices as $\mathbf{A} = \mathbf{BWE}$. The out-degree matrix $\mathbf{D}$ can be represented as $\mathbf{D} = \mathbf{BWB}^{\mathsf{T}}$.

Let $\mathbf{H}$ denote the node-to-edge transition probability matrix, with $[\mathbf{H}]_{i,u}$ representing the probability that the surfer will go through an out-edge u of node i, i.e.,

$$[\mathbf{H}]_{i,u} = \begin{cases} w_u/w_i, & \text{if edge } u \text{ is an out-edge of node } i, \\ 0, & \text{otherwise,} \end{cases}$$

where w_u is the weight of edge u and w_i is the out-degree of node i. $\mathbf{H}$ can be represented using incidence matrices as $\mathbf{H} = \mathbf{D}^{-1}\mathbf{BW}$. The node-to-node transition probability matrix can then be represented as $\mathbf{P} = \mathbf{HE}$.

In the second-order random walk, we use $\mathbf{M}$ to represent the edge-to-edge transition matrix, with element $[\mathbf{M}]_{u,v} = p_{u,v} = p_{i,j,k}$, where $u = (i, j)$ and $v = (j, k)$. If the second-order transition probability degenerates to its first-order form, i.e., if $p_{i,j,k} = p_{j,k}$, we have $\mathbf{M} = \mathbf{EH}$.

Lemma 1 *If $p_{i,j,k} = p_{j,k}$, we have that* $\mathbf{M} = \mathbf{EH}$.

Proof Let $u = (i, j)$ and $v = (j, k)$. There is only one nonzero element in the row vector $[\mathbf{E}]_{u,:}$, i.e., $[\mathbf{E}]_{u,j} = 1$. There is only one nonzero element in the column vector $[\mathbf{H}]_{:,v}$, i.e., $[\mathbf{H}]_{j,v} = p_{j,k}$. Thus, we have $p_{j,k} = [\mathbf{E}]_{u,j} \cdot [\mathbf{H}]_{j,v} = [\mathbf{E}]_{u,:} \cdot [\mathbf{H}]_{:,v}$. We also have that $p_{i,j,k} = [\mathbf{M}]_{u,v}$. Since $p_{i,j,k} = p_{j,k}$, we have that $[\mathbf{M}]_{u,v} = [\mathbf{E}]_{u,:} \cdot [\mathbf{H}]_{:,v}$ for any two edges u and v. Thus, we have that $\mathbf{M} = \mathbf{EH}$. □

3.3 Obtaining edge transition probabilities

In the first-order random walk, the element $p_{i,j}$ in the node-to-node transition matrix $\mathbf{P}$ is calculated as $p_{i,j} = \frac{w_{i,j}}{w_i}$, where $w_{i,j}$ and w_i are the weight of edge (i, j) and out-degree of node i, respectively. Next, we discuss two different ways to obtain the edge-to-edge transition probability.

Utilizing network flow data: In many applications, the information on the node visiting sequences is available. For example, as discussed in Sect. 1, we may know the sequences of web domains browsed by different users, or we may have the tweet cascade information. In this case, we can break each sequence into trigrams, i.e., segments consisting of two consecutive edges [36]. For example, sequence $i \to j \to k \to l$ can be broken into two trigrams, $i \to j \to k$ and $j \to k \to l$.

To obtain the second-order transition probability, recall that $p_{i,j,k}$ is the conditional probability of visiting edge (j, k) given that the previously visited edge is (i, j). Let $\gamma_{i,j,k}$ be the number of trigrams $i \to j \to k$. $p_{i,j,k}$ can be calculated as

$$p_{i,j,k} = \frac{\gamma_{i,j,k}}{\sum_{l \in O_j} \gamma_{i,j,l}}, \quad (2)$$

where O_j is the set of out-neighbor nodes of j. That is, $p_{i,j,k}$ is the proportion of $i \to j \to k$ trigrams in all trigrams with (i, j) being the first edge.

When the network flow data are not available, we can use the following approach to obtain $p_{i,j,k}$.

Autoregressive model: By taking the previous step into consideration, the autoregressive model calculates the second-order transition probability as follows [35]

$$p_{i,j,k} = \frac{p'_{i,j,k}}{\sum_{l \in O_j} p'_{i,j,l}}, \quad (3)$$

where $p'_{i,j,k} = (1 - \alpha)p_{j,k} + \alpha p_{i,k}$. The parameter α $(0 \le \alpha < 1)$ is a constant to control the strength of effect from the previous step. If $\alpha = 0$, the second-order transition probability degenerates to the first-order transition probability, i.e., $p_{i,j,k} = p_{j,k}$.

The edge-to-edge transition matrix $\mathbf{M}$ based on the autoregressive model can be represented using incidence matrices. Let

$$\mathbf{M}' = (1 - \alpha)\mathbf{EH} + \alpha(\mathbf{EB}) \odot (\mathbf{B}^{\mathsf{T}}\mathbf{PE}^{\mathsf{T}}), \quad (4)$$

Table 2 Recursive equations of various measures

	First order	Second order
RW	$\mathbf{r} = \mathbf{P}^{\mathsf{T}}\mathbf{r}$	$\mathbf{s} = \mathbf{M}^{\mathsf{T}}\mathbf{s}$ $\mathbf{r} = \mathbf{E}^{\mathsf{T}}\mathbf{s}$
PR	$\mathbf{r} = c\mathbf{P}^{\mathsf{T}}\mathbf{r} + (1-c)\mathbf{1}/n$	$\mathbf{s} = c\mathbf{M}^{\mathsf{T}}\mathbf{s} + (1-c)\mathbf{H}^{\mathsf{T}}\mathbf{1}/n$ $\mathbf{r} = c\mathbf{E}^{\mathsf{T}}\mathbf{s} + (1-c)\mathbf{1}/n$
PP	$\mathbf{r} = c\mathbf{P}^{\mathsf{T}}\mathbf{r} + (1-c)\mathbf{q}$	$\mathbf{s} = c\mathbf{M}^{\mathsf{T}}\mathbf{s} + (1-c)\mathbf{H}^{\mathsf{T}}\mathbf{q}$ $\mathbf{r} = c\mathbf{E}^{\mathsf{T}}\mathbf{s} + (1-c)\mathbf{q}$
SR	$\mathbf{R} = c\mathbf{P}\mathbf{R}\mathbf{P}^{\mathsf{T}} + (1-c)\mathbf{I}$	$\mathbf{S} = c\mathbf{M}\mathbf{S}\mathbf{M}^{\mathsf{T}} + (1-c)\mathbf{E}\mathbf{E}^{\mathsf{T}}$ $\mathbf{R} = c\mathbf{H}\mathbf{S}\mathbf{H}^{\mathsf{T}} + (1-c)\mathbf{I}$
SS	$\mathbf{R} = \frac{c}{2}(\mathbf{P}\mathbf{R} + \mathbf{R}\mathbf{P}^{\mathsf{T}}) + (1-c)\mathbf{I}$	$\mathbf{S} = \frac{c}{2}(\mathbf{M}\mathbf{S} + \mathbf{S}\mathbf{M}^{\mathsf{T}}) + (1-c)\mathbf{E}\mathbf{E}^{\mathsf{T}}$ $\mathbf{R}' = \frac{c}{2}\mathbf{M}\mathbf{R}' + \frac{c}{2}\mathbf{S}\mathbf{H}^{\mathsf{T}} + (1-c)\mathbf{E}$ $\mathbf{R} = \frac{c}{2}(\mathbf{H}\mathbf{R}' + (\mathbf{R}')^{\mathsf{T}}\mathbf{H}^{\mathsf{T}}) + (1-c)\mathbf{I}$

where $\odot$ denotes the Hadamard (entry-wise) product. Then $\mathbf{M}$ is the row normalized $\mathbf{M}'$ such that $\sum_v p_{u,v} = 1$. If $\alpha = 0$, it degenerates to the first-order form and we have $\mathbf{M} = \mathbf{EH}$.

Note that in addition to the two methods discussed above, other methods, such as calculating the edge similarity based on the line graph [29], can also be applied to calculate the edge-to-edge transition probability. In this paper, we only focus on the two methods discussed here.

3.4 Matrix form

Next we represent the second-order random walk in its matrix form. Let $\mathbf{s}^t$ denote the edge visiting probability vector between time points $(t-1)$ and t, i.e., $\mathbf{s}_u^t = s_{(i,j)}^t$ $(u = (i, j))$. We have

$$\mathbf{s}^{t+1} = \mathbf{M}^{\mathsf{T}}\mathbf{s}^t \; . \tag{5}$$

If $\mathbf{M}$ is primitive, $\mathbf{s}^t$ converges according to the Perron–Frobenius theorem [24]. Let $\mathbf{s} = \lim_{t\to\infty}\mathbf{s}^t$ denote the edge stationary probability. After having $\mathbf{s}$, the node stationary probability is simply the sum of all in-edge stationary probabilities, i.e., $\mathbf{r} = \mathbf{E}^{\mathsf{T}}\mathbf{s}$.

Theorem 1 *If $p_{i,j,k} = p_{j,k}$, the second-order random walk degenerates to the first-order random walk.*

Proof In the first-order random walk, the recursive equation is

$$\mathbf{r} = \mathbf{P}^{\mathsf{T}}\mathbf{r} = \mathbf{E}^{\mathsf{T}}\mathbf{H}^{\mathsf{T}}\mathbf{r} \tag{6}$$

Multiplying $\mathbf{H}^{\mathsf{T}}$ from left to both sides, we have that

$$\mathbf{H}^{\mathsf{T}}\mathbf{r} = \mathbf{H}^{\mathsf{T}}\mathbf{E}^{\mathsf{T}}\mathbf{H}^{\mathsf{T}}\mathbf{r} \tag{7}$$

By Lemma 1, if $p_{i,j,k} = p_{j,k}$, we have that $\mathbf{M} = \mathbf{EH}$. Let $\mathbf{s} = \mathbf{H}^{\mathsf{T}}\mathbf{r}$. We have that

$$\mathbf{s} = \mathbf{H}^{\mathsf{T}}\mathbf{E}^{\mathsf{T}}\mathbf{s} = \mathbf{M}^{\mathsf{T}}\mathbf{s} \tag{8}$$
$$\mathbf{r} = \mathbf{E}^{\mathsf{T}}\mathbf{H}^{\mathsf{T}}\mathbf{r} = \mathbf{E}^{\mathsf{T}}\mathbf{s} \tag{9}$$

Thus, we get the equations for the second-order random walk. That means the solution to the first-order random walk is also a solution to the second-order random walk. Since the solutions are unique, we can complete the proof. □

In the following, we show how to generalize the commonly used proximity measures to their second-order forms. Table 2 summarizes recursive equations of these measures in their first-order and second-order forms. In the table, RW, PR, PP, SR, and SS are shorthand notations for random walk, PageRank, personalized PageRank, SimRank, and SimRank*, respectively.

4 The second-order PageRank

In the first-order PageRank, the surfer has a probability of c to follow the node-to-node transition probabilities and a probability of $(1-c)$ to randomly jump to any node in the graph. Figure 5a illustrates the jumping process.

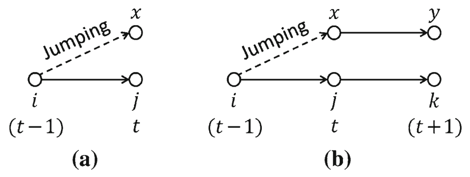

Fig. 5 The jumping process in PageRank. **a** First order and **b** second order

The matrix form of the first-order PageRank is $\mathbf{r} = c\mathbf{P}^{\mathsf{T}}\mathbf{r} + (1-c)\mathbf{1}/n$, where $\mathbf{r}$ is the node visiting probability vector, $\mathbf{P}$ is the node-to-node transition matrix, and $\mathbf{1}$ is a vector of all 1's.

Similarly, in the second-order PageRank, the surfer has a probability of c to follow the edge-to-edge transition probabilities and a probability of $(1-c)$ to randomly jump to any node in the graph. Its matrix form can be written as $\mathbf{s} = c\mathbf{M}^{\mathsf{T}}\mathbf{s} + (1-c)\mathbf{v}$, where $\mathbf{v}$ is the vector corresponding to the jumping process. To determine $\mathbf{v}$, we consider the jumping process in further details.

Figure 5b shows the jumping process in the second-order PageRank. At time point $(t-1)$, starting from node i, with probability c, the surfer first visits node j and then k by following the second-order transition probability $p_{i,j,k}$, and with probability $(1-c)$, the surfer randomly jumps to any node x first and then visits node y. After jumping, the effect of the previous step is lost, thus $p_{i,x,y} = p_{x,y}$, which is the first-order transition probability. Since the sum of probabilities to jump to node x is $(1-c)/n$, the probability of visiting edge (x, y) between time points t and $(t+1)$ is $(1-c)\,p_{x,y}/n$. Thus, we have $\mathbf{v} = \mathbf{H}^{\mathsf{T}}\mathbf{1}/n$, where $\mathbf{H}$ is the node-to-edge transition matrix introduced in Sect. 3.2. Finally, we have that

$$\mathbf{s} = c\mathbf{M}^{\mathsf{T}}\mathbf{s} + (1-c)\mathbf{H}^{\mathsf{T}}\mathbf{1}/n. \tag{10}$$

Theorem 2 *In the second-order PageRank, if the out-degree of every node is nonzero, there is a unique edge stationary distribution* $\mathbf{s}$.

Proof Since the probability distribution vector $\mathbf{s}$ sums to 1, i.e., $\mathbf{1}^{\mathsf{T}}\mathbf{s} = 1$, we have

$$\mathbf{s} = \left(c\mathbf{M}^{\mathsf{T}} + \frac{(1-c)}{n}\mathbf{H}^{\mathsf{T}}\mathbf{1}^{n\times m}\right)\mathbf{s}, \tag{11}$$

where $\mathbf{1}^{n\times m}$ is an $n \times m$ matrix of all 1's. The matrix $\mathbf{T} = c\mathbf{M}^{\mathsf{T}} + \frac{(1-c)}{n}\mathbf{H}^{\mathsf{T}}\mathbf{1}^{n\times m}$ is primitive since $\mathbf{T}$ is irreducible and has positive diagonal elements [24]. Since the out-degree of every node is nonzero, every column of $\mathbf{T}$ sums to 1. Thus, 1 is an eigenvalue of $\mathbf{T}$. By the Perron–Frobenius theorem [24], there is a unique edge stationary distribution and the power method converges. □

The node stationary distribution $\mathbf{r}$ can be obtained from the edge stationary distribution $\mathbf{s}$. The stationary probability of node i equals c times the sum of the edge stationary probabilities on the in-edges of i, plus an additional jumping probability $(1-c)/n$. The formula of $\mathbf{r}$ is given in Table 2.

The developed second-order PageRank degenerates to its original first-order form when the second-order transition probability is the same as the first-order transition probability, i.e., when $p_{i,j,k} = p_{j,k}$.

Theorem 3 *If $p_{i,j,k} = p_{j,k}$, the second-order PageRank degenerates to the first-order PageRank.*

Proof In the first-order PageRank, the recursive equation is

$$\mathbf{r} = c\mathbf{P}^{\mathsf{T}}\mathbf{r} + (1-c)\mathbf{1}/n = c\mathbf{E}^{\mathsf{T}}\mathbf{H}^{\mathsf{T}}\mathbf{r} + (1-c)\mathbf{1}/n \tag{12}$$

Multiplying $\mathbf{H}^{\mathsf{T}}$ from left to both sides, we have that

$$\mathbf{H}^{\mathsf{T}}\mathbf{r} = c\mathbf{H}^{\mathsf{T}}\mathbf{E}^{\mathsf{T}}\mathbf{H}^{\mathsf{T}}\mathbf{r} + (1-c)\mathbf{H}^{\mathsf{T}}\mathbf{1}/n \tag{13}$$

By Lemma 1, if $p_{i,j,k} = p_{j,k}$, we have that $\mathbf{M} = \mathbf{EH}$. Let $\mathbf{s} = \mathbf{H}^{\mathsf{T}}\mathbf{r}$. We have that

$$\mathbf{s} = c\mathbf{H}^{\mathsf{T}}\mathbf{E}^{\mathsf{T}}\mathbf{s} + (1-c)\mathbf{H}^{\mathsf{T}}\mathbf{1}/n = c\mathbf{M}^{\mathsf{T}}\mathbf{s} + (1-c)\mathbf{H}^{\mathsf{T}}\mathbf{1}/n \tag{14}$$

$$\mathbf{r} = c\mathbf{E}^{\mathsf{T}}\mathbf{H}^{\mathsf{T}}\mathbf{r} + (1-c)\mathbf{1}/n = c\mathbf{E}^{\mathsf{T}}\mathbf{s} + (1-c)\mathbf{1}/n \tag{15}$$

Thus, we get the equations for the second-order PageRank. That is, the solution to the first-order PageRank is also a solution to the second-order PageRank. Since the solutions are unique, we can complete the proof. □

Personalized PageRank is the query-biased version of PageRank. In personalized PageRank, instead of jumping to every node uniformly, the surfer jumps to the given query node q. Thus, for personalized PageRank, the jumping vector is $\mathbf{v} = \mathbf{H}^{\mathsf{T}}\mathbf{q}$, where $\mathbf{q}_q = 1$, and $\mathbf{q}_i = 0$ if $i \neq q$.

The developed second-order personalized PageRank also degenerates to its original first-order form when $p_{i,j,k} = p_{j,k}$. Please see "Appendix A" of Electronic Supplementary Material [1] for the proof.

5 The second-order SimRank

In SimRank, the random walk process involves two random surfers [18]. Next, we first give the preliminary of SimRank and discuss its representation based on meeting paths.

5.1 SimRank and meeting paths

The intuition behind SimRank is that two nodes are similar if their in-neighbors are also similar. Let $r_{i,j}$ denote the SimRank proximity value between nodes i and j. SimRank is defined as

$$r_{i,j} = \begin{cases} 1, & \text{if } i = j, \\ \frac{c}{|I_i|\cdot|I_j|}\sum_{k\in I_i}\sum_{l\in I_j} r_{k,l}, & \text{if } i \neq j, \end{cases} \tag{16}$$

where I_i denotes the set of in-neighbors of node i, and $c \in (0, 1)$ is a constant.

The SimRank value $r_{i,j}$ measures the expected number of steps required before two surfers, one starting at node i and the other at node j, meet at the same node if they randomly walk backward, i.e., from a node to one of its in-neighbor nodes, in lock-step [18].

Since walking backward is counterintuitive, in the following, we study SimRank in the reverse graph, which is obtained by reversing the direction of every edge in the original graph. In the reverse graph, the two random surfers walk forward to a meeting node and SimRank can be defined as

$$r_{i,j} = \begin{cases} 1\,, & \text{if } i = j\,, \\ \frac{c}{|O_i|\cdot|O_j|} \sum_{k\in O_i} \sum_{l\in O_j} r_{k,l}\,, & \text{if } i \neq j\,, \end{cases} \tag{17}$$

where O_i denotes the set of out-neighbors of node i in the reverse graph.

In matrix form, the above recursive definition can be approximated by using a linear equation $\mathbf{R} = c\mathbf{PRP}^{\mathsf{T}} + (1-c)\mathbf{I}$, where matrix $\mathbf{R}$ records proximity values for all node pairs with $[\mathbf{R}]_{i,j} = r_{i,j}$ [13,16,21,27,45,46].

SimRank values can also be represented as the weighted sum of probabilities of visiting all meeting paths [46].

Definition 1 (*Meeting path*) A meeting path ϕ of length $\{a, b\}$ between nodes i and j in a graph $G(V, E)$ is a sequence of nodes, denoted as $z_0 \to z_1 \to \cdots \to z_a \leftarrow \cdots \leftarrow z_{b-1} \leftarrow z_b$, such that $i = z_0$, $j = z_b$, $(z_{t-1}, z_t) \in E$ for $t = 1, 2, \ldots, a$, and $(z_t, z_{t-1}) \in E$ for $t = a+1, a+2, \ldots, b$.

A meeting path of length $\{a, b\}$ is symmetric if $b = 2a$, such as the meeting path $4 \to 2 \to 1 \leftarrow 3 \leftarrow 5$ in Fig. 4a.

A meeting path $\phi : i = z_0 \to z_1 \to \cdots \to z_a \leftarrow \cdots \leftarrow z_{b-1} \leftarrow z_b = j$ can be decomposed into two paths $\rho_1 : i = z_0 \to z_1 \to \cdots \to z_a$ and $\rho_2 : j = z_b \to z_{b-1} \to \cdots \to z_a$. In the first-order random walk, starting from i, the probability to visit path ρ_1 is $\mathbb{P}[\rho_1] = \prod_{t=1}^{a} p_{z_{t-1},z_t}$. Similarly, starting from j, the probability to visit path ρ_2 is $\mathbb{P}[\rho_2] = \prod_{t=a+1}^{b} p_{z_t,z_{t-1}}$. The probability for the two surfers to visit ϕ and meet at node z_a is thus $\mathbb{P}[\phi] = \mathbb{P}[\rho_1] \cdot \mathbb{P}[\rho_2]$.

Let $\Phi_{i,j}^{a,b}$ denote the set of all meeting paths of length $\{a, b\}$ between nodes i and j, and $\mathbb{P}[\Phi_{i,j}^{a,b}]$ be the sum of probabilities of visiting the meeting paths in $\Phi_{i,j}^{a,b}$. We have the following Lemma [46].

Lemma 2

$$\mathbb{P}\left[\Phi_{i,j}^{a,b}\right] = \left[\mathbf{P}^a (\mathbf{P}^{\mathsf{T}})^{b-a}\right]_{i,j}$$

Thus, the SimRank value $r_{i,j}$ can be represented as

$$r_{i,j} = (1-c) \sum_{t=0}^{\infty} c^t \mathbb{P}\left[\Phi_{i,j}^{t,2t}\right] = (1-c) \sum_{t=0}^{\infty} c^t \left[\mathbf{P}^t (\mathbf{P}^{\mathsf{T}})^t\right]_{i,j} \tag{18}$$

That is, $r_{i,j}$ is the weighted sum of probabilities of visiting all symmetric meeting paths between nodes i and j. In matrix form, we have $\mathbf{R} = (1-c) \sum_{t=0}^{\infty} c^t \mathbf{P}^t (\mathbf{P}^{\mathsf{T}})^t$.

5.2 Visiting meeting paths in the second order

To develop the second-order SimRank, we need to know the probability of visiting the meeting paths in the second order. Consider the second-order visiting probability of path $\rho_1 : i = z_0 \to z_1 \to \cdots \to z_a$. Starting from node i, in the first step, the surfer follows the first-order transition probability, and then in the subsequent steps, the surfer follows the second-order transition probabilities. Thus in the second-order random walk, starting from i, the probability to visit path ρ_1 is $\mathbb{M}[\rho_1] = p_{z_0,z_1} \prod_{t=1}^{a-1} p_{z_{t-1},z_t,z_{t+1}}$. Similarly, the probability to visit path ρ_2 is $\mathbb{M}[\rho_2] = p_{z_b,z_{b-1}} \prod_{t=a+1}^{b-1} p_{z_{t+1},z_t,z_{t-1}}$. The probability for the two surfers to visit the meeting path ϕ and meet at node z_a is $\mathbb{M}[\phi] = \mathbb{M}[\rho_1] \cdot \mathbb{M}[\rho_2]$.

Let $\mathbb{M}[\Phi_{i,j}^{a,b}] = \sum_{\phi\in\Phi_{i,j}^{a,b}} \mathbb{M}[\phi]$ be the sum of probabilities of visiting the meeting paths in $\Phi_{i,j}^{a,b}$ in the second order. The following lemma shows how to compute $\mathbb{M}[\Phi_{i,j}^{a,b}]$ for different cases.

Lemma 3

$$\mathbb{M}[\Phi_{i,j}^{a,b}] = \begin{cases} \mathbf{I}_{i,j}, & \text{if } 0 = a = b, \\ [\mathbf{HM}^{a-1}\mathbf{E}]_{i,j}, & \text{if } 0 < a = b, \\ [\mathbf{E}^{\mathsf{T}}(\mathbf{M}^{\mathsf{T}})^{b-1}\mathbf{H}^{\mathsf{T}}]_{i,j}, & \text{if } 0 = a < b, \\ [\mathbf{HM}^{a-1}\mathbf{EE}^{\mathsf{T}}(\mathbf{M}^{\mathsf{T}})^{b-a-1}\mathbf{H}^{\mathsf{T}}]_{i,j}, & \text{if } 0 < a < b. \end{cases}$$

Please see Appendix B [1] for the proof. Lemma 2 for the first-order random walk is a special case of Lemma 3 when the second-order transition probability is the same as the first-order transition probability.

Lemma 4 *If* $p_{i,j,k} = p_{j,k}$, *we have that* $\mathbb{M}[\Phi_{i,j}^{a,b}] = \mathbb{P}[\Phi_{i,j}^{a,b}]$.

Please see Appendix B [1] for the proof.

5.3 The second-order SimRank

Replacing $\mathbb{P}[\Phi_{i,j}^{t,2t}]$ by $\mathbb{M}[\Phi_{i,j}^{t,2t}]$ in Eq. (18), we have the second-order SimRank proximity

$$r_{i,j} = (1-c) \sum_{t=0}^{\infty} c^t \mathbb{M}\left[\Phi_{i,j}^{t,2t}\right]. \tag{19}$$

Theorem 4 *The matrix form for the second-order SimRank is*

$$\mathbf{S} = c\mathbf{M}\mathbf{S}\mathbf{M}^\top + (1-c)\mathbf{E}\mathbf{E}^\top \tag{20}$$
$$\mathbf{R} = c\mathbf{H}\mathbf{S}\mathbf{H}^\top + (1-c)\mathbf{I} \tag{21}$$

Please see Appendix B [1] for the proof.

Theorem 5 *There exists a unique solution to the second-order SimRank.*

Proof From the recursive equation of **S**, we have

$$\mathbf{S} = (1-c)\sum_{t=0}^{\infty} c^t \mathbf{M}^t \mathbf{E}\mathbf{E}^\top (\mathbf{M}^\top)^t \tag{22}$$

Let $\mathbf{S}^{(\eta)} = (1-c)\sum_{t=0}^{\eta} c^t \mathbf{M}^t \mathbf{E}\mathbf{E}^\top (\mathbf{M}^\top)^t$. Lemma 5 in Appendix B [1] shows that $\|\mathbf{S} - \mathbf{S}^{(\eta)}\|_{\max} \le c^{\eta+1}$ for any η ($\eta \ge 0$). The convergence of the series follows directly from Lemma 5 and $\lim_{\eta\to\infty} c^{\eta+1} = 0$ $(0 < c < 1)$. Thus, **S** exists.

Next, we prove that **S** is unique. Suppose that **S** and **S**′ are two solutions and we have

$$\mathbf{S} = c\mathbf{M}\mathbf{S}\mathbf{M}^\top + (1-c)\mathbf{E}\mathbf{E}^\top \tag{23}$$
$$\mathbf{S}' = c\mathbf{M}\mathbf{S}'\mathbf{M}^\top + (1-c)\mathbf{E}\mathbf{E}^\top \tag{24}$$

Let $\Delta = \mathbf{S} - \mathbf{S}'$ be the difference. We have $\Delta = c\mathbf{M}\Delta\mathbf{M}^\top$. Let $|\Delta_{u,v}| = \|\Delta\|_{\max}$ for some $u, v \in E$. We have

$$\begin{aligned}
\|\Delta\|_{\max} &= |\Delta_{u,v}| = c \cdot \left|[\mathbf{M}]_{u,:} \cdot \Delta \cdot ([\mathbf{M}]_{v,:})^\top\right| \\
&\le c \sum_{x\in O_u} \sum_{y\in O_v} p_{u,x} \cdot p_{v,y} \cdot \left|[\Delta]_{x,y}\right| \\
&\le c \sum_{x\in O_u} \sum_{y\in O_v} p_{u,x} \cdot p_{v,y} \cdot \|\Delta\|_{\max} = c \cdot \|\Delta\|_{\max},
\end{aligned}$$

where O_u denotes the set of out-neighbor edges of edge u. Since $0 < c < 1$, we have that $\|\Delta\|_{\max} = 0$ and $\mathbf{S} = \mathbf{S}'$. Thus, **S** is unique.

Given that **S** exists and is unique, **R** also exists and is unique. This completes the proof of Theorem 5. □

The second-order SimRank degenerates to its original first-order form when the second-order transition probability is the same as the first-order transition probability. Please see Appendix A [1] for the proof.

6 The second-order SimRank*

SimRank* [46] is a variant of SimRank that considers non-symmetric meeting paths. In this section, we first give the preliminary of SimRank* and then propose the second-order SimRank*. When we develop the second-order SimRank*, we follow an approach similar to that used for developing the second-order SimRank. The equations are summarized in Table 2.

6.1 SimRank*

SimRank* considers asymmetric meeting paths. In matrix form, SimRank* can be represented by the following recursive equation [46]

$$\mathbf{R} = c(\mathbf{P}\mathbf{R} + \mathbf{R}\mathbf{P}^\top)/2 + (1-c)\mathbf{I}, \tag{25}$$

where **R** records proximity values for all node pairs with $[\mathbf{R}]_{i,j} = r_{i,j}$, **P** denotes the transition probability matrix calculated by using the reverse graph, **I** denotes the identity matrix, and $c \in (0, 1)$ is a constant decay factor. By recursively substituting the left- hand side of Eq. (25) into the right-hand side, we obtain the series expansion

$$\mathbf{R} = (1-c)\sum_{t=0}^{\infty} \frac{c^t}{2^t} \sum_{a=0}^{t} \binom{t}{a} \mathbf{P}^a (\mathbf{P}^\top)^{t-a}, \tag{26}$$

where $\binom{t}{a}$ is the binomial coefficient [46]. By Lemma 2, each element $[\mathbf{P}^a(\mathbf{P}^\top)^{t-a}]_{i,j} = \mathbb{P}[\Phi_{i,j}^{a,t}]$ denotes the sum of probabilities of visiting the meeting paths in $\Phi_{i,j}^{a,t}$. We have that

$$r_{i,j} = (1-c)\sum_{t=0}^{\infty} \frac{c^t}{2^t} \sum_{a=0}^{t} \binom{t}{a} \mathbb{P}\left[\Phi_{i,j}^{a,t}\right]. \tag{27}$$

That is, $r_{i,j}$ is the weighted sum of probabilities of visiting all the meeting paths between nodes i and j.

6.2 The second-order SimRank*

To derive the second-order SimRank*, we replace the probability of visiting the meeting paths in the first-order random walk with that in the second order. That is, we replace $\mathbb{P}[\Phi_{i,j}^{a,t}]$ by $\mathbb{M}[\Phi_{i,j}^{a,t}]$ in Eq. (27) and obtain the second-order SimRank* proximity

$$r_{i,j} = (1-c)\sum_{t=0}^{\infty} \frac{c^t}{2^t} \sum_{a=0}^{t} \binom{t}{a} \mathbb{M}[\Phi_{i,j}^{a,t}] \tag{28}$$

From this definition, we can derive the matrix form for the second-order SimRank*.

Theorem 6 *The matrix form for the second-order SimRank* is*

$$\mathbf{S} = c(\mathbf{M}\mathbf{S} + \mathbf{S}\mathbf{M}^\top)/2 + (1-c)\mathbf{E}\mathbf{E}^\top \tag{29}$$
$$\mathbf{R}' = c\mathbf{M}\mathbf{R}'/2 + c\mathbf{S}\mathbf{H}^\top/2 + (1-c)\mathbf{E} \tag{30}$$
$$\mathbf{R} = c(\mathbf{H}\mathbf{R}' + (\mathbf{R}')^\top\mathbf{H}^\top)/2 + (1-c)\mathbf{I} \tag{31}$$

Theorem 7 *The solution to the second-order SimRank* exists and is unique.*

Please see Appendix C [1] for the proofs.

The second-order SimRank* degenerates to its original first-order form when the second-order transition probability is the same as the first-order transition probability. Please see Appendix A [1] for the proof.

7 Computing algorithms

In this section, we discuss how to efficiently compute the developed second-order measures. We first study the power iteration method which utilizes the recursive definitions to compute the exact proximity values. The power methods for SimRank and SimRank* compute all-pairs proximity matrix; thus, it is prohibitive in large graphs. Thus, we develop the single-source algorithm to compute a single column of the proximity matrix. The power method and single-source algorithm both are deterministic and need to iterate over the entire graph; thus, the complexity is high. To speed up the computation, we develop the Monte Carlo methods, which are randomized algorithms and provide a trade-off between accuracy and efficiency. We formally prove that the estimated value (1) converges to the exact proximity value when the sample size is large and (2) is sharply concentrated around the exact value.

7.1 The power iteration method

Given the recursive equations in Table 2, we can apply the power iteration methods to compute the second-order measures. For example, the power method computes the second-order PageRank as follows

$$\mathbf{s}^t = \begin{cases} \mathbf{H}^\mathsf{T}\mathbf{1}/n, & \text{if } t = 0, \\ c\mathbf{M}^\mathsf{T}\mathbf{s}^{t-1} + (1-c)\mathbf{H}^\mathsf{T}\mathbf{1}/n, & \text{if } t > 0. \end{cases} \tag{32}$$

Let $\mathbf{s}$ be the converged edge stationary vector. We can then compute node stationary vector $\mathbf{r} = c\mathbf{E}^\mathsf{T}\mathbf{s} + (1-c)\mathbf{1}/n$.

Time complexity: Let $\sigma = \sum_{i\in V} |I_i|\cdot|O_i|$ denote the number of second-order transition probabilities, i.e., the number of nonzero elements in matrix $\mathbf{M}$. In each iteration, the matrix–vector product $\mathbf{M}^\mathsf{T}\mathbf{s}^{t-1}$ needs $O(\sigma)$ time. Suppose that the power method needs β iteration to converge. It runs in $O(\beta\sigma)$ time for the second-order PageRank. Similarly, the power method for the second-order personalized PageRank also runs in $O(\beta\sigma)$ time.

The power iteration method computes the second-order SimRank as follows

$$\mathbf{S}^{(t)} = \begin{cases} (1-c)\mathbf{E}\mathbf{E}^\mathsf{T}, & \text{if } t = 0, \\ c\mathbf{M}\mathbf{S}^{(t-1)}\mathbf{M}^\mathsf{T} + (1-c)\mathbf{E}\mathbf{E}^\mathsf{T}, & \text{if } t > 0. \end{cases} \tag{33}$$

Algorithm 1 Single-source algorithm for the first-order SR [31]

Input: $G(V, E)$, query node q, decay factor c, maximum length η, transition matrix $\mathbf{P}$
Output: proximity vector $\hat{\mathbf{r}}$

```
1: $\hat{\mathbf{r}} \leftarrow \mathbf{0}$; $\mathbf{r}' \leftarrow (1-c)\mathbf{q}$;
2: for $t \leftarrow 0$ to $\eta$ do
3:     $\hat{\mathbf{r}} \leftarrow \hat{\mathbf{r}} + c^t\mathbf{P}^t\mathbf{r}'$, $\mathbf{r}' \leftarrow \mathbf{P}^\mathsf{T}\mathbf{r}'$;
```

Let $\mathbf{S}$ be the converged edge proximity matrix. We then compute the node proximity matrix as $\mathbf{R} = c\mathbf{H}\mathbf{S}\mathbf{H}^\mathsf{T} + (1-c)\mathbf{I}$.

Time complexity: In each iteration, the matrix–matrix products $\mathbf{M}\mathbf{S}^{(t-1)}\mathbf{M}^\mathsf{T}$ need $O(m\sigma)$ time, where m is the number of edges in the graph and σ is the number of nonzero elements in $\mathbf{M}$. Suppose that the power method needs β iteration to converge. It runs in $O(\beta m\sigma)$ time for the second-order SimRank. Similarly, the power method for the second-order SimRank* also runs in $O(\beta m\sigma)$.

7.2 The single-source algorithm

The power iteration methods for SimRank and SimRank* compute all-pairs proximity matrix $\mathbf{R}$; thus, they are prohibitive in large graphs because of the high time and space complexity. In the next, we develop the single-source algorithms, which compute a single column of matrix $\mathbf{R}$.

Single-source algorithm for the first-order SimRank The original proximity vector can be approximated by the proximity vector

$$\hat{\mathbf{r}} = (1-c)\sum_{t=0}^{\eta} c^t\mathbf{P}^t(\mathbf{P}^\mathsf{T})^t\mathbf{q}, \tag{34}$$

where η denotes the truncated length. In the approximate proximity vector $\hat{\mathbf{r}}$, only the paths of length less than or equal to η are counted. Please see Appendix D.1 [1] for more details. We can easily prove that $0 \le r_i - \hat{r}_i \le c^{\eta+1}$, where r_i and $\hat{r}_i$ denote the exact and approximate proximity values, respectively. That is, the error between the exact proximity value r_i and the approximate proximity value $\hat{r}_i$ is deterministic and small.

Algorithm 1 shows the single-source algorithm for computing SimRank [31]. It exactly calculates the right-hand side of Eq. (34) by maintaining vector $\mathbf{r}' = (1-c)(\mathbf{P}^\mathsf{T})^t\mathbf{q}$. It performs $O(\eta^2)$ matrix–vector products thus runs in $O(\eta^2 m)$.

As an example, if $c = 0.8$ and $\eta = 40$, we have $c^{\eta+1} < 1.07 \times 10^{-4}$. Thus η can be set to 40 and is orders of magnitude smaller than the number of nodes in the graph. Thus, the complexity is significantly reduced compared with the power iteration method.

Algorithm 2 Single-source algorithm for the second-order SR

Input: $G(V, E)$, query node q, decay factor c, maximum length η, transition matrices $\mathbf{M}$ and $\mathbf{H}$, incidence matrix $\mathbf{E}$

Output: proximity vector $\hat{\mathbf{r}}$

```
1: $\hat{\mathbf{r}} \leftarrow (1-c)\mathbf{q}$; $\mathbf{r}' \leftarrow (1-c)\mathbf{H}^{\mathsf{T}}\mathbf{q}$;
2: for $t \leftarrow 1$ to $\eta$ do
3:     $\hat{\mathbf{r}} \leftarrow \hat{\mathbf{r}} + c^t\mathbf{H}\mathbf{M}^{t-1}\mathbf{E}\mathbf{E}^{\mathsf{T}}\mathbf{r}'$, $\mathbf{r}' \leftarrow \mathbf{M}^{\mathsf{T}}\mathbf{r}'$;
```

Algorithm 3 Single-source algorithm for the first-order SS

Input: $G(V, E)$, query node q, decay factor c, maximum length η, transition matrix $\mathbf{P}$

Output: proximity vector $\hat{\mathbf{r}}$

```
1: $\hat{\mathbf{r}} \leftarrow \mathbf{0}$; $\mathbf{r}' \leftarrow (1-c)\mathbf{q}$; $\mathbf{r}'' \leftarrow \mathbf{r}'$;
2: for $a \leftarrow 0$ to $\eta$ do
3:     if $a > 0$ then $\mathbf{r}' \leftarrow \mathbf{P}^{\mathsf{T}}\mathbf{r}'$, $\mathbf{r}'' \leftarrow \mathbf{r}'$;
4:     for $t \leftarrow a$ to $\eta$ do
5:         $\hat{\mathbf{r}} \leftarrow \hat{\mathbf{r}} + \frac{c^t}{2^t}\binom{t}{a}\mathbf{r}''$, $\mathbf{r}'' \leftarrow \mathbf{P}\mathbf{r}''$;
```

Single-source algorithm for the second-order SimRank: The approximate proximity vector can be represented as

$$\hat{\mathbf{r}} = (1-c)\sum_{t=1}^{\eta} c^t\mathbf{H}\mathbf{M}^{t-1}\mathbf{E}\mathbf{E}^{\mathsf{T}}(\mathbf{M}^{\mathsf{T}})^{t-1}\mathbf{H}^{\mathsf{T}}\mathbf{q} + (1-c)\mathbf{q}, \tag{35}$$

where η denotes the truncated length. Please see Appendix D.2 [1] for more details. We can easily prove that $0 \le r_i - \hat{r}_i \le c^{\eta+1}$. That is, the error between the exact proximity value r_i and the approximate proximity value $\hat{r}_i$ is deterministic and small.

Algorithm 2 shows the single-source algorithm for computing the second-order SimRank. It exactly computes the right-hand side of Eq. (35) by maintaining vector $\mathbf{r}' = (1-c)(\mathbf{M}^{\mathsf{T}})^{t-1}\mathbf{H}^{\mathsf{T}}\mathbf{q}$. It performs $O(\eta^2)$ matrix–vector products thus runs in $O(\eta^2\sigma)$, where $\sigma = \sum_{i\in V}|I_i|\cdot|O_i|$ denotes the number of second-order transition probabilities, i.e., the number of nonzero elements in matrix $\mathbf{M}$.

*Single-source algorithm for the first-order SimRank**: The approximate proximity vector can be represented as

$$\hat{\mathbf{r}} = (1-c)\sum_{a=0}^{\eta}\sum_{t=a}^{\eta}\frac{c^t}{2^t}\binom{t}{a}\mathbf{P}^{t-a}(\mathbf{P}^{\mathsf{T}})^a\mathbf{q}, \tag{36}$$

where η denotes the truncated length. Please see Appendix D.3 [1] for more details. We can prove that $0 \le r_i - \hat{r}_i \le c^{\eta+1}$. That is, the approximate proximity value $\hat{r}_i$ is very close to the exact proximity value r_i.

Algorithm 3 shows the single-source algorithm for computing SimRank*. It exactly calculates the right-hand side of Eq. (36) by maintaining vectors $\mathbf{r}' = (1-c)(\mathbf{P}^{\mathsf{T}})^a\mathbf{q}$ and $\mathbf{r}'' = \mathbf{P}^{t-a}\mathbf{r}'$. Algorithm 3 performs $O(\eta^2)$ matrix–vector products thus runs in $O(\eta^2 m)$.

Algorithm 4 Single-source algorithm for the second-order SS

Input: $G(V, E)$, query node q, decay factor c, maximum length η, transition matrices $\mathbf{M}$ and $\mathbf{H}$, incidence matrix $\mathbf{E}$

Output: proximity vector $\hat{\mathbf{r}}$

```
1: $\hat{\mathbf{r}} \leftarrow \mathbf{0}$; $\mathbf{r}' \leftarrow (1-c)\mathbf{q}$;
2: for $a \leftarrow 0$ to $\eta$ do
3:     if $a = 1$ then $\mathbf{r}' \leftarrow \mathbf{H}^{\mathsf{T}}\mathbf{r}'$;
4:     if $a > 1$ then $\mathbf{r}' \leftarrow \mathbf{M}^{\mathsf{T}}\mathbf{r}'$;
5:     for $t \leftarrow a$ to $\eta$ do
6:         if $t = a = 0$ then $\hat{\mathbf{r}} \leftarrow \hat{\mathbf{r}} + \mathbf{r}'$, $\mathbf{r}'' \leftarrow \mathbf{E}\mathbf{r}'$;
7:         if $t = a > 0$ then $\hat{\mathbf{r}} \leftarrow \hat{\mathbf{r}} + \frac{c^t}{2^t}\mathbf{E}^{\mathsf{T}}\mathbf{r}'$, $\mathbf{r}'' \leftarrow \mathbf{E}\mathbf{E}^{\mathsf{T}}\mathbf{r}'$;
8:         if $t >$          then $\hat{\mathbf{r}} \leftarrow \hat{\mathbf{r}} + \frac{c^t}{2^t}\binom{t}{a}\mathbf{H}\mathbf{r}''$, $\mathbf{r}'' \leftarrow \mathbf{M}\mathbf{r}''$;
```

*Single-source algorithm for the second-order SimRank**: The approximate proximity vector can be represented as

$$\begin{aligned}\hat{\mathbf{r}} &= (1-c)\mathbf{q} + (1-c)\sum_{t=1}^{\eta}\frac{c^t}{2^t}\mathbf{H}\mathbf{M}^{t-1}\mathbf{E}\mathbf{q} \\ &\quad + (1-c)\sum_{t=1}^{\eta}\frac{c^t}{2^t}\mathbf{E}^{\mathsf{T}}(\mathbf{M}^{\mathsf{T}})^{t-1}\mathbf{H}^{\mathsf{T}}\mathbf{q} \\ &\quad + (1-c)\sum_{a=1}^{\eta-1}\sum_{t=a+1}^{\eta}\frac{c^t}{2^t}\binom{t}{a}\mathbf{H}\mathbf{M}^{t-a-1}\mathbf{E}\mathbf{E}^{\mathsf{T}}(\mathbf{M}^{\mathsf{T}})^{a-1}\mathbf{H}^{\mathsf{T}}\mathbf{q},\end{aligned} \tag{37}$$

where η denotes the truncated length. Please see Appendix D.4 [1] for more details. We can prove that $0 \le r_i - \hat{r}_i \le c^{\eta+1}$.

Algorithm 4 shows the single-source algorithm for computing the second-order SimRank*. It exactly computes the right-hand side of Eq. (37) by maintaining vectors $\mathbf{r}'$ and $\mathbf{r}''$ shown as follows. Algorithm 4 performs $O(\eta^2)$ matrix–

if $a = 0$	$\mathbf{r}' = (1-c)\mathbf{q}$	$\mathbf{r}'' = \mathbf{M}^{t-1}\mathbf{E}\mathbf{r}'$
if $a > 0$	$\mathbf{r}' = (1-c)(\mathbf{M}^{\mathsf{T}})^{a-1}\mathbf{H}^{\mathsf{T}}\mathbf{q}$	$\mathbf{r}'' = \mathbf{M}^{t-a-1}\mathbf{E}\mathbf{E}^{\mathsf{T}}\mathbf{r}'$

vector products thus runs in $O(\eta^2\sigma)$, where σ denotes the number of nonzero elements in matrix $\mathbf{M}$.

7.3 The Monte Carlo method

Monte Carlo (MC) methods have been recently studied to compute the first-order personalized PageRank [11] and SimRank [10]. Next, we develop MC methods to compute the second-order personalized PageRank, SimRank, and SimRank* and provide the theoretical analysis for the developed methods.

Algorithm 5 The MC algorithm for the first-order PP [11]

Input: $G(V, E)$, query node q, decay factor c, sample size π
Output: estimated proximity vector $\tilde{\mathbf{r}}$

```
1: for each node $i \in V$ do $\tilde{r}_i \leftarrow 0$;                    // initialization
2: repeat $\pi$ times                                                        // sample $\pi$ paths
3:     $a \leftarrow$generate a random number following the geometric
       distribution $\mathbb{P}[\mathbb{A} = a] = (1 - c) \cdot c^a$;
4:     $z_0 \leftarrow q$; bSuccess $\leftarrow$ true;
5:     for $t \leftarrow 1$ to $a$ do
6:         if $|O_{z_{t-1}}| = 0$ then bSuccess $\leftarrow$ false, break;
7:         $z_t \leftarrow$randomly pick a node from $O_{z_{t-1}}$ according to the
           first-order transition probability;
8:     if bSuccess = true then $\tilde{r}_{z_a} \leftarrow \tilde{r}_{z_a} + 1$;
9: for each node $i \in V$ do $\tilde{r}_i \leftarrow \tilde{r}_i / \pi$;      // normalization
```

Algorithm 6 The MC algorithm for the second-order PP

Input: $G(V, E)$, query node q, decay factor c, sample size π
Output: estimated proximity vector $\tilde{\mathbf{r}}$

```
1: for each node $i \in V$ do $\tilde{r}_i \leftarrow 0$;                    // initialization
2: repeat $\pi$ times                                                        // sample $\pi$ paths
3:     $a \leftarrow$generate a random number following the geometric
       distribution $\mathbb{P}[\mathbb{A} = a] = (1 - c) \cdot c^a$;
4:     $z_0 \leftarrow q$; bSuccess $\leftarrow$ true;
5:     for $t \leftarrow 1$ to $a$ do
6:         if $|O_{z_{t-1}}| = 0$ then bSuccess $\leftarrow$ false, break;
7:         if $t = 1$ then $z_t \leftarrow$randomly pick a node from $O_{z_{t-1}}$
           according to the first-order transition probability;
8:         else $z_t \leftarrow$ randomly pick a node from $O_{z_{t-1}}$ according
                to the second-order transition probability;
9:     if bSuccess = true then $\tilde{r}_{z_a} \leftarrow \tilde{r}_{z_a} + 1$;
10: for each node $i \in V$ do $\tilde{r}_i \leftarrow \tilde{r}_i / \pi$;     // normalization
```

7.3.1 Computing personalized PageRank

To illustrate the basic idea, we begin with the MC algorithm for the first-order personalized PageRank [11], which is shown in Algorithm 5. It is based on the following series expansion of personalized PageRank

$$r_i = (1 - c) \sum_{t=0}^{\infty} c^t \mathbb{P}\left[\Phi_{i,q}^{0,t}\right] = (1 - c) \sum_{t=0}^{\infty} c^t \mathbb{P}\left[\Phi_{q,i}^{t,t}\right] \tag{38}$$

That is, the proximity r_i can be represented as the weighted sum of probabilities of visiting all paths from node q to i. The longer the path length t, the smaller the weight $(1 - c) \cdot c^t$. Based on this interpretation, in line 3 of Algorithm 5, the MC method determines the path length a based on the geometric distribution. Then, starting from the query node q, the algorithm simulates a path of length a in lines 4–7. When simulating a path, at each time point t $(1 \le t \le a)$, the algorithm randomly picks an out-neighbor of the previous node z_{t-1} to visit according to the first-order transition probability. The algorithm stops when the simulated path reaches length a or there is no out-neighbor to pick. The algorithm repeats this process π times, where π is the number of paths to be simulated. The proximity value of node i is estimated as the fraction of the π paths that end at i.

We can extend this MC algorithm to compute the second-order personalized PageRank as shown in Algorithm 6. It is based on the series expansion

$$r_i = (1-c) \sum_{t=0}^{\infty} c^t \mathbb{M}\left[\Phi_{i,q}^{0,t}\right] = (1-c) \sum_{t=0}^{\infty} c^t \mathbb{M}\left[\Phi_{q,i}^{t,t}\right] \tag{39}$$

That is, the proximity r_i can be represented as the weighted sum of probabilities of visiting all paths from node q to i in the second-order random walk. The difference between Algorithms 5 and 6 is how to sample a path. In lines 5–7 of Algorithm 5, at each step, the algorithm picks an out-neighbor with the first-order transition probability. In lines 5–8 of Algorithm 6, only in the first step, i.e., when $t = 1$, the algorithm picks an out-neighbor with the first-order transition probability. Then, the algorithm picks out-neighbors with the second-order transition probabilities when simulating subsequent steps.

Theorem 8 shows that when the sample size is large, the estimated proximity $\tilde{r}_i$ converges to the exact proximity r_i. Theorem 9 shows that the error is bounded by a term that is exponentially small in terms of the sample size.

Theorem 8 *The estimated proximity $\tilde{r}_i$ converges to the exact proximity r_i when $\pi \to \infty$.*

Proof In Algorithm 6, if we successfully sample a path ending at node i, we will increase $\tilde{r}_i$ by 1; otherwise, $\tilde{r}_i$ is unchanged. Let $\mathbb{S}_i^{(d)}$ be a Bernoulli random variable denoting the incremental value of $\tilde{r}_i$ at the dth iteration (lines 3–9). Random variables $\mathbb{S}_i^{(1)}, \mathbb{S}_i^{(2)}, \ldots, \mathbb{S}_i^{(\pi)}$ are independent and identically distributed. Let $\mathbb{S}_i$ be a Bernoulli random variable following the same distribution as $\mathbb{S}_i^{(d)}$'s. Lemma 9 in Appendix E.1 [1] shows that the expected value of $\mathbb{S}_i$ equals the exact proximity r_i, i.e., $\mathbb{E}[\mathbb{S}_i] = r_i$.

Let $\overline{\mathbb{S}}_i = \frac{1}{\pi} \sum_{d=1}^{\pi} \mathbb{S}_i^{(d)}$ be the sample average, which represents the estimated proximity $\tilde{r}_i$. By the law of large numbers, if the sample size $\pi \to \infty$, $\overline{\mathbb{S}}_i$ converges to the expected value $\mathbb{E}[\mathbb{S}_i] = r_i$. This completes the proof of Theorem 8. □

Theorem 9 *For any $\epsilon > 0$, we have that*

$$\mathbb{P}[|\tilde{r}_i - r_i| \ge \epsilon] \le 2 \cdot \exp(-2\pi\epsilon^2) \tag{40}$$

Proof Following the notations defined in the proof of Theorem 8, random variables $\mathbb{S}_i^{(1)}, \mathbb{S}_i^{(2)}, \ldots, \mathbb{S}_i^{(\pi)}$ are independent and bounded by interval $[0, 1]$. By Hoeffding's inequality [17], we can prove this theorem. □

Time complexity: Generating a random number costs $O(1)$ time. Since the path length a follows the geometric distribution, the average length is $(1 - c) \sum_{a=0}^{\infty} a c^a = c/(1 - c)$.

Algorithm 7 Basic MC algorithm for the first-order SR [10]

Input: $G(V, E)$, query node q, decay factor c, sample size π, maximum length η
Output: estimated proximity vector $\tilde{\mathbf{r}}$

```
1: for each node $i \in V$ do                    // process each node individually
2:     $\tilde{r}_i \leftarrow 0$;                              // initialization
3:     repeat $\pi$ times                            // sample $\pi$ pairs of paths
4:         sample a path $q = z_0 \rightarrow z_1 \rightarrow \cdots \rightarrow z_\eta$ starting from $q$;
5:         sample a path $i = z'_0 \rightarrow z'_1 \rightarrow \cdots \rightarrow z'_\eta$ starting from $i$;
                                     // find the common node with the smallest offset
6:         for $t = 0$ to $\eta$ do   if $z_t = z'_t$ then $\tilde{r}_i \leftarrow \tilde{r}_i + c^t$, break;
7:     $\tilde{r}_i \leftarrow \tilde{r}_i / \pi$;                          // normalization
```

When sampling a path, at each step, the algorithm randomly picks an out-neighbor, which costs $O(\xi)$ on average, where $\xi = \frac{1}{n}\sum_{i \in V} |O_i|$ denotes the average out-degree. Thus, on average, sampling a path costs $O(\xi c/(1-c))$ time. Sampling π paths costs $O(\pi \xi c/(1-c))$ time. Initialization and normalization cost $O(n)$ time. In total, Algorithm 6 runs in $O(\pi \xi c/(1-c) + n)$ time.

The MC algorithm developed here is readily applicable to compute the second-order PageRank. Let $\mathrm{PR}(i)$ denote the PageRank value of node i, and $\mathrm{PP}_j(i)$ denote the personalized PageRank proximity of node i when j is the query.

Theorem 10

$$\mathrm{PR}(i) = \frac{1}{n}\sum_{j \in V} \mathrm{PP}_j(i)$$

Proof The proof is similar to that of the linearity theorem [19]. We omit the proof here. □

Based on this theorem, we can use Algorithm 6 to compute the personalized PageRank proximity vector for each node. The average of all vectors is the estimated PageRank vector.

7.3.2 Computing the first-order SimRank

The basic MC algorithm for computing the first-order SimRank is proposed in [10], which is shown in Algorithm 7. It is based on the original interpretation of SimRank [18], i.e., the proximity r_i measures the expected number of steps required before two surfers, one starting at the query node q and the other at node i, meet at the same node if they randomly walk on the reverse graph in lock-step.

As shown in Algorithm 7, the algorithm proposed in [10] directly simulates the meeting paths of the two surfers. For each node i, it simulates two paths of length η, one starting from node q and one from i. It then scans these two paths to determine whether there is a common node. The fraction of the sampled paths that do have a common node is used as the estimated proximity value for node i.

Algorithm 7 estimates the proximity of each node i individually and samples a fixed number of paths for each node.

Algorithm 8 The MC algorithm for the first-order SimRank

Input: $G(V, E)$, query node q, decay factor c, sample size π, maximum length η, matrix $\mathbf{X}$
Output: estimated proximity vector $\tilde{\mathbf{r}}$

```
1: for each node $i \in V$ do $\tilde{r}_i \leftarrow 0$;            // initialization
2: repeat $\pi$ times                                // sample $\pi$ meeting paths
3:     $a \leftarrow$generate a random number following the geometric
       distribution $\mathbb{P}[\mathbb{A} = a] = (1-c) \cdot c^a$;
4:     if $a > \eta$ then continue;
5:     [bSuccess, $z_{2a}$, $\delta$] $\leftarrow$ SampleOneMeetingPath($q, a, \mathbf{X}$);
6:     if bSuccess = true then $\tilde{r}_{z_{2a}} \leftarrow \tilde{r}_{z_{2a}} + \delta$;
7: for each node $i \in V$ do $\tilde{r}_i \leftarrow \tilde{r}_i / \pi$;       // normalization
```

This method usually needs to simulate a large number of meeting paths to achieve an accurate estimation since not all simulated paths may have common nodes. The paths that do not have common nodes can only be used in the denominator in the estimated proximity value.

Next we propose a new sampling strategy to compute the SimRank values. Our sampling method estimates the proximity values for all the nodes at the same time. Every simulated path is guaranteed to contribute to the numerator of some node and to the denominators of all nodes. Experimental results show that compared to the previous method, our sampling method needs several orders of magnitude less simulated paths to achieve the same accuracy.

Algorithm 8 shows the overall procedure of the proposed algorithm, which is based on the series expansion

$$r_i = (1-c)\sum_{t=0}^{\infty} c^t \mathbb{P}\left[\Phi_{i,q}^{t,2t}\right] = (1-c)\sum_{t=0}^{\infty} c^t \mathbb{P}\left[\Phi_{q,i}^{t,2t}\right] \tag{41}$$

That is, the proximity r_i can be represented as the weighted sum of probabilities of visiting all meeting paths of length $\{t, 2t\}$ between nodes q and i.

Instead of simulating meeting paths starting from both nodes q and i, we only simulate paths starting from q. Algorithm 9 shows the procedure to sample a meeting path. For a meeting path $\phi : q = z_0 \rightarrow z_1 \rightarrow \cdots \rightarrow z_a \leftarrow \cdots \leftarrow z_{2a-1} \leftarrow z_{2a} = i$, when simulating the first half of the path, we simply follow the first-order transition probability. Since the second half of the meeting path is in reverse order, we need to pick in-neighbors of the visited nodes. To do that, we need to know the probability of visiting in-neighbors. We can use Bayes' theorem to calculate these probabilities.

Let $\mathbb{X}_t$ be a random variable representing the node visited by the surfer at time t. Suppose that node j is an in-neighbor of k, i.e., $j \in I_k$. We have

$$\begin{aligned}\mathbb{P}[\mathbb{X}_{t-1} = j | \mathbb{X}_t = k] &= \frac{\mathbb{P}[\mathbb{X}_{t-1} = j]}{\mathbb{P}[\mathbb{X}_t = k]} \cdot \mathbb{P}[\mathbb{X}_t = k | \mathbb{X}_{t-1} = j] \\ &= \frac{\mathbb{P}[\mathbb{X}_{t-1} = j]}{\mathbb{P}[\mathbb{X}_t = k]} \cdot p_{j,k} \end{aligned} \tag{42}$$

Algorithm 9 [bSuccess, z_{2a}, δ] $\leftarrow$ SampleOneMeetingPath($q, a, \mathbf{X}$)

```
1: $z_0 \leftarrow q$; bSuccess $\leftarrow$ true;                     // start from the query node
2: for $t \leftarrow 1$ to $a$ do                                        // sample the first half
3:     if $|O_{z_{t-1}}| = 0$ then bSuccess $\leftarrow$ false, return;
4:     $z_t \leftarrow$randomly pick a node from $O_{z_{t-1}}$ according to the first-order transition probability;
5: for $t \leftarrow (a+1)$ to $2a$ do                                   // sample the second half
6:     if $|I_{z_{t-1}}| = 0$ or $[\mathbf{X}]_{z_{t-1},2a-t+1} = 0$ then bSuccess $\leftarrow$ false,return;
7:     $z_t \leftarrow$randomly pick a node from $I_{z_{t-1}}$ according to the probability $p_{z_t,z_{t-1}} \cdot [\mathbf{X}]_{z_t,2a-t}/[\mathbf{X}]_{z_{t-1},2a-t+1}$;
8: $\delta \leftarrow [\mathbf{X}]_{z_a,a}/[\mathbf{X}]_{z_{2a},0}$;
```

Algorithm 10 ComputeNodeVisitingProbabilities()

Input: $G(V, E)$, transition matrix $\mathbf{P}$, maximum length η
Output: $n \times (\eta + 1)$ matrix $\mathbf{X}$

```
1: $[\mathbf{X}]_{:,0} \leftarrow \mathbf{1}/n$;      // begin from the uniform probability distribution
2: for $t \leftarrow 1$ to $\eta$ do $[\mathbf{X}]_{:,t} \leftarrow \mathbf{P}^{\mathsf{T}}[\mathbf{X}]_{:,t-1}$;
```

Thus to calculate the probability of visiting in-neighbors, we only need the prior probability of visiting each node. Algorithm 10 computes these probabilities and stores them in an $n \times (\eta + 1)$ matrix $\mathbf{X}$ in the preprocessing stage, with $[\mathbf{X}]_{j,t} = \mathbb{P}[\mathbb{X}_t = j]$, where η is the maximum length of the simulated paths.

Let $\hat{r}_i = (1-c)\sum_{t=0}^{\eta} c^t \mathbb{P}[\Phi_{q,i}^{t,2t}]$ be the SimRank value we try to estimate. Next, we show that our sampling strategy gives accurate estimation of $\hat{r}_i$.

Theorem 11 *The estimated proximity $\tilde{r}_i$ converges to $\hat{r}_i$ when $\pi \to \infty$.*

Proof In Algorithm 8, if we successfully sample a path ending at node i, we will increase $\tilde{r}_i$ by δ; otherwise, $\tilde{r}_i$ is unchanged. For different sampled paths, the corresponding value δ may be different. Let $\mathbb{R}_i^{(d)}$ be a random variable denoting the incremental value of $\tilde{r}_i$ at the dth iteration (lines 3–6). Random variables $\mathbb{R}_i^{(1)}, \mathbb{R}_i^{(2)}, \ldots, \mathbb{R}_i^{(\pi)}$ are independent and identically distributed. Let $\mathbb{R}_i$ be a random variable following the same distribution as $\mathbb{R}_i^{(d)}$'s. Lemma 9 in Appendix E.2 [1] shows that the expected value of $\mathbb{R}_i$ equals the truncated proximity $\hat{r}_i$, i.e., $\mathbb{E}[\mathbb{R}_i] = \hat{r}_i$.

Let $\overline{\mathbb{R}}_i = \frac{1}{\pi}\sum_{d=1}^{\pi}\mathbb{R}_i^{(d)}$ be the sample average, which represents the estimated proximity $\tilde{r}_i$. By the law of large numbers, if the sample size $\pi \to \infty$, $\overline{\mathbb{R}}_i$ converges to the expected value $\mathbb{E}[\mathbb{R}_i] = \hat{r}_i$. This completes the proof of Theorem 11. □

Theorem 12 *For any $\epsilon > 0$, we have that*

$$\mathbb{P}[\tilde{r}_i - \hat{r}_i \le -\epsilon] \le \exp\left(\frac{-\pi\epsilon^2}{2nr_i}\right) \tag{43}$$

$$\mathbb{P}[\tilde{r}_i - \hat{r}_i \ge \epsilon] \le \exp\left(\frac{-\pi\epsilon^2}{2nr_i + 2n\epsilon/3}\right) \tag{44}$$

Algorithm 11 The MC algorithm for the second-order SR

Input: $G(V, E)$, query node q, decay factor c, sample size π, maximum length η, matrix $\mathbf{Y}$
Output: estimated proximity vector $\tilde{\mathbf{r}}$

```
1: for each node $i \in V$ do $\tilde{r}_i \leftarrow 0$;          // initialization
2: repeat $\pi$ times                                                  // sample $\pi$ meeting paths
3:     $a \leftarrow$generate a random number following the geometric distribution $\mathbb{P}[\mathbb{A} = a] = (1-c)\cdot c^a$;
4:     if $a > \eta$ then continue;
5:     [bSuccess, $z_{2a}$, $\delta$] $\leftarrow$ SampleOneMeetingPath2($q, a, \mathbf{Y}$);
6:     if bSuccess = true then $\tilde{r}_{z_{2a}} \leftarrow \tilde{r}_{z_{2a}} + \delta$;
7: for each node $i \in V$ do $\tilde{r}_i \leftarrow \tilde{r}_i/\pi$;      // normalization
```

Proof Following the notations defined in the proof of Theorem 11, random variables $\mathbb{R}_i^{(1)}, \mathbb{R}_i^{(2)}, \ldots, \mathbb{R}_i^{(\pi)}$ are independent and bounded by interval $[0, \delta] \subseteq [0, n]$. Lemma 9 in Appendix E.2 [1] shows that the expected value of $\mathbb{R}_i^2$ is bounded from above by nr_i, i.e., $\mathbb{E}[\mathbb{R}_i^2] \le nr_i$. Thus, we have $\sum_{d=1}^{\pi}\mathbb{E}[(\mathbb{R}_i^{(d)})^2] \le \pi nr_i$. By Theorem 22 in Appendix E.2 [1], we can prove this theorem. □

Time complexity: Since the path length a follows the geometric distribution, the average length is $(1-c)\sum_{a=0}^{\infty} 2ac^a = 2c/(1-c)$. When sampling a path in Algorithm 9, at each step, the algorithm randomly picks an out-neighbor or in-neighbor, which costs $O(\psi)$ time on average, where $\psi = \frac{1}{2n}\sum_{i\in V}\left(|I_i| + |O_i|\right)$ denotes the average degree. Thus, on average, sampling a path costs $O(\psi c/(1-c))$. Sampling π paths costs $O(\pi\psi c/(1-c))$. Initialization and normalization cost $O(n)$. In total, Algorithm 8 runs in $O(\pi\psi c/(1-c)+n)$ time. Algorithm 10 runs in $O(m\eta)$ time.

7.3.3 Computing the second-order SimRank

The developed sampling strategy for the first-order SimRank can be easily extended to the second-order SimRank. The difference is that in the second-order SimRank, we need to follow the second-order transition probability when sampling meeting paths. Algorithm 11 shows the overall procedure of the proposed MC algorithm for computing the second-order SimRank.

Algorithm 11 is based on the series expansion

$$r_i = (1-c)\sum_{t=0}^{\infty} c^t \mathbb{M}\left[\Phi_{i,q}^{t,2t}\right] = (1-c)\sum_{t=0}^{\infty} c^t \mathbb{M}\left[\Phi_{q,i}^{t,2t}\right] \tag{45}$$

That is, the proximity r_i can be represented as the weighted sum of probabilities of visiting all meeting paths of length $\{t, 2t\}$ between nodes q and i.

Similar to the sampling strategy used in Algorithm 8, we only simulate paths starting from q. Algorithm 12 shows the procedure to sample a meeting path. For a meeting path

Algorithm 12 [bSuccess, z_{2a}, δ] $\leftarrow$ SampleOneMeetingPath2($q, a, \mathbf{Y}$)

```
1: $z_0 \leftarrow q$; bSuccess $\leftarrow$ true;   // start from the query node
2: for $t \leftarrow 1$ to $a$ do   // sample the first half
3:     if $|O_{z_{t-1}}| = 0$ then bSuccess $\leftarrow$ false, return;
4:     if $t = 1$ then $z_t \leftarrow$ randomly pick a node from $O_{z_{t-1}}$ according
       to the first-order transition probability $p_{z_{t-1},z_t}$;
5:     else $z_t \leftarrow$ randomly pick a node from $O_{z_{t-1}}$ according to the
       second-order transition probability $p_{z_{t-2},z_{t-1},z_t}$;
6: for $t \leftarrow (a+1)$ to $2a$ do   // sample the second half
7:     if $|I_{z_{t-1}}| = 0$ or $[\mathbf{Y}]_{(z_{t-1},z_{t-2}),2a-t+1} = 0$ then
8:         bSuccess $\leftarrow$ false, return;
9:     if $t = (a+1)$ then $z_t \leftarrow$ pick a node from $I_{z_{t-1}}$ uniformly
       at random;
10:    else $z_t \leftarrow$ randomly pick a node from $I_{z_{t-1}}$ according to
       the probability $p_{z_t,z_{t-1},z_{t-2}} \cdot \frac{[\mathbf{Y}]_{(z_t,z_{t-1}),2a-t}}{[\mathbf{Y}]_{(z_{t-1},z_{t-2}),2a-t+1}}$;
11: $\delta \leftarrow |I_{z_a}| \cdot p_{z_{2a},z_{2a-1}} \cdot [\mathbf{Y}]_{(z_{a+1},z_a),a-1} / [\mathbf{Y}]_{(z_{2a},z_{2a-1}),0}$;
```

Algorithm 13 ComputeEdgeVisitingProbabilities()

```
Input: $G(V, E)$, transition matrix $\mathbf{M}$, maximum length $\eta$
Output: $m \times (\eta + 1)$ matrix $\mathbf{Y}$
1: $[\mathbf{Y}]_{:,0} \leftarrow \mathbf{1}/m$;   // begin from the uniform probability distribution
2: for $t \leftarrow 1$ to $\eta$ do $[\mathbf{Y}]_{:,t} \leftarrow \mathbf{M}^{\mathrm{T}}[\mathbf{Y}]_{:,t-1}$;
```

$\phi : q = z_0 \to z_1 \to \cdots \to z_a \leftarrow \cdots \leftarrow z_{2a-1} \leftarrow z_{2a} = i$, when simulating the first half of the path, we simply follow the first-order transition probability in the first step, and then, the second-order transition probabilities in the following steps. Since the second half of the meeting path is in reverse order, we need to know the probability of picking in-neighbors of the visited nodes. In the first step, we pick a node z_{a+1} from I_{z_a} uniformly at random. In the following steps, we can use Bayes' theorem to calculate these probabilities.

Let $\mathbb{X}_t$ be a random variable representing the node visited by the surfer at time t. Let $\mathbb{Y}_t = (i, j)$ represent the joint event $(\mathbb{X}_{t-1} = i, \mathbb{X}_t = j)$, i.e., the surfer is at node i at time $(t-1)$ and at node j at time t. Suppose that node i is an in-neighbor of j and node j is an in-neighbor of k, i.e., $i \in I_j$ and $j \in I_k$. We have

$$\begin{aligned}&\mathbb{P}[\mathbb{X}_{t-2} = i | \mathbb{X}_{t-1} = j, \mathbb{X}_t = k]\\ =&\frac{\mathbb{P}[\mathbb{X}_{t-2} = i, \mathbb{X}_{t-1} = j]}{\mathbb{P}[\mathbb{X}_{t-1} = j, \mathbb{X}_t = k]} \cdot \mathbb{P}[\mathbb{X}_t = k | \mathbb{X}_{t-2} = i, \mathbb{X}_{t-1} = j]\\ =&\frac{\mathbb{P}[\mathbb{Y}_{t-1} = (i, j)]}{\mathbb{P}[\mathbb{Y}_t = (j, k)]} \cdot p_{i,j,k} \qquad (46)\end{aligned}$$

Thus to calculate the probability of visiting in-neighbors, we only need the prior probability of visiting each edge. Algorithm 13 computes these probabilities and stores them in an $m \times (\eta + 1)$ matrix $\mathbf{Y}$ in the preprocessing stage, with $[\mathbf{Y}]_{(i,j),t} = \mathbb{P}[\mathbb{Y}_t = (i, j)]$, where η is the maximum length of the simulated paths.

Let $\hat{r}_i = (1-c)\sum_{t=0}^{\eta} c^t \mathbb{M}[\Phi_{q,i}^{t,2t}]$ be the SimRank value we try to estimate. Next, we show that our sampling strategy gives accurate estimation of $\hat{r}_i$.

Theorem 13 *The estimated proximity $\tilde{r}_i$ converges to $\hat{r}_i$ when $\pi \to \infty$.*

Proof In Algorithm 11, if we successfully sample a path ending at node i, we will increase $\tilde{r}_i$ by δ; otherwise, $\tilde{r}_i$ is unchanged. For different sampled paths, the corresponding value δ may be different. Let $\mathbb{S}_i^{(d)}$ be a random variable denoting the incremental value of $\tilde{r}_i$ at the dth iteration (lines 3–6). Random variables $\mathbb{S}_i^{(1)}, \mathbb{S}_i^{(2)}, \ldots, \mathbb{S}_i^{(\pi)}$ are independent and identically distributed. Let $\mathbb{S}_i$ be a random variable following the same distribution as $\mathbb{S}_i^{(d)}$'s. Lemma 10 in Appendix E.3 [1] shows that the expected value of $\mathbb{S}_i$ equals the truncated proximity $\hat{r}_i$, i.e., $\mathbb{E}[\mathbb{S}_i] = \hat{r}_i$.

Let $\overline{\mathbb{S}}_i = \frac{1}{\pi}\sum_{d=1}^{\pi} \mathbb{S}_i^{(d)}$ be the sample average, which represents the estimated proximity $\tilde{r}_i$. By the law of large numbers, if the sample size $\pi \to \infty$, $\overline{\mathbb{S}}_i$ converges to the expected value $\mathbb{E}[\mathbb{S}_i] = \hat{r}_i$. This completes the proof of Theorem 13. □

Theorem 14 *For any $\epsilon > 0$, we have that*

$$\mathbb{P}[\tilde{r}_i - \hat{r}_i \le -\epsilon] \le \exp\Big(\frac{-\pi\epsilon^2}{2\kappa m r_i}\Big), \qquad (47)$$

$$\mathbb{P}[\tilde{r}_i - \hat{r}_i \ge \epsilon] \le \exp\Big(\frac{-\pi\epsilon^2}{2\kappa m r_i + 2\kappa m\epsilon/3}\Big), \qquad (48)$$

where κ denotes the maximum in-degree.

Proof Following the notations defined in the proof of Theorem 13, random variables $\mathbb{S}_i^{(1)}, \mathbb{S}_i^{(2)}, \ldots, \mathbb{S}_i^{(\pi)}$ are independent and bounded by interval $[0, \delta] \subseteq [0, \kappa m]$. Lemma 10 in Appendix E.3 [1] shows that the expected value of $\mathbb{S}_i^2$ is bounded from above by $\kappa m r_i$, i.e., $\mathbb{E}[\mathbb{S}_i^2] \le \kappa m r_i$. Thus, we have $\sum_{d=1}^{\pi} \mathbb{E}[(\mathbb{S}_i^{(d)})^2] \le \pi\kappa m r_i$. By Theorem 22 in Appendix E.2 [1], we can prove this theorem. □

Time complexity: Since the path length a follows the geometric distribution, the average length is $(1-c)\sum_{a=0}^{\infty} 2a\,c^a = 2c/(1-c)$. On average, sampling a path costs $O(\psi c/(1-c))$, where ψ denotes the average degree. Sampling π paths costs $O(\pi\psi c/(1-c))$. Initialization and normalization cost $O(n)$. In total, Algorithm 11 runs in $O(\pi\psi c/(1-c) + n)$ time. Algorithm 13 runs in $O(\sigma\eta)$ time.

*7.3.4 Computing the first-order SimRank**

The MC method for the first-order SimRank can be easily extended to compute the first-order SimRank*. In SimRank, only symmetric meeting paths between two nodes can be sampled. In SimRank*, non-symmetric meeting paths can

Algorithm 14 The MC algorithm for the first-order SS

```
Input: $G(V, E)$, query node $q$, decay factor $c$, sample size $\pi$,
       maximum length $\eta$, matrix $\mathbf{X}$
Output: estimated proximity vector $\tilde{\mathbf{r}}$
1: for each node $i \in V$ do $\tilde{r}_i \leftarrow 0$;          // initialization
2: repeat $\pi$ times                                       // sample $\pi$ meeting paths
3:     $b \leftarrow$generate a random number following the geometric
       distribution $\mathbb{P}[\mathbb{B} = b] = (1 - c) \cdot c^b$;
4:     if $b > \eta$ then continue;
5:     $a \leftarrow$generate a random number following the binomial
       distribution $\mathbb{P}[\mathbb{A} = a] = \binom{b}{a}/2^b$;
6:     [bSuccess, $z_b$, $\delta$] $\leftarrow$ SampleOneMeetingPath3($q, a, b, \mathbf{X}$);
7:     if bSuccess = true then $\tilde{r}_{z_b} \leftarrow \tilde{r}_{z_b} + \delta$;
8: for each node $i \in V$ do $\tilde{r}_i \leftarrow \tilde{r}_i/\pi$;     // normalization
```

also be sampled. Algorithm 14 shows the proposed MC algorithm for computing the first-order SimRank*. It is based on the following series expansion

$$r_i = (1-c)\sum_{t=0}^{\infty}\frac{c^t}{2^t}\sum_{a=0}^{t}\binom{t}{a}\mathbb{P}\left[\Phi_{i,q}^{a,t}\right]$$
$$= (1-c)\sum_{t=0}^{\infty}\frac{c^t}{2^t}\sum_{a=0}^{t}\binom{t}{a}\mathbb{P}\left[\Phi_{q,i}^{a,t}\right] \tag{49}$$

That is, the proximity r_i can be represented as the weighted sum of probabilities of visiting all meeting paths of length $\{a, t\}$ between nodes q and i.

Algorithm 15 shows the procedure to sample a meeting path $\phi : q = z_0 \to z_1 \to \cdots \to z_a \leftarrow \cdots \leftarrow z_{b-1} \leftarrow z_b = i$. The sampling process is similar to that in Algorithm 9. When sampling the first half of the path, we simply follow the first-order transition probability. When sampling the second half of the path, we randomly pick in-neighbors of the visited nodes according to the probabilities calculated by using Bayes' theorem. When we calculate these probabilities, we need to use the prior probability of visiting each node. We use Algorithm 10 to compute the prior probabilities and store them in an $n \times (\eta + 1)$ matrix $\mathbf{X}$ in the preprocessing stage, with $[\mathbf{X}]_{j,t} = \mathbb{P}[\mathbb{X}_t = j]$.

Let $\hat{r}_i = (1-c)\sum_{t=0}^{\eta}\frac{c^t}{2^t}\sum_{a=0}^{t}\binom{t}{a}\mathbb{P}[\Phi_{q,i}^{a,t}]$ be the SimRank* value we try to estimate. Next, we show that our sampling strategy gives accurate estimation of $\hat{r}_i$.

Theorem 15 *The estimated proximity $\tilde{r}_i$ converges to $\hat{r}_i$ when $\pi \to \infty$.*

Theorem 16 *For any $\epsilon > 0$, we have that*

$$\mathbb{P}[\tilde{r}_i - \hat{r}_i \le -\epsilon] \le \exp\left(\frac{-\pi\epsilon^2}{2nr_i}\right) \tag{50}$$

$$\mathbb{P}[\tilde{r}_i - \hat{r}_i \ge \epsilon] \le \exp\left(\frac{-\pi\epsilon^2}{2nr_i + 2n\epsilon/3}\right) \tag{51}$$

Please see Appendix E.4 [1] for the proofs.

Algorithm 15 [bSuccess, z_b, δ] $\leftarrow$ SampleOneMeetingPath3($q, a, b, \mathbf{X}$)

```
1: $z_0 \leftarrow q$; bSuccess $\leftarrow$ true;           // start from the query node
2: for $t \leftarrow 1$ to $a$ do                             // sample the first half
3:     if $|O_{z_{t-1}}| = 0$ then bSuccess $\leftarrow$ false, return;
4:     $z_t \leftarrow$randomly pick a node from $O_{z_{t-1}}$ according to the
       first-order transition probability;
5: for $t \leftarrow (a + 1)$ to $b$ do                        // sample the second half
6:     if $|I_{z_{t-1}}| = 0$ or $[\mathbf{X}]_{z_{t-1},b-t+1} = 0$ then bSuccess $\leftarrow$ false, return;
7:     $z_t \leftarrow$randomly pick a node from $I_{z_{t-1}}$ according to the
       probability $p_{z_t,z_{t-1}} \cdot [\mathbf{X}]_{z_t,b-t}/[\mathbf{X}]_{z_{t-1},b-t+1}$;
8: $\delta \leftarrow [\mathbf{X}]_{z_a,b-a}/[\mathbf{X}]_{z_b,0}$;
```

Algorithm 16 The MC algorithm for the second-order SS

```
Input: $G(V, E)$, query node $q$, decay factor $c$, sample size $\pi$,
       maximum length $\eta$, matrix $\mathbf{Y}$
Output: estimated proximity vector $\tilde{\mathbf{r}}$
1: for each node $i \in V$ do $\tilde{r}_i \leftarrow 0$;          // initialization
2: repeat $\pi$ times                                       // sample $\pi$ meeting paths
3:     $b \leftarrow$generate a random number following the geometric
       distribution $\mathbb{P}[\mathbb{B} = b] = (1 - c) \cdot c^b$;
4:     if $b > \eta$ then continue;
5:     $a \leftarrow$generate a random number following the binomial
       distribution $\mathbb{P}[\mathbb{A} = a] = \binom{b}{a}/2^b$;
6:     [bSuccess, $z_b$, $\delta$] $\leftarrow$ SampleOneMeetingPath4($q, a, b, \mathbf{Y}$);
7:     if bSuccess = true then $\tilde{r}_{z_b} \leftarrow \tilde{r}_{z_b} + \delta$;
8: for each node $i \in V$ do $\tilde{r}_i \leftarrow \tilde{r}_i/\pi$;     // normalization
```

Time complexity Since the path length b follows the geometric distribution, the average length is $(1-c)\sum_{b=0}^{\infty} b c^b = c/(1-c)$. On average, sampling a path costs $O(\psi c/(1-c))$, where ψ denotes the average degree. Sampling π paths costs $O(\pi\psi c/(1-c))$. Initialization and normalization cost $O(n)$. In total, Algorithm 14 runs in $O(\pi\psi c/(1-c) + n)$ time.

*7.3.5 Computing the second-order SimRank**

The MC method for the second-order SimRank can be easily extended to compute the second-order SimRank*. Algorithm 16 shows the proposed MC algorithm for computing the second-order SimRank*. It is based on the series expansion

$$r_i = (1-c)\sum_{t=0}^{\infty}\frac{c^t}{2^t}\sum_{a=0}^{t}\binom{t}{a}\mathbb{M}\left[\Phi_{i,q}^{a,t}\right]$$
$$= (1-c)\sum_{t=0}^{\infty}\frac{c^t}{2^t}\sum_{a=0}^{t}\binom{t}{a}\mathbb{M}\left[\Phi_{q,i}^{a,t}\right] \tag{52}$$

That is, the proximity r_i can be represented as the weighted sum of probabilities of visiting all meeting paths of length $\{a, t\}$ between nodes q and i.

Algorithm 17 shows the procedure to sample a meeting path $\phi : q = z_0 \to z_1 \to \cdots \to z_a \leftarrow \cdots \leftarrow z_{b-1} \leftarrow z_b = i$. Similar to the sampling process in Algorithm 12, when simulating the first half of the path, we simply follow the first-order transition probability in the first step, and then, the second-order transition probabilities in the follow-

Algorithm 17 [bSuccess, z_b, δ] $\leftarrow$ SampleOneMeetingPath4($q, a, b, \mathbf{Y}$)

```
1: $z_0 \leftarrow q$; bSuccess $\leftarrow$ true;   // start from the query node
2: for $t \leftarrow 1$ to $a$ do   // sample the first half
3:     if $|O_{z_{t-1}}| = 0$ then bSuccess $\leftarrow$ false, return;
4:     if $t = 1$ then $z_t$ $\leftarrow$randomly pick a node from $O_{z_{t-1}}$ according
       to the first-order transition probability $p_{z_{t-1},z_t}$;
5:     else $z_t$ $\leftarrow$randomly pick a node from $O_{z_{t-1}}$according to the
       second-order transition probability $p_{z_{t-2},z_{t-1},z_t}$;
6: for $t \leftarrow (a+1)$ to $b$ do   // sample the second half
7:     if $|I_{z_{t-1}}| = 0$ or $[\mathbf{Y}]_{(z_{t-1},z_{t-2}),b-t+1} = 0$ then
8:         bSuccess $\leftarrow$ false, return;
9:     if $t = (a+1)$ then $z_t$ $\leftarrow$pick a node from $I_{z_{t-1}}$uniformly
       at random;
10:    else $z_t$ $\leftarrow$randomly pick a node from $I_{z_{t-1}}$ according to
       the probability $p_{z_t,z_{t-1},z_{t-2}} \cdot \frac{[\mathbf{Y}]_{(z_t,z_{t-1}),b-t}}{[\mathbf{Y}]_{(z_{t-1},z_{t-2}),b-t+1}}$;
11: $\delta \leftarrow |I_{z_a}| \cdot p_{z_b,z_{b-1}} \cdot [\mathbf{Y}]_{(z_{a+1},z_a),b-a-1}/[\mathbf{Y}]_{(z_b,z_{b-1}),0}$;
```

ing steps. When simulating the second half of the meeting path, we randomly pick in-neighbors of the visited nodes. In the first step, we pick a node z_{a+1} from I_{z_a} uniformly at random. In the following steps, we use Bayes' theorem to calculate the probabilities. When we calculate the probability of visiting in-neighbors, we need the prior probability of visiting each edge. We use Algorithm 13 to compute these probabilities and store them in an $m \times (\eta + 1)$ matrix $\mathbf{Y}$ in the preprocessing stage, with $[\mathbf{Y}]_{(i,j),t} = \mathbb{P}[\mathbb{Y}_t = (i, j)]$.

Let $\hat{r}_i = (1-c)\sum_{t=0}^{\eta} \frac{c^t}{2^t} \sum_{a=0}^{t} \binom{t}{a} \mathbb{M}[\Phi_{q,i}^{a,t}]$ be the SimRank* value we try to estimate. Next, we show that our sampling strategy gives accurate estimation of $\hat{r}_i$.

Theorem 17 *The estimated proximity $\tilde{r}_i$ converges to $\hat{r}_i$ when $\pi \to \infty$.*

Theorem 18 *For any $\epsilon > 0$, we have that*

$$\mathbb{P}[\tilde{r}_i - \hat{r}_i \leq -\epsilon] \leq \exp\left(\frac{-\pi\epsilon^2}{2\kappa m r_i}\right), \tag{53}$$

$$\mathbb{P}[\tilde{r}_i - \hat{r}_i \geq \epsilon] \leq \exp\left(\frac{-\pi\epsilon^2}{2\kappa m r_i + 2\kappa m\epsilon/3}\right), \tag{54}$$

where κ denotes the maximum in-degree.

Please see Appendix E.5 [1] for the proofs.

Time complexity: Since the path length b follows the geometric distribution, the average length is $(1-c)\sum_{b=0}^{\infty} bc^b = c/(1-c)$. On average, sampling a path costs $O(\psi c/(1-c))$, where ψ denotes the average degree. Sampling π paths costs $O(\pi\psi c/(1-c))$. Initialization and normalization cost $O(n)$. In total, Algorithm 16 runs in $O(\pi\psi c/(1-c) + n)$ time.

8 Experimental results

In this section, we perform comprehensive experimental evaluations on the developed methods. To evaluate the effectiveness of the developed second-order proximity measures, we use both networks with and without the data flow information. We also evaluate the efficiency of proposed computing methods on large real and synthetic networks.

All programs are written in C++, and all experiments are performed on a server with 32G memory, Intel Xeon 3.2GHz CPU, and Redhat OS.

8.1 Networks with data flow information

We first evaluate the effectiveness of the second-order measures using networks with data flow information.

8.1.1 Web domain network with clickstream data

In the web domain network, each node represents a domain, and an edge is weighted by the number of hyperlinks between pages contained in the two connected domains. The web graph was gathered in 2012 and is publicly available at http://webdatacommons.org/hyperlinkgraph/ [26]. It contains 463,824 domains and 6,285,354 edges.

Clickstream data record the sequences of domains visited by different users [5]. The clickstream data are obtained from comScore Inc. It contains 5000 users' clickstreams recorded over 6 months in 2012. The total number of visits is 62.4 million. Each domain was visited 135 times on average.

We first evaluate the domain ranking results of PageRank (PR). The original first-order PR [34], multilinear PR [15], memory PR [36], and our second-order PR (PR^2) are used for comparison. PR uses the first-order transition probability. Multilinear PR approximates the stationary probability of an edge by the product of stationary probabilities of its two end nodes. Memory PR and PR^2 use the second-order transition probabilities based on the frequencies of the trigrams in the clickstream data as discussed in Sect. 3.3.

We use Alexa's top domains (http://www.alexa.com/topsites) as the reference to evaluate the ranking results of the selected measures. The *Normalized Discounted Cumulative Gain* (NDCG) is used as the evaluation metric [46]. The NDCG value at position k is $\text{NDCG}_k = \beta\sum_{i=1}^{k}(2^{s(i)} - 1)/\log_2(1+i)$, where $s(i)$ is the score of the ith node and β is a normalizing factor to ensure the NDCG value of the ground-truth ordering to be 1. We use NDCG_{100} to evaluate the top-100 ranked domains of the selected measures. A retrieved domain gets a score of 1 if it appears in the top-100 domains in Alexa's top domains; otherwise, it gets a score of 0.

In addition to using Alexa's top domains as the reference, we also hired 10 human evaluators to manually evaluate the retrieved domains. An evaluator gives an importance score (ranging from 1 to 5, with 5 being the most important) to each retrieved domain. Table 3 shows the instructions for the judges, and Table 4 shows an example question used in the

Table 3 Instructions in the domain ranking evaluation

Instructions: Pick the best option based on the following criterion	
Very important	Select this if the web domain is very important. For example, "I visited this web domain multiple times every day"
Important	Select this if the web domain is important. For example, "I visited this web domain multiple times every week"
Moderately important	Select this if the web domain is moderately important. For example, "I knew this web domain and I occasionally visited it"
Slightly important	Select this if the web domain is slightly important. For example, "I heard of this web domain before but I never visited it"
Not important	Select this if the web domain is not important at all. For example, "I never heard of this web domain before"

Table 4 Example question in the domain ranking evaluation

Judge the importance of the web domain "google.com"				
Very important	Important	Moderately important	Slightly important	Not important
◎	◎	◎	◎	◎

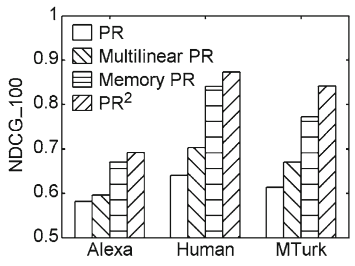

Fig. 6 Ranking accuracy

evaluation. For each domain, the average score of all evaluators is used as its final score. $NDCG_{100}$ is then calculated as the evaluation metric.

We further used Amazon Mechanical Turk (MTurk) to evaluate the retrieved domains. Table 3 shows the guidelines, and Table 4 shows an example question. The guidelines and questions are the same as those for the human evaluators. On average, each retrieved domain is evaluated by 63.4 workers. For each domain, the average score of all workers in MTurk is used as its final score. $NDCG_{100}$ is then calculated as the evaluation metric.

Figure 6 shows the NDCG scores of the selected methods. As we can see, using Alexa's top domains, human evaluators, and Amazon Mechanical Turk as references, the second-order measures, PR^2, memory PR, and multilinear PR, perform better than the original first-order PR. Among the second-order measures, PR^2 and memory PR have higher accuracy than multilinear PR, since the second-order transition probabilities used in PR^2 and memory PR are obtained from the network flow data while the probabilities used in multilinear PR are estimated from the first-order transition probabilities. The better performance of PR^2 over memory PR indicates that the jumping strategy in PR^2 is more effective than the uniform jumping strategy in memory PR.

Next, we evaluate the effectiveness of the proposed measures for the top-k query problem. The evaluated measures include the first-order personalized PageRank (PP), SimRank (SR), and SimRank* (SS) and their second-order forms developed in this paper. We randomly select a web domain as the query node and retrieve the top-20 most relevant domains using the selected measures. We repeat the experiment 100 times. Since there is no ground truth about the proximities between the query nodes and the retrieved nodes, we use both the 10 human evaluators and Amazon Mechanical Turk to evaluate the relevance of retrieved domains. The relevance score ranges from 1 to 5 with 5 being the most relevant. Table 5 shows the instructions for the judges, and Table 6 shows an example question used in the evaluation. In MTurk, on average, each retrieved domain is evaluated by 35.2 workers. The average score of all human evaluators or MTurk workers is used as the final score for a domain. $NDCG_{20}$ then is calculated as the evaluation metric.

Figure 7 shows the accuracy of the selected measures in their first-order and second-order forms. Figure 7a shows the results evaluated by human, and Fig. 7b shows the results evaluated by MTurk. In both evaluation methods, we can see that each second-order measure is more accurate than its first-order counterpart. The second-order measures utilize the real clickstream data to compute the second-order transition probabilities. Since the clickstream data faithfully reflect the similarity among the domains, by leveraging such information, the second-order measures can dramatically improve the accuracy of the results.

Table 5 Instructions in the top-k query evaluation

Instructions: Pick the best option based on the following criterion	
Very relevant	Select this if the two web domains are very relevant. For example, "If I visit one domain, it is very likely that I will visit the other domain"
Relevant	Select this if the two web domains are relevant. For example, "If I visit one domain, it is likely that I will visit the other domain"
Moderately relevant	Select this if the two web domains are moderately relevant. For example, "If I visit one domain, it is moderately likely that I will visit the other domain"
Slightly relevant	Select this if the two web domains are slightly relevant. For example, "If I visit one domain, it is slightly likely that I will visit the other domain"
Not relevant	Select this if the two web domains are not relevant at all. For example, "If I visit one domain, it is unlikely that I will visit the other domain"

Table 6 Example question in the top-k query evaluation

Judge the relevance of the two web domains "apple.com" and "att.com"

Very relevant	Relevant	Moderately relevant	Slightly relevant	Not relevant
◎	◎	◎	◎	◎

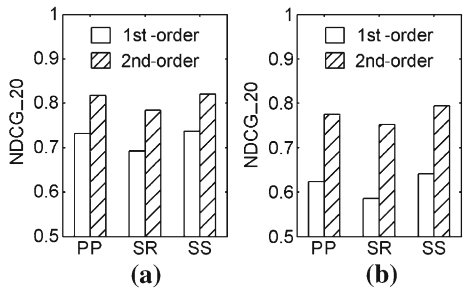

Fig. 7 Query accuracy. **a** Human, **b** MTurk

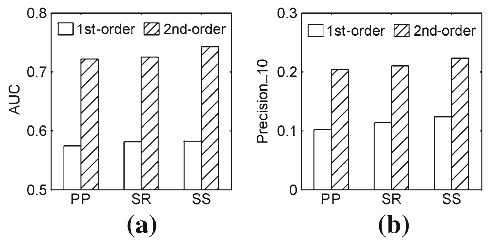

Fig. 8 Link prediction on the Twitter follower network. **a** AUC, **b** Precision$_{10}$

8.1.2 Twitter network with tweet cascade data

A node in the Twitter follower network represents a user and an edge (i, j) represents that user j follows user i. The Twitter follower network used in our experiments was crawled on November 2014. The network contains 231,624 nodes and 3,214,581 edges. We query the timeline from December 2014 to February 2015 of each user once per day to monitor the tweet cascades. The second-order transition probabilities are computed based on frequency of the trigrams in these tweet cascades.

We use the link prediction accuracy to evaluate the effectiveness of the second-order measures. For a given query node, the top-ranked nodes that are not followers of the query node are predicted to follow the query node. The followers newly emerged from March to May 2015 are used as the ground truth to evaluate the predicted results.

We use AUC (area under the ROC curve) and Precision to evaluate the accuracy [30]. AUC can be interpreted as the probability that a randomly chosen user that newly follows the query node is given a higher score than a randomly chosen user that does not follow the query node. Precision is defined as Precision$_k = k'/k$, where k is the total number of predicted users and k' is the number of users that actually started to follow the query node. We randomly pick 10^3 query nodes and report the average.

Figure 8a shows the AUC values of the first-order and second-order PP, SR, and SS. We can observe that the second-order measures improve the AUC values by 23–28% compared to their first-order counterparts. Figure 8b shows the Precision$_{10}$ values. Similarly, the second-order measures outperform the first-order measures consistently. The real tweet cascade data reflect how tweets propagate among different users and provides more accurate transition information than the network topology alone does. The second-order measures use such information thus have better performance.

8.2 Networks without data flow information

When the network data flow information is not available, we use three different applications, including local community detection, link prediction, and graph-based semi-supervised learning, to evaluate the effectiveness of the developed second-order proximity measures. We use the autoregressive model discussed in Sect. 3.3 to obtain the second-order transition probabilities.

8.2.1 Local community detection

The goal of local community detection is to find the community near a given query node [2,42]. Intuitively, the identified local community should contain the nodes having large proximity to the query node. We use the query-biased densest connected subgraph (QB) method [42] and the PageRank-Nibble (NB) method [2] to evaluate the developed second-order measure. For a given query node, both QB and NB compute the node proximity values and use them to find a set of top-ranked nodes as the identified local community. Both methods use the first-order personalized PageRank (PP) as their proximity measure. We simply replace the first-order PP with the proposed second-order PP (PP^2) in QB and NB. All other parts in QB and NB remain the same as the original algorithms. The second-order transition probability is computed by the autoregressive model and the default setting for α is 0.2.

We use F-score and consistency [42] as the evaluation metrics. F-score measures the accuracy of the detected community with regard to the ground-truth community labels. Consistency measures the standard deviation of F-scores of the identified communities when different nodes in the same community are used as the query nodes. A high consistency value indicates that the method tends to find the same local community no matter which node in it is used as the query. We randomly pick 10^3 query nodes and report the average.

We first use real networks to evaluate the performance of the second-order measure. Table 7 shows the statistics of real networks. These datasets are provided with ground-truth community memberships and are publicly available at http://snap.stanford.edu.

Figure 9a shows the F-scores on these networks. QB with PP^2 outperforms QB with PP for 26–44%. NB with PP^2 outperforms NB with PP for 18–62%. Using PP^2, the random surfer is more likely to be trapped within the local community containing the query node, since it takes the previous step of the surfer into consideration. This results that the nodes in

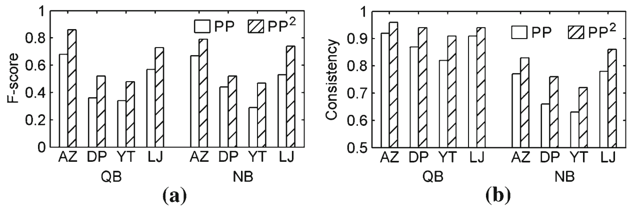

Fig. 9 F-scores and consistency values on real networks. **a** F-score, **b** consistency

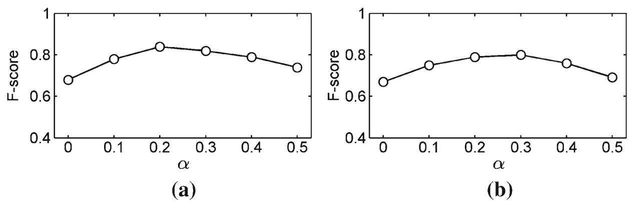

Fig. 10 Tuning the parameter α (Amazon network, PP^2). **a** The QB method, **b** the NB method

the local community have larger proximity values than the nodes outside the community. It helps improve the accuracy of the local community detection methods.

Figure 9b shows the consistency results. QB with PP^2 outperforms QB with PP for 3–11%. NB with PP^2 outperforms NB with PP for 8–15%. High consistency is important for local community detection, since the identified communities should be similar even if different nodes in the same community are used as the query. The higher consistency value of PP^2 demonstrates that it better captures the community structures.

Next we evaluate the sensitivity of PP^2 with respect to the tuning parameter α. Figure 10a shows the F-scores of QB with PP^2 on the Amazon network for different α values. We can see that the performance is stable when varying α. When $\alpha = 0.2$, QB has the best performance. Figure 10a shows the results of NB with PP^2. A similar trend can be observed. Note that when $\alpha = 0$, PP^2 degenerates to PP and has the same performance as PP.

In addition to real networks, we also generate a collection of synthetic networks using the graph generator in [22] to evaluate the developed second-order measures. The number of nodes in the network is 2^{20}, and the number of edges is 10^7. The network generating model contains a mixing param-

Table 7 Statistics of real networks

Datasets	Abbr.	#nodes	#edges	#communities
Amazon	AZ	334,863	925,872	151,037
DBLP	DP	317,080	1,049,866	13,477
YouTube	YT	1,134,890	2,987,624	8,385
LiveJournal	LJ	3,997,962	34,681,189	287,512

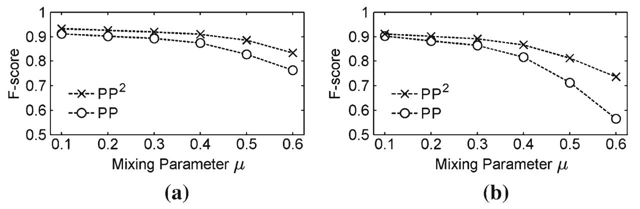

Fig. 11 F-scores on synthetic networks. **a** The QB method, **b** the NB method

eter μ, which indicates the proportion of a node's neighbors that reside in other communities. By tuning μ, we can vary the clearness of the community structure: The boundaries between different communities become less clear for larger μ values.

Figure 11 shows the F-scores on synthetic networks when using QB and NB to detect the local communities. As we can see, PP^2 achieves better performance than PP consistently. Moreover, the performance gap between PP^2 and PP becomes larger for larger μ. This demonstrates that PP^2 is more robust to the noise in the networks than PP . The reason is that in PP^2 the random surfer is likely to stay in the same community rather than to walk across the boundary of the community.

8.2.2 Link prediction

We further evaluate the link prediction accuracy of the second-order measures using the DBLP co-author network. A node in the network represents an author and the edge weight represents the number of papers that two connected authors have co-authored. We select the papers published in the database conferences including SIGMOD, VLDB, ICDE, EDBT, ICDT, and PODS, and data mining conferences including KDD, ICDM, SDM, PKDD, and PAKDD, from 2004 to 2013. The papers published in 2004 to 2008 are used to construct the training network, which contains 146,527 nodes and 426,835 edges. The papers published in 2009 to 2013 are used to obtain the newly emerged links among the authors, which are used as the ground truth for testing.

The left figure in Figure 12 shows the AUC values. We can see that the second-order measures improve the AUC value by 9–16% compared to the first-order measures. The right figure shows the Precision$_{10}$ values. The second-order measures improve the Precision$_{10}$ value by 24–47%. Since the

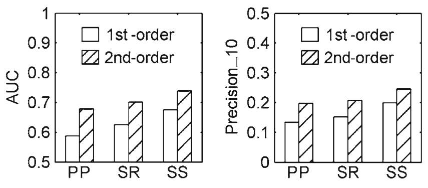

Fig. 12 Link prediction on the co-author network

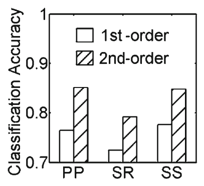

Fig. 13 Graph-based semi-supervised learning

second-order measures better capture the community structure in the network, they can significantly improve the link prediction accuracy.

8.2.3 Graph-based semi-supervised learning

In graph-based semi-supervised learning, a graph is constructed to connect similar data objects [49]. The goal is to predict the unknown class labels using the partially labeled data.

We use the USPS dataset, which contains 9, 298 images of handwritten digits from 0 to 9 [25,48]. A weighted k-NN graph is constructed with $k = 20$. We use the Gaussian kernel [49] to compute the edge weight $w_{i,j}$ if i is within j's k nearest neighbors or vice versa. We randomly pick 20 nodes as labeled nodes and make sure that there is at least one labeled node for each class. The label of the nearest neighbor is used as the predicted class label for unlabeled nodes. We repeat this process 10^3 times and report the average classification accuracy.

Figure 13 shows the classification accuracy using PP, SR, and SS in the first-order and second-order forms. We can see that the second-order measures outperform their first-order counterparts. The second-order measures take the community structure in the k-NN graph into account and thus have better performance than the first-order measures.

8.3 Efficiency evaluation

We evaluate the efficiency of the proposed single-source algorithms and Monte Carlo (MC) methods on real and synthetic networks. Table 7 shows the statistics of real networks. The synthetic graphs are based on the R-MAT model [6]. We use the graph generator available at http://github.com/dhruvbird/GTgraph and its default parameters to generate a series of graphs with different sizes. Table 8 shows the statistics. The algorithms for PR have similar performance as those for PP. Thus, we focus on PP, SR, and SS.

In the MC methods, we set the parameter $\eta = 20$ and $\pi = 4n$, where n denotes the number of nodes in the graph. Let $\mathbf{r}$ and $\tilde{\mathbf{r}}$ denote the exact and estimated proximity vectors, respectively. The error of the MC method is defined as $\text{Error} = \|\mathbf{r} - \tilde{\mathbf{r}}\|_1 / \|\mathbf{r}\|_1$, where $\|\mathbf{r}\|_1 = \sum_i |r_i|$ denotes the sum of absolute values. In PP^2, the exact proximity values are computed by the power method. In SR^2 and SS^2, the

Table 8 Statistics of synthetic networks

#nodes	1×2^{24}	2×2^{24}	4×2^{24}	8×2^{24}
#edges	2.5×10^8	5×10^8	1×10^9	2×10^9

exact proximity values are computed by Algorithm 2 and 4, respectively, since the power methods for SR^2 and SS^2 are prohibitive on large graphs. The parameter η in Algorithm 2 or 4 is also set to 20.

8.3.1 Algorithms for the second-order measures

Figure 14a shows the running time of the power iteration method and the developed MC algorithm for PP^2 on real networks. We can see that the MC method is 1–2 orders of magnitude faster than the power method. Note that in these experiments, the error of the MC method is less than 10^{-2}. Thus with little loss in accuracy, the MC algorithm can dramatically improve the running time. Figure 14b shows the running time of the single-source algorithm and the MC method for SR^2. Figure 14c shows the running time of the single-source algorithm and the MC method for SS^2. For both SR^2 and SS^2, the MC method is 1–2 orders of magnitude faster than the single-source algorithm and the error of the MC method is less than 10^{-2}.

Figure 15a shows the running time of the algorithms for PP^2 on synthetic networks. Similarly, the MC method is 1–2 orders of magnitude faster than the power method. Figure 15b shows the running time of the algorithms for SR^2. Figure 15c shows the running time of the algorithms for SS^2. For both SR^2 and SS^2, the MC method is 1–2 orders of magnitude faster than the single-source algorithm. The error of the MC method is less than 10^{-2} in all these experiments.

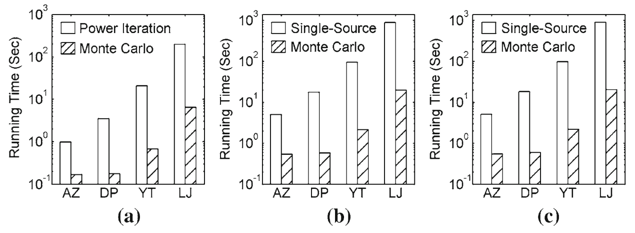

Fig. 14 Running time on real networks. **a** PP^2, **b** SR^2, **c** SS^2

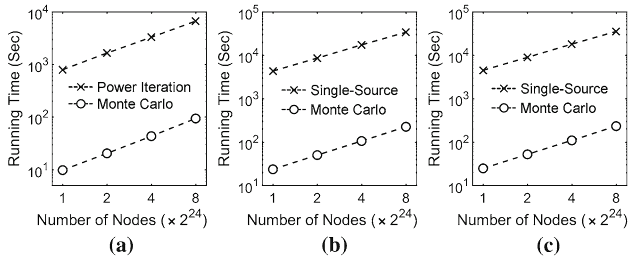

Fig. 15 Running time on synthetic networks. **a** PP^2, **b** SR^2, **c** SS^2

8.3.2 The developed sampling strategy

We further compare the sampling strategy in the MC method developed in [10], and our sampling strategy described in Algorithm 9. Recall that the method in [10] samples meeting paths starting from the query node q and every other node i, while our method samples the meeting paths all starting from the query node. We compare the running time that the two methods need to take to achieve the same accuracy. When varying the number of sampled paths, the running time and accuracy of the two methods will change correspondingly. For each setting, we repeat the query 10^2 times with randomly picked query nodes and report the average running time and error.

Figure 16a shows the error versus running time on the LiveJournal network. We can see that to achieve the same accuracy, the proposed method is about 3 orders of magnitude faster than the previous method. This demonstrates the advantage of the proposed sampling strategy. Figure 16b shows the error versus running time on the synthetic graph with 2^{26} nodes. A similar trend can be observed.

8.3.3 First order versus second order

In the next, we further compare the efficiency of the algorithms for computing the first-order and second-order measures on real and synthetic networks.

Figure 17a shows the running time of the power methods for PP and PP^2 on real networks. The power method for PP^2 is slower than that for PP. The reason is that the power

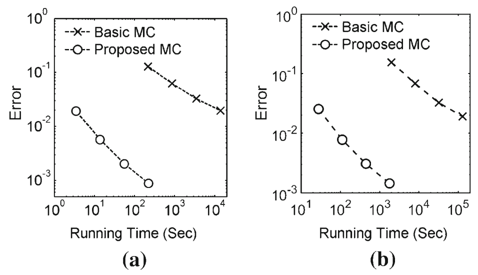

Fig. 16 Error versus running time (first-order SimRank). **a** Real network, LiveJournal, **b** Synthetic network, 2^{26} nodes

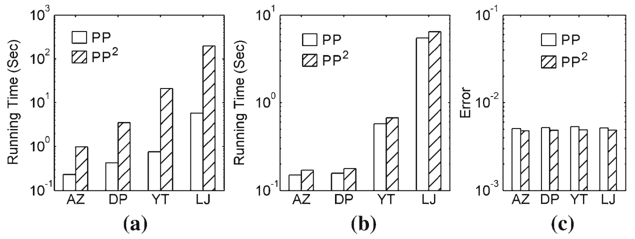

Fig. 17 PP, real networks. **a** Power method, **b** MC, runtime, **c** MC, error

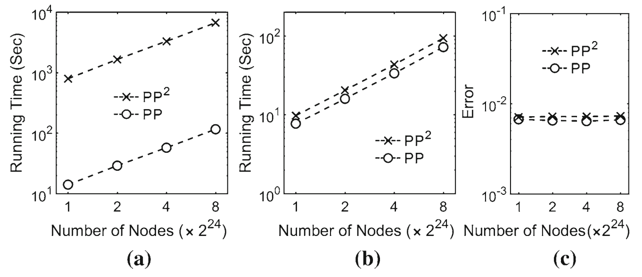

Fig. 18 PP, synthetic networks. **a** Power method, **b** MC, runtime, **c** MC, error

method needs to compute matrix–vector products; thus, the time complexity is proportional to the number of nonzero elements in the transition matrix. In PP, the number of nonzero elements in the node-to-node transition matrix is equal to the number of edges in the graph. In PP^2, the number of nonzero elements in the edge-to-edge transition matrix is equal to the number of paths of length 2 in the graph. In real networks, the number of paths of length 2 is usually 1–2 orders of magnitude larger than the number of edges.

Figure 17b shows the running time of the MC methods for PP and PP^2 on real networks. We can observe that the MC method for PP^2 runs as efficiently as that for PP. The reason is that the time complexity of the MC method depends on the sample size and the average degree in the graph. It is independent on the number of nonzero elements in the transition matrix since the MC method does not compute matrix–vector products. Figure 17c shows the error of the MC methods for PP and PP^2 on real networks. We can see that the two methods generate similar errors. From Fig. 17, we can see that the MC method is especially suitable for computing PP^2.

Figure 18a shows the running time of the power methods for PP and PP^2 on synthetic networks. Similar to the results on real networks, the power method for PP^2 is slower than that for PP.

Figure 18b shows the running time of the MC methods for PP and PP^2 on synthetic networks. The MC method for PP^2 runs as efficiently as that for PP. Figure 18c shows the error of the MC methods for PP and PP^2 on synthetic networks. The two methods generate similar errors. From Fig. 18, we can conclude that the MC method is more suitable for computing PP^2.

Figure 19a shows the running time of the single-source algorithms for SR and SR^2 on real networks. The single-source algorithm needs to compute matrix–vector products; thus, the time complexity is proportional to the number of nonzero elements in the transition matrix.

Figure 19b shows the running time of the MC methods for SR and SR^2 on real networks. The MC method for SR^2 runs as efficiently as that for SR. The reason is that the time complexity of the MC method depends on the sample size

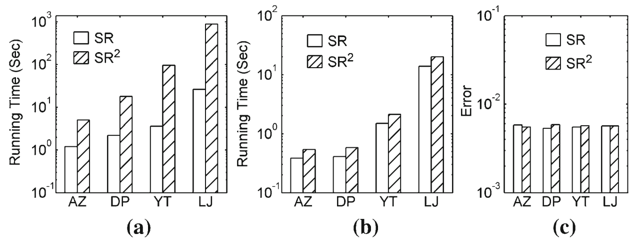

Fig. 19 SR, real networks. **a** Single-source, **b** MC, runtime, **c** MC, error

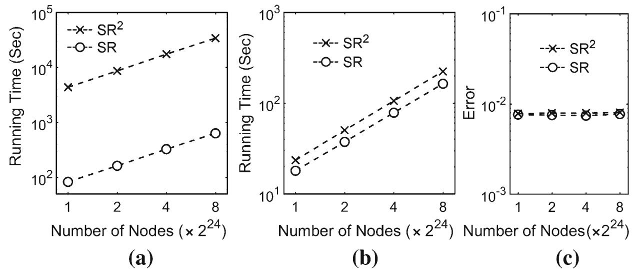

Fig. 20 SR, synthetic networks. **a** Single-source, **b** MC, runtime, **c** MC, error

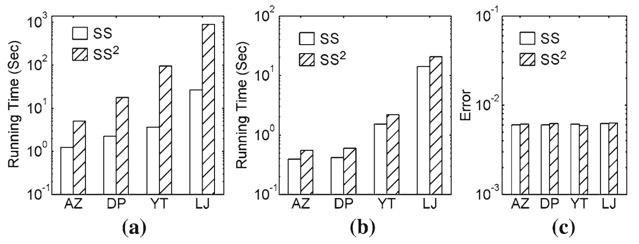

Fig. 21 SS, real networks. **a** Single-source, **b** MC, runtime, **c** MC, error

and the average degree in the graph. Figure 19c shows the error of the MC methods for SR and SR^2 on real networks. The two methods generate similar errors. From Fig. 19, we can see that the MC method is more suitable for SR^2.

Figure 20a shows the running time of the single-source algorithms for SR and SR^2 on synthetic networks. Figure 20b shows the running time of the MC methods for SR and SR^2 on synthetic networks. Similar results can be observed as on real networks.

Figure 21a shows the running time of the single-source algorithms for SS and SS^2 on real networks. Figure 21b shows the running time of the MC methods for SS and SS^2 on real networks. Figure 21c shows the error of the MC methods. The patterns observed from Fig. 21 are similar to those observed from Fig. 19. From Fig. 21, we can see that the MC method is more suitable for SS^2.

Figure 22a shows the running time of the single-source algorithms for SS and SS^2 on synthetic networks. Figure 22b shows the running time of the MC methods for SS and

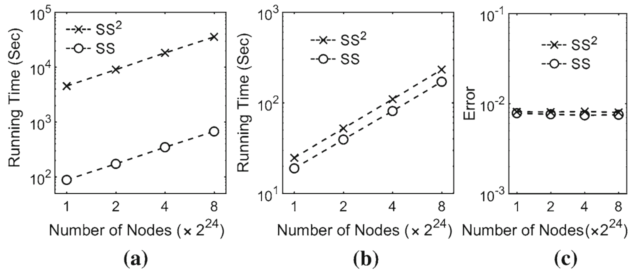

Fig. 22 SS, synthetic networks. **a** Single-source, **b** MC, runtime, **c** MC, error

SS^2 on synthetic networks. Similar results can be observed as on real networks.

In conclusion, both the power method and the single-source method need to compute matrix–vector products; thus they, run slower when they are used for computing the second-order measures. The MC method does not need to compute matrix–vector products, and it still runs efficiently when it is used for computing the second-order measures.

9 Conclusions

Designing effective proximity measures for large graphs is an important and challenging task. Most existing random walk-based measures only use the first-order transition probability. In this paper, we investigate the second-order random walk measures which can capture the cluster structures in the graph and better model real-life applications. We provide rigorous theoretical foundations for the second-order random walk and develop second-order forms for commonly used measures. We further develop effective Monte Carlo methods to compute these measures. Extensive experimental results demonstrate that the second-order measures can effectively improve the accuracy in various applications, and the developed Monte Carlo methods can significantly speed up the computation with little loss in accuracy.

Acknowledgements This work was partially supported by the National Science Foundation Grants IIS-11623-74, CAREER, and the NIH Grant R01GM115833.

References

1. www.robwu.net. Accessed 20 Nov 2017
2. Andersen, R., Chung, F., Lang, K.: Local graph partitioning using PageRank vectors. In: FOCS, pp. 475–486 (2006)
3. Benson, A.R., Gleich, D.F., Leskovec, J.: Tensor spectral clustering for partitioning higher-order network structures. In: SDM, pp. 118–126 (2015)
4. Benson, A.R., Gleich, D.F., Leskovec, J.: Higher-order organization of complex networks. Science **353**(6295), 163–166 (2016)
5. Bucklin, R.E., Sismeiro, C.: Click here for internet insight: advances in clickstream data analysis in marketing. J. Interact. Mark. **23**(1), 35–48 (2009)
6. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-MAT: a recursive model for graph mining. In: SDM, pp. 442–446 (2004)
7. Chung, F., Lu, L.: Old and new concentration inequalities. In: Chung, F., Lu, L. (eds.) Complex Graphs and Networks Chap 12. AMS, Providence (2006)
8. Cohen, S., et al.: A survey on proximity measures for social networks. In: Search Computing, pp. 191–206 (2012)
9. Fang, Y., Chang, K.C.-C., Lauw, H.W.: Roundtriprank: graph-based proximity with importance and specificity? In: ICDE, pp. 613–624 (2013)
10. Fogaras, D., Rácz, B.: Scaling link-based similarity search. In: WWW, pp. 641–650 (2005)
11. Fogaras, D., Rácz, B., Csalogány, K., et al.: Towards scaling fully personalized PageRank: algorithms, lower bounds, and experiments. Internet Math. **2**(3), 333–358 (2005)
12. Fouss, F., Pirotte, A., Renders, J.-M., Saerens, M.: Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. TKDE **19**(3), 355–369 (2007)
13. Fujiwara, Y., Nakatsuji, M. et al.: Efficient search algorithm for SimRank. In: ICDE, pp. 589–600 (2013)
14. Gleich, D.F.: Pagerank beyond the web. SIAM Rev. **57**(3), 321–363 (2015)
15. Gleich, D.F., Lim, L.-H., Yu, Y.: Multilinear PageRank. SIAM J. Matrix Anal. Appl. **36**(4), 1507–1541 (2015)
16. He, G., Feng, H., Li, C., Chen, H.: Parallel SimRank computation on large graphs with iterative aggregation. In: SIGKDD, pp. 543–552 (2010)
17. Hoeffding, W.: Probability inequalities for sums of bounded random variables. JASA **58**(301), 13–30 (1963)
18. Jeh, G., Widom, J.: SimRank: a measure of structural-context similarity. In: KDD, pp. 538–543 (2002)
19. Jeh, G., Widom, J.: Scaling personalized web search. In: WWW, pp. 271–279 (2003)
20. Katz, L.: A new status index derived from sociometric analysis. Psychometrika **18**(1), 39–43 (1953)
21. Kusumoto, M., Maehara, T., Kawarabayashi, K.: Scalable similarity search for SimRank. In: SIGMOD, pp. 325–336 (2014)
22. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Phys. Rev. E **78**(4), 046110 (2008)
23. Langville, A.N., Meyer, C.D.: Deeper inside PageRank. Internet Math. **1**(3), 335–380 (2004)
24. Langville, A.N., Meyer, C.D.: The mathematics guide. In: Langville, A.N., Meyer, C.D. (eds.) Google's PageRank and Beyond: The Science of Search Engine Rankings. Princeton University Press, Princeton (2011)
25. LeCun, Y., Boser, B.E., Denker, J.S., et al.: Handwritten digit recognition with a back-propagation network. In: NIPS (1990)
26. Lehmberg, O., et al.: Graph structure in the web: aggregated by pay-level domain. In: WebSci, pp. 119–128 (2014)
27. 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, pp. 465–476 (2010)
28. Liben-Nowell, D., Kleinberg, J.: The link-prediction problem for social networks. JASIST **58**(7), 1019–1031 (2007)
29. Lim, S., Ryu, S., Kwon, S., Jung, K., Lee, J.-G.: LinkSCAN*: overlapping community detection using the link-space transformation. In: ICDE, pp. 292–303 (2014)
30. Lü, L., Zhou, T.: Link prediction in complex networks: a survey. Physica A **390**(6), 1150–1170 (2011)
31. Maehara, T., Kusumoto, M., et al.: Efficient SimRank computation via linearization. arXiv:1411.7228 (2014)
32. Mei, Q., Zhou, D., Church, K.: Query suggestion using hitting time. In: CIKM, pp. 469–478 (2008)

33. Meyer, C.D.: Matrix Analysis and Applied Linear Algebra. SIAM, Philadelphia (2000)
34. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: bringing order to the web. Stanford InfoLab, Stanford (1999)
35. Raftery, A.E.: A model for high-order Markov chains. J. R. Stat. Soc. Ser. B **47**(3), 528–539 (1985)
36. Rosvall, M., Esquivel, A.V., Lancichinetti, A., et al.: Memory in network flows and its effects on spreading dynamics and community detection. Nat. Commun. **5**, 4630 (2014)
37. Rothe, S., Schütze, H.: CoSimRank: a flexible & efficient graph-theoretic similarity measure. In: ACL, pp. 1392–1402 (2014)
38. Sarkar, P., Moore, A.: A tractable approach to finding closest truncated-commute-time neighbors in large graphs. In: UAI (2012)
39. Sarkar, P., Moore, A.W.: Fast nearest-neighbor search in disk-resident graphs. In: KDD, pp. 513–522 (2010)
40. Tong, H., Faloutsos, C., Pan, J.-Y.: Fast random walk with restart and its applications. In: ICDM, pp. 613–622 (2006)
41. Wu, Y., Bian, Y., Zhang, X.: Remember where you came from: on the second-order random walk based proximity measures. PVLDB **10**(1), 13–24 (2017)
42. Wu, Y., Jin, R., Li, J., Zhang, X.: Robust local community detection: on free rider effect and its elimination. PVLDB **8**(7), 798–809 (2015)
43. Wu, Y., Jin, R., Zhang, X.: Fast and unified local search for random walk based k-nearest-neighbor query in large graphs. In SIGMOD, pp. 1139–1150 (2014)
44. Wu, Y., Jin, R., Zhang, X.: Efficient and exact local search for random walk based top-*k* proximity query in large graphs. TKDE **28**(5), 1160–1174 (2016)
45. Yu, W., Lin, X., Le, J.: Taming computational complexity: efficient and parallel SimRank optimizations on undirected graphs. In WAIM, pp. 280–296 (2010)
46. Yu, W., Lin, X., Zhang, W., Chang, L., Pei, J.: More is simpler: effectively and efficiently assessing node-pair similarities based on hyperlinks. PVLDB **7**(1), 13–24 (2013)
47. Zhang, C., Shou, L., Chen, K., Chen, G., Bei, Y.: Evaluating geo-social influence in location-based social networks. In CKIM, pp. 1442–1451 (2012)
48. Zhu, X., Ghahramani, Z., Lafferty, J.: Semi-supervised learning using Gaussian fields and harmonic functions. In: ICML, pp. 912–919 (2003)
49. Zhu, X., Goldberg, A.: Graph-based semi-supervised learning. In: Zhu, X., Goldberg, A. (eds.) Introduction to Semi-supervised Learning. Morgan & Claypool Publishers, San Rafel (2009)