Susceptible-infected-spreading-based network embedding in static and temporal networks

Zhan, Xiu Xiu; Li, Ziyu; Masuda, Naoki; Holme, Petter; Wang, Huijuan

DOI
10.1140/epjds/s13688-020-00248-5

Publication date
2020

Document Version
Final published version

Published in
EPJ Data Science

Zhan et al. *EPJ Data Science* (2020) 9:30
https://doi.org/10.1140/epjds/s13688-020-00248-5

EPJ Data Science
a SpringerOpen Journal

EPJ.org

REGULAR ARTICLE Open Access

Susceptible-infected-spreading-based network embedding in static and temporal networks

Xiu-Xiu Zhan[1], Ziyu Li[1], Naoki Masuda[2,3], Petter Holme[4] and Huijuan Wang[1*]

*Correspondence:
H.Wang@tudelft.nl
[1]Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands
Full list of author information is available at the end of the article

Abstract

Link prediction can be used to extract missing information, identify spurious interactions as well as forecast network evolution. Network embedding is a methodology to assign coordinates to nodes in a low-dimensional vector space. By embedding nodes into vectors, the link prediction problem can be converted into a similarity comparison task. Nodes with similar embedding vectors are more likely to be connected. Classic network embedding algorithms are random-walk-based. They sample trajectory paths via random walks and generate node pairs from the trajectory paths. The node pair set is further used as the input for a Skip-Gram model, a representative language model that embeds nodes (which are regarded as words) into vectors. In the present study, we propose to replace random walk processes by a spreading process, namely the susceptible-infected (SI) model, to sample paths. Specifically, we propose two susceptible-infected-spreading-based algorithms, i.e., *S*usceptible-*I*nfected *N*etwork *E*mbedding (*SINE*) on static networks and *T*emporal *S*usceptible-*I*nfected *N*etwork *E*mbedding (*TSINE*) on temporal networks. The performance of our algorithms is evaluated by the missing link prediction task in comparison with state-of-the-art static and temporal network embedding algorithms. Results show that *SINE* and *TSINE* outperform the baselines across all six empirical datasets. We further find that the performance of *SINE* is mostly better than *TSINE*, suggesting that temporal information does not necessarily improve the embedding for missing link prediction. Moreover, we study the effect of the sampling size, quantified as the total length of the trajectory paths, on the performance of the embedding algorithms. The better performance of *SINE* and *TSINE* requires a smaller sampling size in comparison with the baseline algorithms. Hence, SI-spreading-based embedding tends to be more applicable to large-scale networks.

Keywords: Network embedding; SI spreading process; Link prediction

1 Introduction

Real-world systems can be represented as networks, with nodes representing the components and links representing the connections between them [1, 2]. The study of complex networks pervades in different fields [3]. For example, with biological or chemical networks, scientists study interactions between proteins or chemicals to discover new drugs

Springer

[4, 5]. With social networks, researchers tend to classify or cluster users into groups or communities, which is useful for many tasks, such as advertising, search and recommendation [6, 7]. With communication networks, learning the network structure can help understand how information spreads over the networks [2, 8]. These are only a few examples of the important role of analyzing networks. For all these examples, the data may be incomplete. If so, it could be important to be able to predict the links most likely to be missing. If the network is evolving, it could be crucial to forecast the next link to be added. For both of these applications one needs link prediction [9–13].

In link prediction, one estimates the likelihood that two nodes are adjacent to each other based on the observed network structure [14]. Methods using similarity-based metrics, maximum likelihood algorithms and probabilistic models are major families of link prediction methods [15]. Recently, network embedding, which embeds nodes into a low-dimensional vector space, has attracted much attention in solving the link prediction problem [15–20]. The similarity between the embedding vectors of two nodes is used to evaluate whether they would be connected or not. Different algorithms have been proposed to obtain network embedding vectors. A simplest embedding method is to take the row or column vector in the adjacency matrix, which is called an adjacency vector of the corresponding node, as the embedding vector. Then, the representation space is N-dimensional, where N is the number of nodes. As real-world networks are mostly large and sparse, the adjacency vector of a node is sparse and high-dimensional. In addition, the adjacency matrix only contains the first-order neighborhood information, and therefore the adjacency vector neglects the high-order structure of the network such as paths that are longer than an edge. These factors limit the precision of network embedding based on the adjacency vector in link prediction tasks. Work in the early 2000s attempted to embed nodes into a low dimension space using dimension reduction techniques [21–23]. Isomap [21], locally linear embedding (LLE) [22] and Laplacian eigenmap [23] are algorithms based on the k-nearest graph, where nodes i and j are connected by a link in the k-nearest graph if the length of the shortest path between i and j is within the k-th shortest among the length of all the shortest paths from i to any other nodes. Matrix factorization algorithms decompose the adjacency matrix into the product of two low-dimensional rectangular matrices. The columns of the rectangular matrices are the embedding vectors for nodes. Singular value decomposition (SVD) [24] is one commonly used and simple matrix factorization algorithm. However, the computation complexity of most of the aforementioned algorithms is at least quadratic in terms of N, limiting their applicability to large networks with millions of nodes.

Random-walk-based network embedding is a promising family of computationally efficient algorithms. These algorithms exploit truncated random walks to capture the proximity between nodes [25–27] generally via the following three steps [28–30]: (1) Sample the network by running random walks to generate trajectory paths. (2) Generate a node pair set from the trajectory paths: each node on the trajectory path is viewed as a center node, the nearby nodes within a given distance are considered as the neighboring nodes. A node pair in the node pair set is formed by a center node and each of its neighboring nodes. (3) Apply a word embedding model such as Skip-Gram to learn the embedding vector for each node by using the node pair set as input. Skip-Gram assumes nodes that are similar in topology or content tend to have similar representations [27]. Algorithms have been designed using different random walks to capture high-order structure on networks. For

example, DeepWalk [25] and Node2Vec [28] adopted uniform and biased random walks, respectively, to sample the network structure. In addition, random-walk-based embedding methods have also been developed for temporal networks, signed networks and multilayer networks [31–34].

In contrast to random-walk-based embedding, here we propose SI-spreading-based network embedding algorithms for static and temporal networks. We deploy the susceptible-infected (SI) spreading process on the given network, either static or temporal, and use the corresponding spreading trajectories to generate the node pair set, which is fed to the Skip-Gram to derive the embedding vectors. The trajectories of an SI spreading process capture the tree-like sub-network centered at the seed node, whereas random walk explores long walks that possibly revisit the same node. We evaluate our static network embedding algorithm, named *SINE*,[a] and temporal network embedding, *TSINE*, via a missing link prediction task in six real-world social networks. We compare our algorithms with state-of-the-art static and temporal network embedding methods. We show that both *SINE* and *TSINE* outperform other static and temporal network embedding algorithms, respectively. In most cases, the static network embedding, *SINE*, performs better than *TSINE*, which additionally uses temporal network information. In addition, we evaluate the efficiency of SI-spreading-based network embedding via exploring the sampling size for the Skip-Gram, quantified as the sum of the length of trajectory paths, in relation to its performance on the link prediction task. We show that high performance of SI-spreading-based network embedding algorithms requires a significantly smaller sampling size compared to random-walk-based embeddings. We further explore what kind of links can be better predicted to further explain why our proposed algorithms show better performance than the baselines.

The rest of the paper is organized as follows. We propose our method in Sect. 2. In Sect. 2.1, we propose our SI-spreading-based sampling method for static networks and generation of the node pair set from the trajectory paths. Skip-Gram model is introduced in Sect. 2.2. We introduce an SI-spreading-based sampling method for temporal networks in Sect. 2.3. In Sect. 3, our embedding algorithms are evaluated on a missing link prediction task on real-world static and temporal social networks. The paper is concluded in Sect. 4.

2 SI-spreading-based embedding algorithm

This section introduces SI-spreading-based network embedding methods. Firstly, we illustrate our SI-spreading-based network embedding method for static networks in Sects. 2.1 and 2.2. Section 2.3 generalizes the method to temporal network.

Because we propose the network embedding methods for both static and temporal networks, we start with the notations for temporal networks, of which the static networks are special cases. A temporal network is represented as $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the node set and $\mathcal{L} = \{l(i,j,t), t \in [0,T], i,j \in \mathcal{N}\}$ is the set of time-stamped contacts. The element $l(i,j,t)$ in $\mathcal{L}$ represents a bidirectional contact between nodes i and j at time t. We consider discrete time and assume that all contacts have a duration of one discrete time step.[b] We use $[0,T]$ to represent the observation time window, and $N = |\mathcal{N}|$ is the number of nodes. The aggregated static network $G = (\mathcal{N}, E)$ is derived from a temporal network $\mathcal{G}$. Two nodes are connected in G if there is at least one contact between them in $\mathcal{G}$. E is the edge set of G. We formulate the network embedding problem as follows:

Given a network $G = (\mathcal{N}, E)$, static network embedding aims to learn a low-dimensional representation for each node $i \in \mathcal{N}$. The node embedding matrix for all the nodes is given by $\mathbf{U} \in R^{d \times N}$, where d is the dimension of the embedding vectors ($d < N$). The i-th column of $\mathbf{U}$, i.e., $\overrightarrow{u_i} \in R^{d \times 1}$, represents the embedding vector of node i.

2.1 SI-spreading-based static network sampling

The SI spreading process on a static network is defined as follows: each node is in one of the two states at any time step, i.e., susceptible (S) or infected (I); initially, one seed node is infected; an infected node independently infects each of its susceptible neighbors with an infection probability β at each time step; the process stops when no node can be infected further. To derive the node pair set as the input for Skip-Gram, we carry out the following steps as described in Sections 2.1.1 and 2.1.2.

2.1.1 Construction of spreading trajectory paths

In each iteration or realization of the SI spreading process, a node i is selected uniformly at random as the seed. We perform the SI spreading process from seed node i. The spreading trajectory $\mathcal{T}_i(\beta)$ is the union of all the nodes that finally get infected supplied with all the links that have transmitted infection between node pairs. If a susceptible node i becomes infected via more than one infected neighbors at the same time step, we uniformly randomly choose one of these infected neighbors, say j, and assume i gets infected only by neighbor j via the edge in between. Hence, edge (i, j) will be included into the spreading trajectories. The spreading trajectories are exactly trees under this assumption.

From each of the spreading trajectory $\mathcal{T}_i(\beta)$, we construct m_i trajectory paths, each of which is the path between the root node i and a randomly selected leaf node in $\mathcal{T}_i(\beta)$. The

Algorithm 1 Generation of trajectory paths from SI spreading process

Input: $G = (\mathcal{N}, E)$, B, $L_{\max}$, β, m_i

Output: node trajectory path set D

Initialize number of context windows $C = 0$
Initialize node trajectory path set $D = \varnothing$
while $B - C > 0$ **do**
    Randomly choose node i as the seed to start the SI spreading
    Generate spreading trajectory tree $\mathcal{T}_i(\beta)$
    Randomly choose m_i trajectory paths $D_{g_i}(g_i = 1, \ldots, m_i)$ from $\mathcal{T}_i(\beta)$
    for $g_i = 1, \ldots, m_i$ **do**
        if $|D_{g_i}| > L_{\max}$ **then**
            Choose the first $L_{\max}$ nodes from D_{g_i} to form $D^*_{g_i}$
            Add the trajectory $D^*_{g_i}$ to D
            $C = C + |D^*_{g_i}|$
        else
            Add the trajectory D_{g_i} to D
            $C = C + |D_{g_i}|$
        end if
    end for
end while
return D

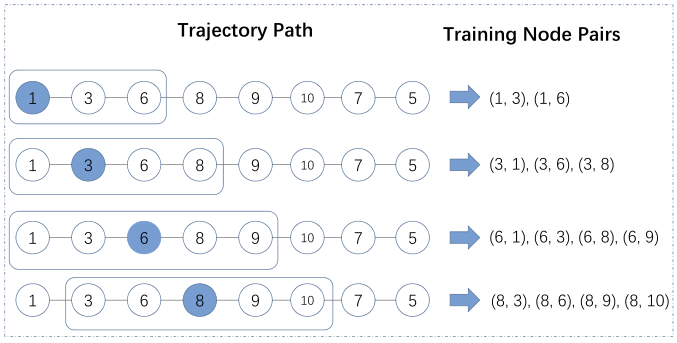

Figure 1 Generating node pairs from a trajectory path 1, 3, 6, 8, 9, 10, 7, 5. The window size $\omega = 2$ and only the first four nodes 1, 3, 6 and 8 as the center node are illustrated as examples

number m_i of trajectory paths to be extracted from $\mathcal{T}_i(\beta)$ is assumed to be given by

$$m_i = \max\left\{1, \frac{\mathcal{K}(i)}{\sum_{j\in\mathcal{N}} \mathcal{K}(j)} m_{\max}\right\},$$

where $m_{\max}$ is a control parameter and $\mathcal{K}(i)$ is the degree of the root node i in the static network (or aggregated network).

The trajectory paths may have different lengths (i.e., number of nodes in the path). For a trajectory path whose length is larger than $L_{\max} = 20$, we only take the first $L_{\max}$ nodes on the path.

From each iteration of the SI spreading process that starts from a randomly chosen seed node i, we can generate a spreading trajectory $\mathcal{T}_i(\beta)$ and m_i trajectory paths from $\mathcal{T}_i(\beta)$. We stop such iteration to generate new trajectory paths once the total length of the trajectory paths collected reaches the sampling size $B = NX$, where X is a control parameter. We consider $X \in \{1, 2, 5, 10, 25, 50, 100, 150, 200, 250, 300, 350\}$. We compare different algorithms using the same B for fair comparison [31] to understand the influence of the sampling size. We show how to sample the trajectory paths in Algorithm 1.

2.1.2 Node pair set generation

We illustrate how to generate the node pairs, the input of the Skip-Gram, from a trajectory path in Fig. 1. Consider a trajectory path, 1, 3, 6, 8, 9, 10, 7, 5, starting from node 1 and ending at node 5. We set each node, e.g., node 3, as the center node, and the neighboring nodes of the center node are defined as nodes within $\omega = 2$ hops. The neighboring nodes of node 3 are, 1, 6 and 8. We thus obtain ordered node pairs $(3, 1)$, $(3, 6)$, and $(3, 8)$. Thus, we use the union of node pairs centered at each node in each of trajectory path as the input to the Skip-Gram model.

2.2 Skip-Gram model

We illustrate how the Skip-Gram derives the embedding vector for each node based on the input node pair set. We denote by $N_{SI}(i)$ the neighboring set for a node i derived from the SI spreading process. A neighboring node j of i may appear multiple times in $N_{SI}(i)$ if (i, j) appears multiple times in the node pair set.

Given node i, let $p(j|i)$ be the probability of observing neighboring node j. We model the conditional probability $p(j|i)$ as the softmax unit parametrized by the product of the

embedding vectors, i.e., $\overrightarrow{u_i}$ and $\overrightarrow{u_j}$, as follows:

$$p(j|i) = \log \frac{\exp(\overrightarrow{u_i} \cdot \overrightarrow{u_j}^T)}{\sum_{k\in\mathcal{N}} \exp(\overrightarrow{u_i} \cdot \overrightarrow{u_k}^T)}. \tag{1}$$

Skip-Gram is to derive the set of the N embedding vectors that maximizes the log probability of observing every neighboring node from $N_{SI}(i)$ for each i. Therefore, one maximizes

$$\max \mathcal{O} = \sum_{i\in\mathcal{N}} \sum_{j\in N_{SI}(i)} \log p(j|i). \tag{2}$$

Equation (2) can be further simplified to

$$\max \mathcal{O} = \sum_{i\in\mathcal{N}} \left(-\log Z_i + \sum_{j\in N_{SI}(i)} \overrightarrow{u_i} \cdot \overrightarrow{u_k}^T \right), \tag{3}$$

where

$$Z_i = \sum_{k\in\mathcal{N}} \exp\left(\overrightarrow{u_i} \cdot \overrightarrow{u_k}^T\right). \tag{4}$$

To compute Z_i for a given i, we need to traverse the entire node set $\mathcal{N}$, which is computationally costly. To solve this problem, we introduce negative sampling [27], which randomly selects a certain number of nodes k from $\mathcal{N}$ to approximate Z_i. To get the embedding vectors for each node, we use the stochastic gradient ascent to optimize Eq. (3).

We name this static SI-spreading-based network embedding algorithm that uses Skip-Gram model as *SINE*.

2.3 SI-spreading-based temporal network sampling

We generalize *SINE* to the SI-spreading-based temporal network embedding by deploying SI spreading processes on the given temporal network, namely, *TSINE*. For a temporal network $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, SI spreading follows the time step of the contacts in $\mathcal{G}$. Initially, node i is chosen as the seed of the spreading process. At every time step $t \in [0, T]$, an infected node infects each of its susceptible neighbor in the snapshot through the contact between them with probability β. The process stops at time T. We construct the spreading trajectory starting from node i as $\mathcal{T}_i(\beta)$, which records the union of nodes that get infected together with the contacts through which these nodes get infected. We propose two protocols to select the seed node of the SI spreading. In the first protocol, we start by selecting uniformly at random a node i as the seed. Then, we select uniformly at random a time step from all the times of contacts made by node i as the starting point of the spreading process, i.e., the time when i gets initially infected. We refer to this protocol as *TSINE1*. In the second protocol, we choose a node i uniformly at random as the seed and start the spreading at the time when node i has the first contact. We refer to this protocol as *TSINE2*.

Both *TSINE1* and *TSINE2* generate the node pair set from the spreading trajectory $\mathcal{T}_i(\beta)$ in the same way as described in Sect. 2.1. The node pair set is the input of the Skip-Gram for calculating the embedding vectors. The SI-spreading-based temporal network embedding uses the information on the time stamps of contacts in addition to the information used by the static network embedding.

3 Results

For the link prediction task in a static network, we remove a certain fraction of links from the given network and predict these missing links based on the remaining links. We apply our static network embedding algorithm to the remaining static network to derive the embedding vectors for the nodes, which are used for link prediction. For a temporal network, we select a fraction of node pairs that have at least one contact. We remove all the contacts between the selected node pairs from the given temporal network. Then, we attempt to predict whether the selected node pairs have at least one contact or not based on the remaining temporal network. We confine ourselves to the prediction of whether the selected node pairs have contact(s) or not, instead of predicting the number of contacts and their associated time stamps between each selected node pair. We use the area under the curve (AUC) score to evaluate the performance of the algorithms on the link prediction task. The AUC quantifies the probability of ranking a random node pair that is connected or has at least a contact higher than a random node pair that is not connected or has no contact.

3.1 Empirical networks

We consider temporal networks, each of which records the contacts and their corresponding time stamps between every node pair. For each temporal network $\mathcal{G}$, one can obtain the corresponding static network G by aggregating the contacts between each node pair over time. In other words, two nodes are connected in static network G if there is at least one contact between them in $\mathcal{G}$. The static network G derived from $\mathcal{G}$ is unweighted by definition. We consider the following temporal social network data sets.

- *HT2009* [35] is a network of face-to-face contacts between the attendees of the ACM Hypertext 2009 conference.
- *Manufacturing Email (ME)* [36] is an email contact network between employees in a mid-sized manufacturing company.
- *Haggle* [37] records the physical contacts between individuals via wireless devices.
- *Fb-forum* [38] captures the contacts between students at University of Califonia, Irvine, in a Facebook-like online forum.
- *DNC* [39] is an email contact network in the 2016 Democratic National Committee email leak.
- *CollegeMsg* [40] records messages between the users of an online community of students from the University of California, Irvine.

Table 1 provides some properties of the empirical temporal networks. In the first three columns we show the properties of the temporal networks, i.e., the number of nodes (N), timestamps (T) and contacts ($|\mathcal{L}|$). In the remaining columns, we show the properties of the corresponding aggregate static networks, including the number of links ($|E|$), link density, average degree, and clustering coefficient. The temporal networks are considerably different in size, which ranges from hundreds to thousands of nodes, as well as in the network density and clustering coefficient. Choosing networks with different properties allows us to investigate whether the performance of our algorithms is consistent across networks.

3.2 Baseline algorithms

The performance of network embedding algorithms depends on the sampling process to obtain the (node pair) input data as well as the learning model. In this work, we aim to

Table 1 Properties of the empirical temporal networks. The number of nodes (N), timestamps (T), and contacts ($|\mathcal{L}|$) are shown. In addition, the number of links ($|E|$), link density, average degree, and clustering coefficient of the corresponding static network are shown

Dataset	N	T	$\lvert\mathcal{L}\rvert$	$\lvert E\rvert$	Link density	Average degree	Clustering coefficient
HT2009	113	5246	20,818	2196	0.35	38.87	0.53
ME	167	57,842	82,927	3251	0.23	38.93	0.59
Haggle	274	15,662	28,244	2124	0.57	15.50	0.63
Fb-forum	899	33,515	33,720	7046	0.02	15.68	0.06
DNC	1891	19,383	39,264	4465	0.002	4.72	0.21
CollegeMsg	1899	58,911	59,835	13,838	0.01	14.57	0.11

understand the effect of the sampling process that we proposed on the performance of an embedding algorithm. Hence, we choose as baseline three state-of-the-art network embedding algorithms (*DeepWalk*, *Node2Vec* and *CTDNE*) that share the same Skip-Gram learning model as our algorithms. These baseline algorithms and the algorithms that we proposed differ only in the method to sample trajectory paths and the node pair set, which is used as the input of the Skip-Gram. *DeepWalk* [25] and *Node2Vec* [28] are static network embedding algorithms based on random walks. *CTDNE* [31] is a temporal network embedding algorithm based on random walks. We also choose *tNodeEmbed* [41] as a baseline model, which uses a different learning model from the above models. It learns temporal network embedding by combining static node embeddings learned from *Node2Vec* with Recurrent Neural Networks. Via this baseline, we would like to explore whether the performance improvement via the design of learning model is significantly higher than that via the design of the sampling process.

- *DeepWalk* [25] deploys classic random walks on a given static network.
- *Node2Vec* [28] deploys biased random walks on a given static network. The biased random walk gives a trade-off between breadth-first-like sampling and depth-first-like sampling of the neighborhood, which is controlled via two hyper-parameters p and q. We use a grid search over $p, q \in \{0.01, 0.25, 0.5, 1, 2, 4\}$ to obtain embeddings that achieve the largest AUC value for link prediction.
- *CTDNE* [31]: *CTDNE* is a temporal network embedding algorithm based on temporal random walks. The main idea is that the timestamp of the next temporal contact on the walk should be larger than the timestamps of previously traversed contacts. Given a temporal network $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, the starting contact for the temporal random walk is selected uniformly at random. Thus, every contact has probability $1/|\mathcal{L}|$ to be selected as the starting contact. Assume that a random walker visits node i at time step t. We define $\Gamma_t(i)$ as the set of nodes that have contacted node i after time t allowing duplicated elements. A node may appear multiple times in $\Gamma_t(i)$ because it may have multiple contacts with node i over the course of time. The next node to walk to is uniformly selected from $\Gamma_t(i)$, i.e., every node in $\Gamma_t(i)$ is chosen with probability $1/|\Gamma_t(i)|$. Nguyen et al. [31] generalized the starting contact and the successor node of a temporal walk to other distributions beyond the uniform distribution illustrated here. When we compare the performance of the algorithms on link prediction, we explore the embeddings that give the largest AUC value for link prediction of *CTDNE* by taking into account all possible generalizations proposed by Nguyen et al.
- *tNodeEmbed* [41]: The objective of *tNodeEmbed* is to preserve the static network neighborhood for each node and the stability of embeddings over time. Given a temporal network observed within $[1, T]$, a set of $T_1 \leq T$ coarse-grained network

snapshots can be obtained, where each snapshot is the temporal network aggregated over the time window of the snapshot. In each snapshot network, an embedding vector can be obtained for each node using *Node2Vec*. Recurrent Neural Networks (RNN) is further employed among the embeddings at different time snapshots to obtain a final d-dimensional embedding vector for each node. For each dataset, we tune hyper-parameters such as the number of snapshots and learning rate to obtain the largest AUC value. The number of snapshots for obtaining optimal AUC for *HT2009*, *ME*, *Haggle*, *Fb-forum*, *DNC* and *CollegeMsg* are 22, 24, 27, 29, 42 and 34, respectively.

In our SI-spreading-based algorithms for both static and temporal networks, we set $\beta \in \{0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. We use window size $\omega = 10$ and embedding dimension $d = 128$ for our algorithms and the baseline algorithms.

3.3 Performance evaluation

3.3.1 Training and test sets

In this section, we illustrate how to generate the training and test sets in the link prediction task in temporal and static networks. We run the network embedding algorithms on the corresponding training set and obtain embedding vector for each node, and use AUC to evaluate link prediction performance in the test set.

Given a temporal network $\mathcal{G}$, we select uniformly at random 75% node pairs among the node pairs that have at least one contact between them in $\mathcal{G}$ as the training set for temporal embedding algorithms, including all the contacts and their timestamps. The training set for static network embedding algorithms is the aggregation of the training set for temporal embedding algorithms. In other words, for every node pair, there is a link between the two nodes in the training set for static network embedding if and only if they have at least one contact in the training set for temporal embedding algorithms.

We use the remaining 25% node pairs among the node pairs that have at least one contact of $\mathcal{G}$ as the positive links in the test set. We label these node pairs 1. Then, we uniformly randomly sample an equal number of node pairs in $\mathcal{G}$ which have no contact between them. These node pairs are used as negative links in the test set, which we label 0. The same test set is used for the link prediction task in both temporal and static networks.

For each temporal network data set, we randomly split the network to obtain the training and test set according to the procedures given above five times. Both random walks and SI spreading processes are stochastic. For each split data, we run each algorithm on the training set and perform the link prediction on the test set for ten realizations. Therefore, we obtain ten AUC scores for each splitting of the data into the training and test sets, evening the randomness stemming from stochasticity of the random walk or SI spreading processes. We obtain the AUC score for each algorithm with a given parameter set as an average over 50 realizations in total.

3.3.2 Evaluation results

We summarize the overall performance of the algorithms on missing link prediction in Table 2. For each algorithm, we tune the parameters and show the optimal average AUC score.

First of all, *tNodeEmbed* with its advanced design like RNN in the learning model performs worse than the algorithms we proposed in 3 out of the 6 networks. This illustrates

Table 2 AUC scores for link prediction. All the results shown are the average over 50 realizations. Bold indicates the optimal AUC among the embedding algorithms, * indicates the optimal AUC among all the algorithms. L2, L3, L4 are the short for link prediction metrics which counts the number of $l = 2, 3, 4$ paths, respectively

Dataset	DeepWalk	Node2Vec	CTDNE	tNodeEmbed	TSINE1	TSINE2	SINE	L2	L3	L4
HT2009	0.5209	0.5572	0.6038	0.5358	0.6740	**0.6819**	0.6726	0.7069*	0.7066	0.7055
ME	0.6439	0.6619	0.6575	0.7281	0.7329	0.7462	**0.7744**	0.7855	0.7878*	0.7790
Haggle	0.3823	0.7807	0.7796	**0.8702***	0.8051	0.8151	0.8267	0.8167	0.8255	0.8226
Fb-forum	0.5392	0.6882	0.6942	0.6013	0.7104	0.7195	**0.7302***	0.5606	0.7179	0.7203
DNC	0.5822	0.5933	0.7274	**0.9105***	0.7539	0.7529	0.7642	0.7704	0.7627	0.7193
CollegeMsg	0.5356	0.5454	0.7872	**0.8724***	0.8257	0.8321	0.8368	0.7176	0.8609	0.8203

the importance of exploring the design of the sampling process in embedding algorithms, not only of the learning model. From now on, we will focus on embedding algorithms based on Skip-Gram, but not *tNodeEmbed* any more, to investigate the influence of the sampling process on the performance of the embedding algorithms.

Among the static network embedding algorithms, *SINE* significantly outperforms *DeepWalk* and *Node2Vec*. The improvement in the AUC score is up to 30% on the *CollegeMsg* dataset. Additionally, we have tested the static embedding algorithms on another two large static networks. The network description and the embedding results are shown in Tables S1 and S2 in the Supporting Information 1. The out-performance of *SINE* in such large networks is even more evident. Embedding algorithms *CTDNE*, *TSINE1* and *TSINE2* are for temporal networks. The SI-spreading-based algorithms (i.e., *TSINE1* and *TSINE2*) also show better performance than random-walk-based one (*CTDNE*). Additionally, *TSINE2* is slightly better than *TSINE1* on all data sets. Therefore, we will focus on *TSINE2* in the following analysis. In fact, *SINE* shows better performance than temporal network embedding methods including *TSINE2* on all data sets except for *HT2009*. It has been shown that temporal information is important for learning embeddings [31, 42, 43]. However, up to our numerical efforts, *SINE* outperforms the temporal network algorithms although SINE deliberately neglects temporal information.

To explore further the difference in performance among the embedding algorithms, we investigate the distribution of the dot product of node embedding vectors. Given a link (i, j) in the test set, we compute the dot product of the two end nodes' embedding vectors, i.e., $\overrightarrow{u_i} \cdot \overrightarrow{u_j}^T$. We show the dot product distribution for the positive links and negative links in the test set separately. For each embedding algorithm, we consider only the parameter set that maximizes the AUC, i.e., the parameter values with which the results are shown in Table 2. We show the distribution of the dot product for *Haggle* in Fig. 2 and for the other data sets in Figure S1–S5 in the Supporting Information 1. Compared to the random-walk-based algorithms, *TSINE2* and *SINE* yield more distinguishable distributions between the positive (grey) and the negative links (pink). This result supports the better performance of SI-spreading-based embeddings than random-walk-based ones.

The embedding algorithms differ only in the sampling method to generate the node pair set. These algorithms use the same Skip-Gram architecture, which takes the node pair set as input, to deduce the embedding vector for each node. We explore further how the algorithms differ in the node pair sets that they sampled. The objective is to discover the relation between the properties of the sampled node pairs and the performance of an embedding method. We represent the node pair set generated by an embedding method as a network $G_S = (\mathcal{N}, E_S)$, so called the sampled network. Two nodes are connected in G_S if

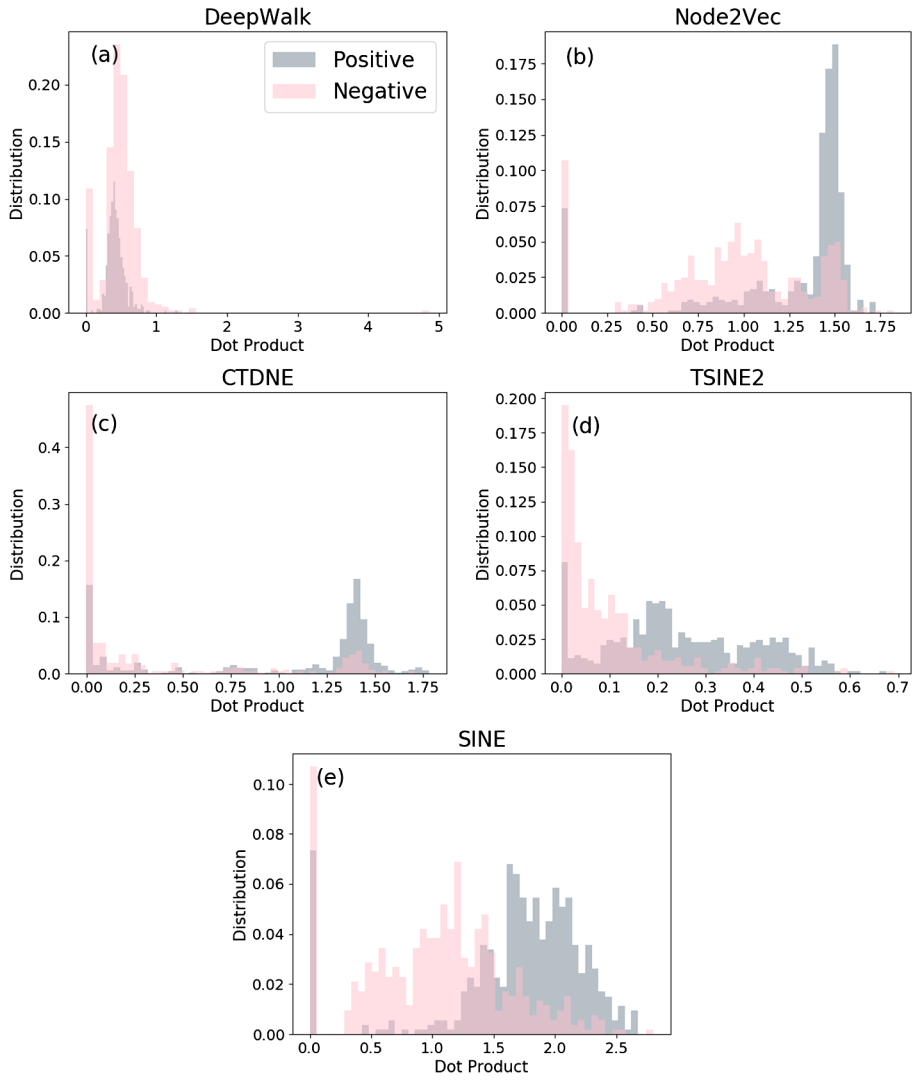

Figure 2 The dot product distribution of the two end nodes' embedding vectors of the positive and negative links in the test set. We show the result of the *Haggle* data set. For each algorithm, we use the same parameter settings as that of Table 2 to obtain the embeddings. Dot products of positive links are shown in grey. Negative links are shown in pink. The results are shown for algorithms (**a**) *DeepWalk*; (**b**) *Node2Vec*; (**c**) *CTDNE*; (**d**) *TSINE2* and (**e**) *SINE*

they form a node pair in the node pair set. We note that G_S is an unweighted network. For each algorithm, with the parameter set that maximizes the AUC, we show the cumulative degree distribution of its sampled network G_S in Fig. 3. The cumulative degree distribution of the training set for static network is also given. Compared to the cumulative degree distribution of the training set, the sampled networks tend to have a higher node degree. Zhang et al. and Gao et al. [30, 44] have shown that when the degree distribution of G_S is closer to that of the training set, the prediction performance of a random-walk-based algorithm tends to be better. Even though SI-spreading based algorithms perform the best across most of the data sets, we have not found a direct relation between the performance of the embedding algorithm and similarity between the degree distribution of the sampled network and that of the training set.

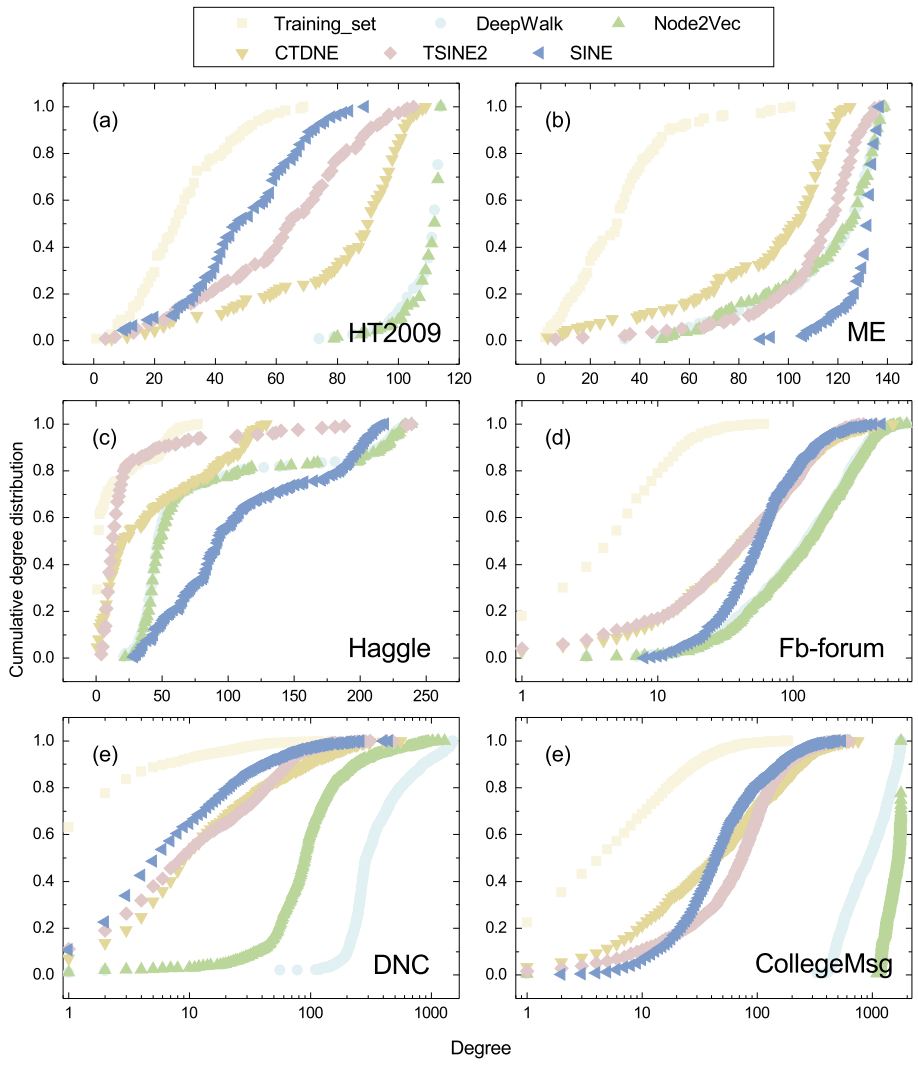

Figure 3 Cumulative degree distribution of the static network derived from the training set and that of the sampled networks G_S from different algorithms. We show the results for (**a**) *HT2009*; (**b**) *ME*; (**c**) *Haggle*; (**d**) *Fb-forum*; (**e**) *DNC*; (**f**) *CollegeMsg*

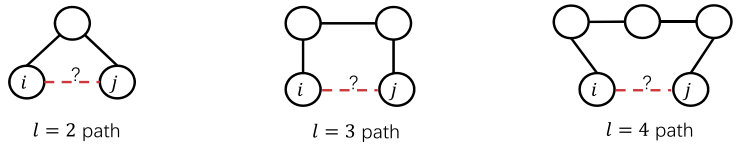

Figure 4 Illustration of l paths between a pair of nodes i and j. Here we show $l = 2, 3, 4$

Similarity-based methods such as the number of $l = 2, 3, 4$ paths have been used for link prediction problem [10]. An l path between two nodes refers to a path that contains l links. We show examples of $l = 2, 3, 4$ path between a node pair i and j in Fig. 4. Kovács et al. [45] have shown that l paths ($l = 3, 4$) outperform existing link prediction methods in predicting protein interaction. Cao et al. [46] found that network embedding algorithms

based on random walks sometimes perform worse in link prediction than the number of $l = 2$ paths or equivalently the number of common neighbors. This result suggests a limit of random-walk-based embedding in identifying the links between node pairs that have many common neighbors. Therefore, we explore further whether our SI-spreading-based algorithms can overcome this limitation, thus possibly explain their outperformance.

We investigate what kind of network structure surrounding links makes them more easily be predicted. For every positive link in the test set, we study its two end nodes' topological properties (i.e., the number of $l = 2$, $l = 3$ and $l = 4$ paths) and the dot product of the embedding vectors of its two end nodes. Given a network, the parameters of each embedding algorithm are tuned to maximize the AUC, as given in Table 2. We take data set *Haggle* as an example. Figure 5 shows the relation between the dot product of the embedding vectors and the number of $l = 2, 3, 4$ paths of the two end nodes of a positive link in the test set for all the embedding methods. The Pearson correlation coefficient (PCC) between the two variables for all the networks and algorithms is given in Table S3 in the Supporting Information 1. Figure 5 and Table S3 together show that the dot product of the embedding vectors constructed from *TSINE2* and *SINE* is more strongly correlated with the number of l paths, where $l = 2, 3$ or 4, than the random-walk-based embeddings. This result suggests that SI-spreading-based algorithms may better predict the links whose two end nodes have many l-paths, thus overcoming the limit of random-walk-based embedding algorithms.

The number of $l = 2, 3$ paths has been used to predict links in [10, 45, 46]. The observation and the limit of random-walk-based embedding algorithms motivate us to use the number of $l = 2, 3, 4$ paths between a node pair to predict missing links. Take $l = 2$ paths as an example. For every link in the test set, the number of $l = 2$ paths between the two end nodes in the training set is used to estimate the likelihood of connection between them. In the networks we considered, two end nodes of a link tend to be connected by $l = 2$, $l = 3$ and $l = 4$ paths (see Figs. 5). Table 2 ($L2$, $L3$, $L4$ shown in the table correspond to the method of using the number of $l = 2, 3, 4$ path for link prediction) shows that in such networks, the similarity-based methods do not evidently outperform the SI-spreading-based embedding. Actually, the SI-spreading-based embedding performs better in two out of six networks.

Next, we study the effect of the sampling size, B, on the performance of each algorithm. The sampling size is quantified as the total length of the trajectory paths as defined in Sect. 2.1. Given a network, we set $B = NX$, where N is the size of the network and $X \in \{1, 2, 5, 10, 25, 50, 100, 150\}$. We evaluate our SI-spreading-based embedding algorithms *SINE* and *TSINE2*, and one random-walk-based embedding algorithm *CTDNE*, because *CTDNE* performs mostly the best among all random-walk-based algorithms. The result is shown in Fig. 6. For each X, we tune the other parameters to show the optimal AUC in the figure. Both *SINE* and *TSINE2* perform better than *CTDNE* and are relatively insensitive to the sampling size. This means that they achieve a good performance even when the sampling size is small, even with $X = 1$. This is because the node pair set sampled remains relatively the same when X varies. We give the overlap between node pairs set sampled by different X of *HT*2009 as an explanation in Figures S6–S7 in the Supporting Information 1. The random-walk-based algorithm, *CTDNE*, however, requires a relatively large sampling size to achieve a comparable performance with *SINE* and *TSINE2*.

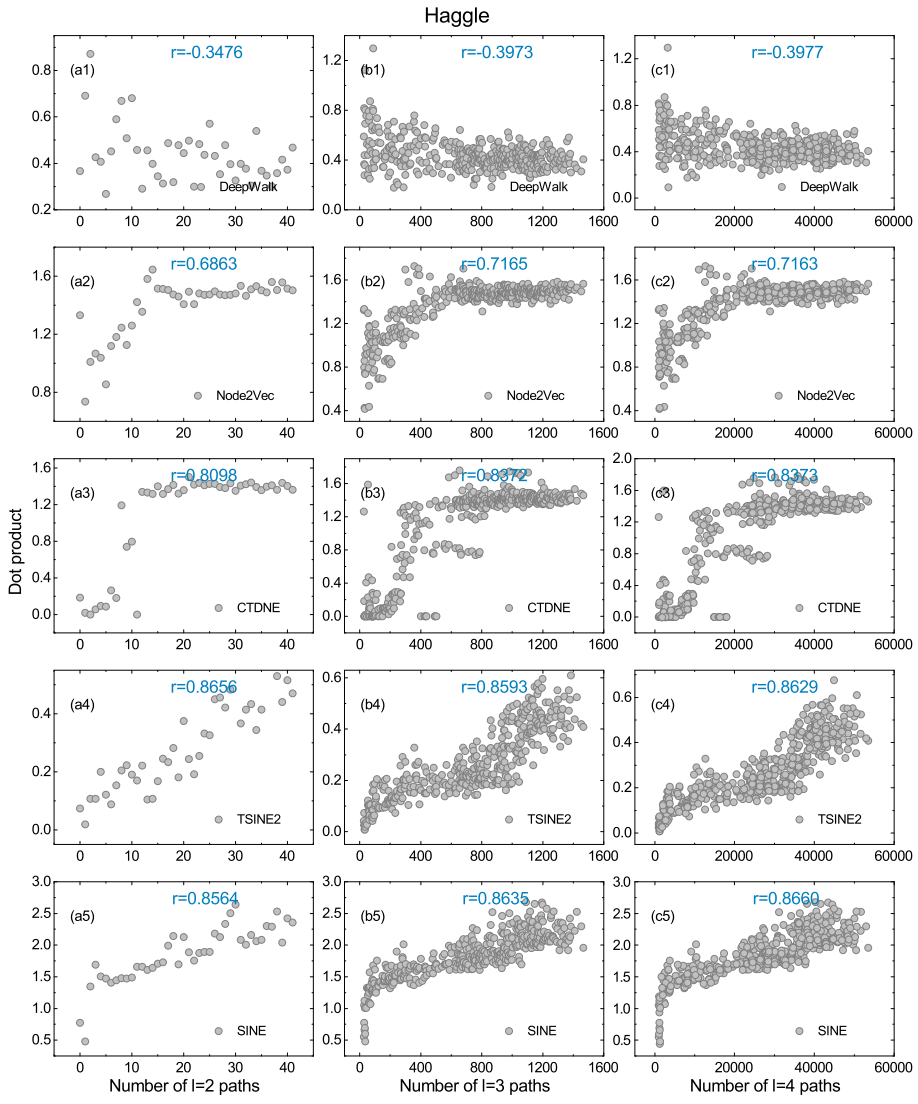

Figure 5 Relation between the dot product of the two end nodes' embedding vectors and the number of $l = 2, 3, 4$ paths between the two end nodes of the positive links in the test set for *Haggle* data set. (**a1**–**a5**), (**b1**–**b5**) and (**c1**–**c5**) are the results for the number of $l = 2, 3, 4$ paths, respectively

The AUC as a function of the infection probability β is shown in Fig. 7. For each β, we tune the other parameters to show the optimal AUC. The SI-spreading-based algorithms achieve high performance with a small infection probability ($0.001 \leq \beta \leq 0.1$) for all the data sets. The high performance of SI-spreading-based embedding algorithms with the small value of X and β across different networks motivates the further study whether one can optimize the performance by searching a smaller range of the parameter values. Finally, we test how the AUC values change with the embedding dimension d for embedding algorithm *SINE* [47]. The results are shown in Fig. 8. For each of the datasets, we only change the value of d, the other parameters are set the same as these of Table 2. We choose $d \in \{2, 4, 8, 16, 32, 64, 128, 256\}$. With the increase of d, the AUC value firstly slightly increases and then stays stable for all the network datasets. Figure 8 shows that $d = 128$ as the default value is sufficient to produce high AUC values for missing link pre-

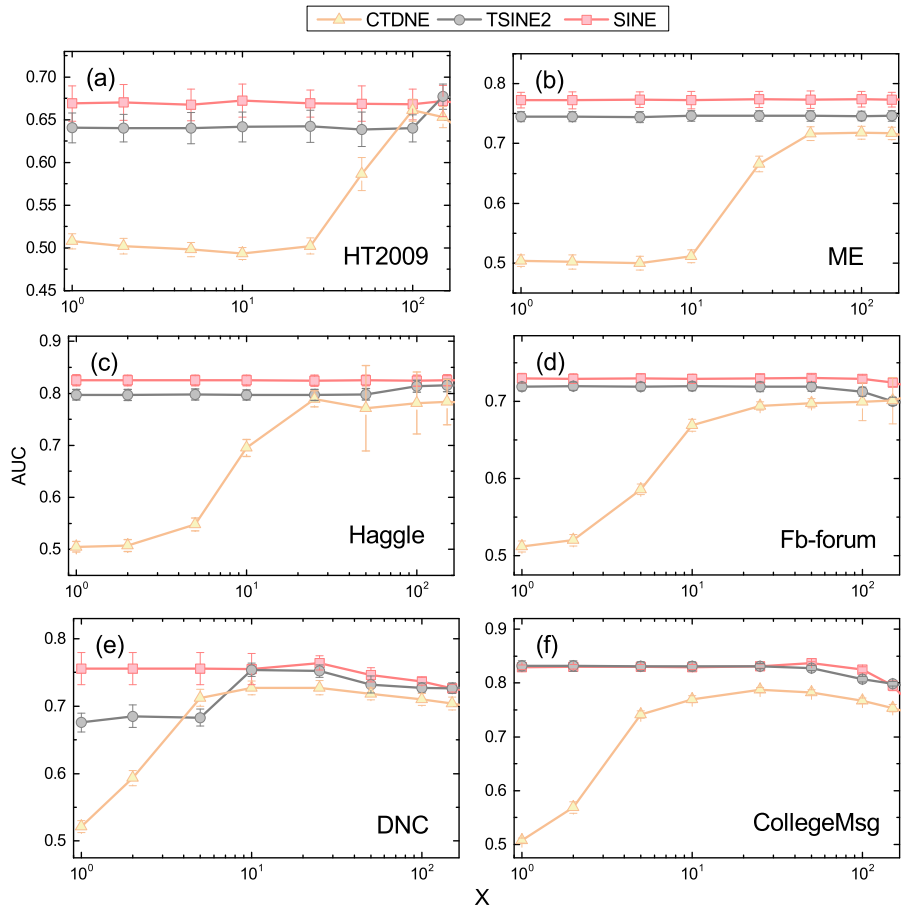

Figure 6 Influence of the sampling size $B = NX$ on the link prediction performance, i.e., AUC score. The error bar shows the standard deviation of the AUC score calculated on the basis of 50 realizations. We show the results for (**a**) *HT2009*; (**b**) *ME*; (**c**) *Haggle*; (**d**) *Fb-forum*; (**e**) *DNC*; (**f**) *CollegeMsg*

diction. Figure 8 also shows the possibility that a similar performance can be achieved by a smaller dimension for the networks we have considered.

The out-performance and efficiency of the algorithms we proposed lie in SI spreading process which samples the information that can not be captured by random-walk-based sampling process. We take *SINE* as an example. Each trajectory path generated by the SI-spreading-based sampling method doesn't have any repeat node on the path. All the nodes on the path are thus unique. The biased random-walk sampling method used in *Node2Vec* can practically interpolate between depth-first-sampling and breadth-first-sampling via tuning the parameters. Still, the walks generated, even tuned towards the DFS may contain repeat nodes. Consider the SI spreading trajectory obtained from a random seed node i on a given static network with a given infection probability β. Links in the static network close to the seed node (root) are more likely to be included in the spreading trajectory. When $\beta = 1$, all links incident to the seed will be included in the trajectory. Links incident to the neighbors of the seed will be included in the trajectory if they link to new nodes but not the seed nor the neighbors of the seed. In this case, the trajectory path between the seed node and a uniformly randomly selected node is the shortest path in the static network. When β is small, the trajectory tends to be large in depth (the largest hopcount between

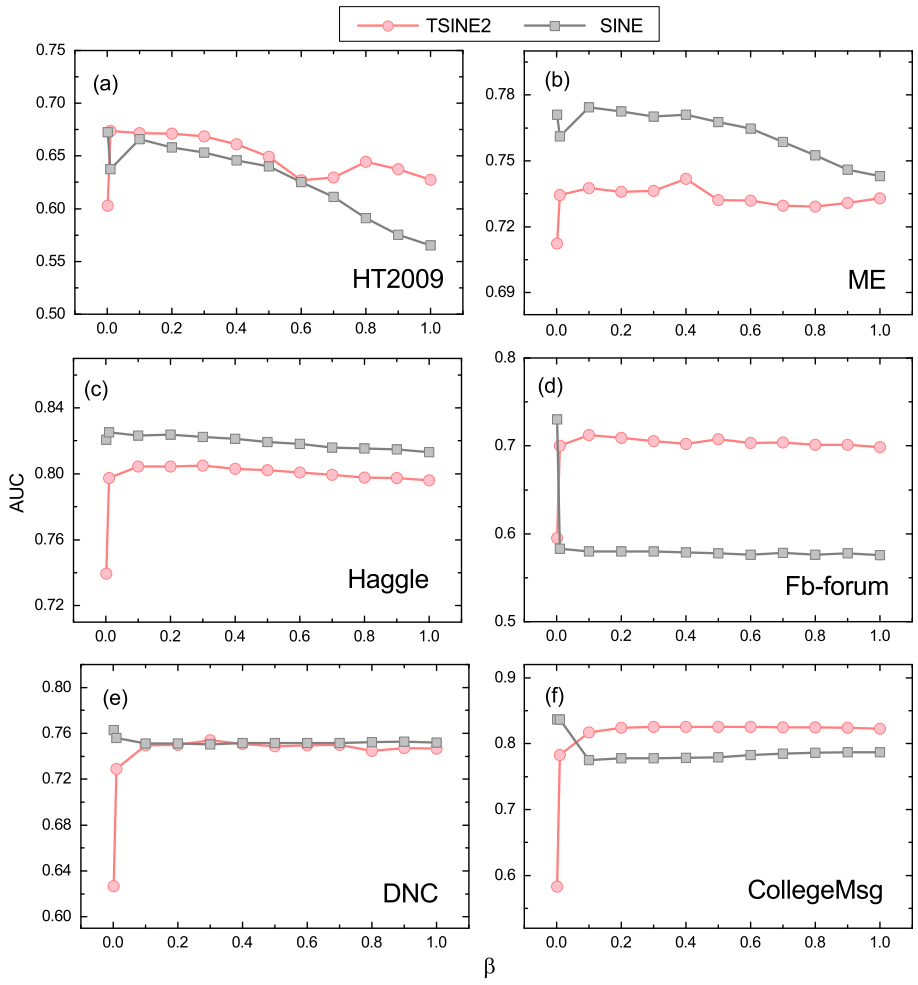

Figure 7 AUC as a function of β for SI-spreading-based network embedding algorithms. We show the results for (**a**) *HT2009*; (**b**) *ME*; (**c**) *Haggle*; (**d**) *Fb-forum*; (**e**) *DNC*; (**f**) *CollegeMsg*

the seed and any other node) and the trajectory path tends to be longer, as shown in Figure S6 in the Supporting Information 1. The links around high degree nodes tend to appear in a trajectory tree. The infection probability balances the preference of links around high degree nodes and links close to the seed to appear in the trajectories thus trajectory paths. Such kind of sampling over the neighborhood of the seed and the links of high degree nodes without repeat node in trajectory paths cannot be captured by random walk type sampling.

The optimal performance is obtained when β is small. As shown in Figure S6 in the Supporting Information 1, the average length of a trajectory path is around 3.5 for *HT2009* and *ME*, which is far smaller than the length of walks derived from the random-walk-sampling strategy. Such a small length of trajectory paths without redundant nodes in each path, and the tendency of links around high degree nodes to appear in a trajectory paths may explain why a small sampling size (the total number of nodes in the trajectory paths) is sufficient to capture the essential information thus obtain high performance.

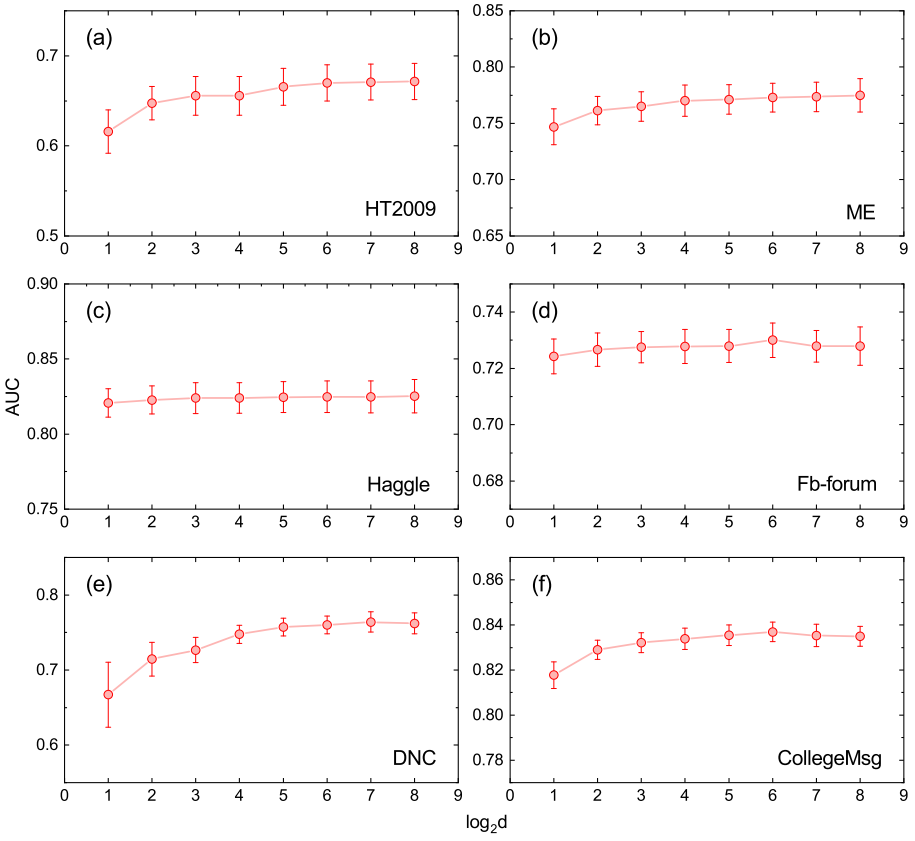

Figure 8 AUC as a function of embedding dimension d of embedding algorithm *SINE*. We show the results for (**a**) *HT2009*; (**b**) *ME*; (**c**) *Haggle*; (**d**) *Fb-forum*; (**e**) *DNC*; (**f**) *CollegeMsg*

4 Conclusions

In this paper, we proposed network embedding algorithms based on SI spreading processes in contrast to the previously proposed embedding algorithms based on random walks [48, 49]. We further evaluated the embedding algorithms on the missing link prediction task. The key point of an embedding algorithm is how to design a strategy to sample trajectories to obtain embedding vectors for nodes. We used the SI model to this end. The algorithms that we proposed are *SINE* and *TSINE*, which use static and temporal networks, respectively.

On six empirical data sets, the SI-spreading-based network embedding algorithm on the static network, i.e., *SINE*, gains much more improvement than state-of-the-art random-walk-based network embedding algorithms across all the data sets. The SI-spreading-based network embedding algorithms on the temporal network, *TSINE1* and *TSINE2*, also show better performance than the temporal random-walk-based algorithm. Temporal information provides additional information that may be useful for constructing embedding vectors [31, 42, 43]. However, we find that *SINE* outperforms *TSINE*, which uses timestamps of the contacts. This result suggests that temporal information does not necessarily improve the embedding for missing link prediction. Moreover, when the sampling size of the Skip-Gram is small, the performance of the SI-spreading-based embedding algorithms is still high. Sampling trajectory paths takes time especially for large-scale networks. Therefore, our observation that the SI-spreading-based algorithms require less

samples than other algorithms promises the applicability of the SI-spreading-based algorithms to larger networks than the random-walk-based algorithms. Finally, we show insights of why SI-spreading-based embedding algorithms performs the best by investigating what kind of links are likely to be predicted.

We deem the following future work as important. We have already applied susceptible-infected-susceptible (SIS) model and evaluated the SIS-spreading-based embedding. However, this generalization has not improved the performance in the link prediction task. Therefore, one may explore whether or not sampling the network information via the other spreading processes, such as susceptible-infected-recovered (SIR) model, further improves the embedding. It is also interesting to explore further the performance of the SI-spreading-based algorithms in other tasks such as classification and visualization. Moreover, the SI-spreading-based sampling strategies can also be generalized to other types of networks, e.g., directed networks, signed networks, and multilayer networks.

Supplementary information

Supplementary information accompanies this paper at https://doi.org/10.1140/epjds/s13688-020-00248-5.

Additional file 1. Supplementary information (PDF 601 kB)

Acknowledgements
We thank the SocioPatterns collaboration (http://www.sociopatterns.org) for providing the data sets.

Funding
XZ is supported by the China Scholarship Council (CSC). NM acknowledges support from AFOSR European Office (under Grant No. FA9550-19-1-7024). PH was supported by JSPS KAKENHI Grant Number JP 18H01655 and by the Grant for Basic Science Research Projects by the Sumitomo Foundation. HW would like to thank Netherlands Organisation for Scientific Research NWO (TOP Grant no. 612.001.802).

Abbreviations
SI, susceptible-Infected; SINE, Susceptible-Infected network embedding on static networks; TSINE, Susceptible-Infected network embedding on temporal networks; LLE, locally linear embedding; SVD, Singular value decomposition; AUC, area under the curve; CTDNE, Continuous-Time Dynamic Network Embeddings; tNodeEmbed, Node Embedding over Temporal Graphs; BFS, Breadth-first Sampling; DFS, Depth-first Sampling; PCC, Pearson correlation coefficient; SIS, susceptible-infected-susceptible; SIR, susceptible-infected-recovered.

Availability of data and materials
We use open data which can be downloaded on http://www.sociopatterns.org and https://snap.stanford.edu/data/. The source code will be available from the first author based on reasonable request.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
All authors planed the study; XZ and ZL performed the experiments and prepared the figures. All authors analyzed the results and wrote the manuscript. All authors read and approved the final manuscript.

Author details
[1]Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands. [2]Department of Mathematics, University at Buffalo, State University of New York, Buffalo, NY 14260-2900, New York, USA. [3]Computational and Data-Enabled Science and Engineering Program, University at Buffalo, State University of New York, Buffalo, NY 14260-2900, New York, USA. [4]Tokyo Tech World Research Hub Initiative (WRHI), Institute of Innovative Research, Tokyo Institute of Technology, Yokohama 226-8503, Japan.

Endnotes
[a] The abbreviation *SINE* of our *S*usceptible-*I*nfected *N*etwork *E*mbedding on static networks should be distinguished from *SiNE*, a signed network embedding model proposed in [50].

[b] Real-world temporal networks are measured or sampled at discrete time steps with possibly different sampling frequencies thus different durations of a time step, limited by the measurement technique as well as the targeting research question. Moreover, a contact may actually last less than a unit time step, which is not observable due to the limited sampling frequency. These two factors motivate us to consider a wide range of the infection probability

per unit time step $\beta \in \{0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ in the SI-spreading-based embedding algorithms to get the optimal embedding vector for every node.

Publisher's Note

Received: 15 April 2020 Accepted: 25 September 2020 Published online: 16 October 2020

References

1. Newman ME (2003) The structure and function of complex networks. SIAM Rev 45(2):167–256
2. Zhang Z-K, Liu C, Zhan X-X, Lu X, Zhang C-X, Zhang Y-C (2016) Dynamics of information diffusion and its applications on complex networks. Phys Rep 651:1–34
3. Costa LdF, Oliveira ON Jr, Travieso G, Rodrigues FA, Villas Boas PR, Antiqueira L, Viana MP, Correa Rocha LE (2011) Analyzing and modeling real-world phenomena with complex networks: a survey of applications. Adv Phys 60(3):329–412
4. Qi Y, Bar-Joseph Z, Klein-Seetharaman J (2006) Evaluation of different biological data and computational classification methods for use in protein interaction prediction. Proteins, Struct Funct Bioinform 63(3):490–500
5. Girvan M, Newman ME (2002) Community structure in social and biological networks. Proc Natl Acad Sci USA 99(12):7821–7826
6. Jacob Y, Denoyer L, Gallinari P (2014) Learning latent representations of nodes for classifying in heterogeneous social networks. In: Proceedings of the 7th ACM international conference on web search and data mining. ACM, New York, pp 373–382
7. Traud AL, Mucha PJ, Porter MA (2012) Social structure of Facebook networks. Phys A, Stat Mech Appl 391(16):4165–4180
8. Wang H, Li Q, D'Agostino Gregorio, Havlin S, Stanley HE, Van Mieghem P (2013) Effect of the interconnected network structure on the epidemic threshold. Phys Rev E 88(2):022801
9. Liben-Nowell D, Kleinberg J (2007) The link-prediction problem for social networks. J Am Soc Inf Sci Technol 58(7):1019–1031
10. Lü L, Zhou T (2011) Link prediction in complex networks: a survey. Phys A, Stat Mech Appl 390(6):1150–1170
11. Lü L, Medo M, Yeung CH, Zhang Y-C, Zhang Z-K, Zhou T (2012) Recommender systems. Phys Rep 519(1):1–49
12. Martínez V, Berzal F, Cubero J-C (2017) A survey of link prediction in complex networks. ACM Comput Surv 49(4):69
13. Liu C, Ma Y, Zhao J, Nussinov R, Zhang Y-C, Cheng F, Zhang Z-K (2020) Computational network biology: data, model, and applications. Phys Rep 846:1–66
14. Getoor L, Diehl CP (2005) Link mining: a survey. ACM SIGKDD Explor Newsl 7(2):3–12
15. Cui P, Wang X, Pei J, Zhu W (2019) A survey on network embedding. IEEE Trans Knowl Data Eng 31(5):833–852
16. Wang X, Cui P, Wang J, Pei J, Zhu W, Yang S (2017) Community preserving network embedding. In: Thirty-first AAAI conference on artificial intelligence
17. Pandhre S, Mittal H, Gupta M, Balasubramanian VN (2018) Stwalk: learning trajectory representations in temporal graphs. In: Proceedings of the ACM India joint international conference on data science and management of data, pp 210–219
18. Béres F, Kelen DM, Pálovics R, Benczúr AA (2019) Node embeddings in dynamic graphs. Appl Netw Sci 4(1):64
19. Sato K, Oka M, Barrat A, Cattuto C (2019) Dyane: dynamics-aware node embedding for temporal networks. arXiv preprint. arXiv:1909.05976
20. Torricelli M, Karsai M, Gauvin L (2020) weg2vec: event embedding for temporal networks. Sci Rep 10(1):1–11
21. Tenenbaum JB, De Silva V, Langford JC (2000) A global geometric framework for nonlinear dimensionality reduction. Science 290(5500):2319–2323
22. Roweis ST, Saul LK (2000) Nonlinear dimensionality reduction by locally linear embedding. Science 290(5500):2323–2326
23. Belkin M, Niyogi P (2002) Laplacian eigenmaps and spectral techniques for embedding and clustering. In: Advances in neural information processing systems, pp 585–591
24. Golub GH, Reinsch C (1971) Singular value decomposition and least squares solutions pp 134–151
25. Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, pp 701–710
26. Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q (2015) Line: large-scale information network embedding. In: Proceedings of the 24th international conference on world wide web, pp 1067–1077. International World Wide Web Conferences Steering Committee
27. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems, pp 3111–3119
28. Grover A, Leskovec J (2016) node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, pp 855–864
29. Cao Z, Wang L, de Melo G (2018) Link prediction via subgraph embedding-based convex matrix completion. In: Proceedings of the 32nd AAAI conference on artificial intelligence (AAAI 2018). AAAI Press, Menlo Park
30. Zhang Y, Shi Z, Feng D, Zhan X-X (2019) Degree-biased random walk for large-scale network embedding. Future Gener Comput Syst 100:198–209
31. Nguyen GH, Lee JB, Rossi RA, Ahmed NK, Koh E, Kim S (2018) Continuous-time dynamic network embeddings. In: Companion of the web conference 2018 on the web conference 2018, pp 969–976. International World Wide Web Conferences Steering Committee
32. Yuan S, Wu X, Xiang Y (2017) Sne: signed network embedding. In: Pacific-Asia conference on knowledge discovery and data mining. Springer, Berlin, pp 183–195
33. Bagavathi A, Krishnan S (2018) Multi-net: a scalable multiplex network embedding framework. In: International conference on complex networks and their applications. Springer, Berlin, pp 119–131

34. Qu C, Zhan X-X, Wang G, Wu J, Zhang Z-K (2019) Temporal information gathering process for node ranking in time-varying networks. Chaos, Interdiscip J Nonlinear Sci 29(3):033116
35. Isella L, Stehlé J, Barrat A, Cattuto C, Pinton J-F, Van den Broeck W (2011) What's in a crowd? Analysis of face-to-face behavioral networks. J Theor Biol 271(1):166–180
36. Michalski R, Palus S, Kazienko P (2011) Matching organizational structure and social network extracted from email communication. In: International conference on business information systems. Springer, Berlin, pp 197–206
37. Chaintreau A, Hui P, Crowcroft J, Diot C, Gass R, Scott J (2007) Impact of human mobility on opportunistic forwarding algorithms. IEEE Trans Mob Comput 6:606–620
38. Opsahl T (2013) Triadic closure in two-mode networks: redefining the global and local clustering coefficients. Soc Netw 35(2):159–167
39. DNC emails network dataset—KONECT (2017) http://konect.uni-koblenz.de/networks/dnc-temporalGraph
40. Opsahl T, Panzarasa P (2009) Clustering in weighted networks. Soc Netw 31(2):155–163
41. Singer U, Guy I, Radinsky K (2019) Node embedding over temporal graphs. arXiv preprint. arXiv:1903.08889
42. Zuo Y, Liu G, Lin H, Guo J, Hu X, Wu J (2018) Embedding temporal network via neighborhood formation. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining. ACM, New York, pp 2857–2866
43. Zhou L, Yang Y, Ren X, Wu F, Zhuang Y (2018) Dynamic network embedding by modeling triadic closure process. In: Thirty-second AAAI conference on artificial intelligence
44. Gao M, Chen L, He X, Zhou A (2018) Bine: bipartite network embedding. In: The 41st international ACM SIGIR conference on research & development in information retrieval, pp 715–724
45. Kovács IA, Luck K, Spirohn K, Wang Y, Pollis C, Schlabach S, Bian W, Kim D-K, Kishore N, Hao T et al (2019) Network-based prediction of protein interactions. Nat Commun 10(1):1–8
46. Cao R-M, Liu S-Y, Xu X-K (2019) Network embedding for link prediction: the pitfall and improvement. Chaos, Interdiscip J Nonlinear Sci 29(10):103102
47. Yin Z, Shen Y (2018) On the dimensionality of word embedding. In: Advances in neural information processing systems, pp 887–898
48. Zhan X-X, Hanjalic A, Wang H (2019) Information diffusion backbones in temporal networks. Sci Rep 9(1):6798
49. Zhan X-X, Liu C, Zhou G, Zhang Z-K, Sun G-Q, Zhu JJ, Jin Z (2018) Coupling dynamics of epidemic spreading and information diffusion on complex networks. Appl Math Comput 332:437–448
50. Wang S, Tang J, Aggarwal C, Chang Y, Liu H (2017) Signed network embedding in social media. In: Proceedings of the 2017 SIAM international conference on data mining. SIAM, Philadelphia, pp 327–335