LP-ROBIN: Link Prediction in Dynamic Networks exploiting Incremental Node Embedding

Emanuele Pio Barracchia[a], Gianvito Pio[a,b,*], Albert Bifet[d,e], Heitor Murilo Gomes[d], Bernhard Pfahringer[d], Michelangelo Ceci[a,b,c]

[a] *Dept. of Computer Science, University of Bari Aldo Moro, Bari (Italy)*
[b] *Big Data Lab. - National Interuniversity Consortium for Informatics, Rome (Italy)*
[c] *Dept. of Knowledge Technologies, Jožef Stefan Institute, Ljubljana (Slovenia)*
[d] *Dept. of Computer Science, University of Waikato (New Zealand)*
[e] *LTCI, Télécom Paris, Institute Polytechnique de Paris (France)*

Abstract

In many real-world domains, data can naturally be represented as networks. This is the case of social networks, bibliographic networks, sensor networks and biological networks. Some dynamism often characterizes these networks as their structure (i.e., nodes and edges) continually evolves. Considering this dynamism is essential for analyzing these networks accurately. In this work, we propose LP-ROBIN, a novel method that exploits incremental embedding to capture the dynamism of the network structure and predicts new links, which can be used to suggest friends in social networks, or predict interactions in biological networks, just to cite some. Differently from the state-of-the-art methods, LP-ROBIN can work with mutable sets of nodes, i.e., new nodes may appear over time without being known in advance. After the arrival of new data, LP-ROBIN does not need to retrain the model from scratch, but learns the embeddings of the new nodes and links and updates the latent representations of old ones, to reflect changes in the network structure for link prediction purposes. The experimental results show that LP-ROBIN achieves better performances, in terms of AUC and F1-score, and competitive running times with respect to baselines, static node embedding approaches and state-of-the-art methods which use dynamic node embedding.

Keywords: Link prediction, Dynamic networks, Node embedding

[*] Corresponding author

1. Introduction

Nowadays, everything is connected, ranging from smartphones in a mobile network to people on social networks. Relationships can represent any kind of interaction, and even sent emails and messages can establish and describe a network of interactions. Over the last few years, the importance of such networks has rapidly gained attention, becoming one of the most studied fields. In particular, social networks like Twitter and Instagram became very central in these studies, since they represent a powerful and important tool for users to express their ideas and preferences. This data can be analyzed, for example, to help companies in the design of targeted products or to analyze the spread of the information after a disaster, in order to help emergency agencies to develop mitigation plans [22].

Looking at the research advances in the machine learning community, one of the most challenging aspect is the possibility to consider the temporal dimension: we can find several recent approaches that perform link prediction [18], discover communities and analyze their evolution [10], or detect anomalies [33] in dynamic networks, exploiting traditional and temporal features. The dynamicity, i.e. the ability to evolve over time, is an important characteristic of real-world networks: every second, 8,799 new tweets are posted on Twitter, 963 Instagram photos are uploaded, 2,870,940 emails are sent around the world[1], leading to new relationships in the networks.

The ability to catch the dynamicity of a network is therefore fundamental, since it is possible to capture additional aspects, such as temporal or seasonal trends, or the establishment of a new link between two users as a consequence of the creation of other links. Additionally, traditional (static) approaches usually need to analyze the whole data from scratch every time a new chunk of data arrives, leading to significantly affecting the computational efficiency and compromising the applicability of such approaches to real scenarios. Such issues are commonly alleviated by analyzing only the last snapshots [8] or by relying on a sliding-window setting [2], possibly leading to discarding potentially useful information. To face these problems, there is a need for new approaches that can capture the dynamic nature of the networks and adapt the extracted models in an incremental way, by processing newly arriving data in a lifelong machine learning setting.

[1]Statistics collected on 8th January 2020 at 11.00 am from `https://www.internetlivestats.com/one-second/`

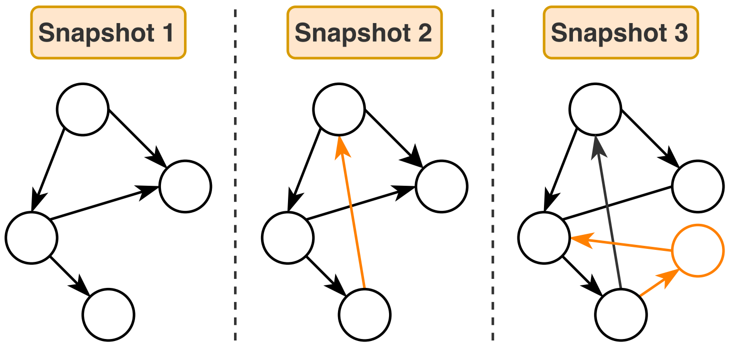


Figure 1: Example of different temporal snapshots of a network. We highlight in orange new nodes and edges appearing in each snapshot.

As already introduced, one of the tasks commonly investigated in the literature is the link prediction task. Given a network, the goal is to predict the existence of new links, e.g., new relationships between people in a social network, new collaborations in a professional network, or a new message in an email exchange system. Focusing on this task, in this paper we propose a new method, called LP-ROBIN (Link Prediction through incRemental nOde emBeddINg), that is able to solve the link prediction task on dynamic networks where structural changes, such as the addition of new nodes and new links, may happen over time (Figure 1). LP-ROBIN treats the link prediction task as a binary classification problem, by considering as *positive* the links that will appear in the next snapshot of the network. To achieve this goal, a neural network is trained on a latent representation of the network structure, that embeds the properties of both links and involved nodes. Both the classifier and the latent representation are not learned in batch mode, but incrementally, without the need to retrain the models from scratch after the arrival of new data. The strength of this strategy is the possibility to learn hidden patterns from the latent representation of data, and exploit them to discover new links over time. This characteristic makes LP-ROBIN a general and powerful tool, potentially applicable to many application domains.

The node embedding approach proposed in this paper is based on random walks [36, 16] and aims at preserving the network closeness among nodes in the learned feature space. Differently from existing state-of-the-art methods, LP-ROBIN learns an embedding for random walks and constructs node embedding by aggregation. In particular, nodes are represented according to the random walks in which they appear and their position in the walks, that weighs the contribution of each walk in the representation of the node. Therefore, nodes appearing close to each other in the network will likely be involved in multiple common random walks, leading to properly capturing network autocorrelation phenomena [42]. In social analysis, autocorrelation can be recognized in the homophily principle, which states that people linked through relationships (e.g., friendship) usually share sociodemographic, behavioral, and intra-personal characteristics. This principle can be easily extended to (and should be taken into account for) any network of interactions. Moreover, the approach we introduce in this paper updates the embedding model incrementally, without re-learning it from scratch when new data arrives.

Overall, the main methodological contributions provided by LP-ROBIN can be summarized as follows:

- it can work with dynamic networks where nodes and links may evolve over time and are not known apriori. This characteristic makes LP-ROBIN applicable in real-world scenarios, where it is almost impossible to know or predict the nodes that will appear in future snapshots;
- it performs node embedding in dynamic networks by updating the latent representation incrementally with the arrival of new data, rather than re-learning it from scratch;
- it solves the link prediction task on dynamic networks, leveraging possible hidden patterns discovered from the incremental node embedding.

These characteristics also allow LP-ROBIN to naturally handle possible issues caused by *concept drift*, namely, relevant changes in the structure of the network or of properties of nodes and links, that may compromise the accuracy of the predictive model over time. Indeed, the incremental fashion we propose for learning the embeddings and the classifier allows LP-ROBIN to gradually forget old or obsolete representations of nodes and links, and to adapt to the new state of the network.

The rest of the paper is structured as follows: in Section 2 we briefly describe the work in the literature that is related to the present paper; in Section 3 we describe the proposed method LP-ROBIN; in Section 4 we outline the time complexity of LP-ROBIN, while in Section 5 we evaluate its performance on publicly available datasets and discuss the results. Finally, in Section 6 we draw some conclusions and outline possible future work.

2. Related work

Real-world networks, such as social networks and professional networks, change their state (in terms of nodes, links and features) over time. Due to this characteristic, they are usually referred to as *dynamic networks*.

In the literature, many works focused on the study of dynamic networks, aiming to predict their evolution on the basis of historical data. For example, in [12] the authors developed different methods to identify spammers in evolving multi-relational social networks. In particular, they analyze the interactions among users in the network, and exploit structural features, sequence modeling and collective reasoning to detect spammer accounts. However, the models need to be retrained every time new samples arrive. This aspect represents a strong limitation in real-world scenarios, where the rate of generation of new data is high.

In [38], the authors proposed a new framework to study influence in professional networks. During the creation of a model, the authors consider different types of actions performed by users (e.g., changing jobs, adding a new skill, following content, joining groups) and analyze the effect of such actions on the network. Their framework also exploits an action propagation graph to capture the behavior of the users in response to the actions taken by their friends. Moreover, in [40], the authors propose a new framework, called SourceSeer, that exploits articles, blogs, social networks, search engine logs and micro-blogging services, to forecast disease outbreaks.

Due to the plethora of different analyses that can be performed on network data, in the following, we focus on the task considered in this paper. In particular, we briefly review existing methods that solve the link prediction task in dynamic networks, as well as methods aiming to represent nodes and links in a numerical feature space (network embedding approaches).

2.1. Link prediction in dynamic networks

One relevant approach that is able to solve the link prediction task in dynamic networks is that proposed in [37]. In particular, the authors propose the method GraTFEL, that extends the adoption of graphlets[2] to dynamic networks. When a link is added or removed, some graphlets will be affected, producing graphlet transition events. GraTFEL builds a representation of each node pair by exploiting, in combination with an unsupervised feature

[2]Small connected non-isomorphic induced subgraphs of a network.

extraction approach, the frequency of the graphlet transition events associated with the pair. These representations are then used to solve the link prediction task by learning a supervised classification model.

In [48], the authors proposed LIST, a method that performs link prediction on dynamic networks taking into account both the structure of the network at the current timestamp, and the evolution of the patterns discovered during the analysis of the different snapshots. To reach its goal, LIST makes use of network propagation and temporal matrix factorization techniques.

In [8] the authors proposed an approach to compute and track the probability of the existence of a link over time. Each link is represented by multiple similarity metrics observed at each snapshot from the structure of the network (e.g., Common Neighbors, Shortest Path and Jaccard Index) and Stochastic Learning Weak Estimators (SLWEs), i.e., estimators which goal is to update class probabilities every time new instances are observed. However, the representation for each link is built by considering only the last k snapshots, discarding previous, potentially useful, data.

In [7] the authors proposed a framework that can detect missing links in the analyzed snapshot and predict the appearance of new links in future snapshots. The framework exploits the concept of *attractive force* between nodes, computed as the ratio between the product of the node degrees and the shortest-distance between them. This approach directly analyzes the adjacency matrix, possibly leading to inefficiencies for large networks.

In [9], the author proposed NexT, a framework that solves link prediction tasks in dynamic, location-based social networks (LBSNs), with the goal of forecasting the next location of an individual, on the basis of the observations of her mobility behavior, of the recent locations she visited, as well as of the global mobility in the considered geographic area. NexT integrates frequent pattern mining and a tree-based classifier, that also exploits a set of spatio-temporal features representing locations and movements along them.

In [46], the authors addressed the challenge of tracking trajectories and communication records of mobile phone users, and identifying correlations between the users' movements and their closeness in the social network, with the final goal of predicting the appearance of new links. The authors demonstrated that mobility-based measures can lead to competitive performance, which can be further improved by also exploiting network-based measures.

In [31], a survey of state-of-the-art methods for link prediction in social networks has been proposed. In particular, the authors categorize the existing approaches, emphasizing their strong and weak points, and discuss the challenges related to this task. In particular, some of the problems highlighted by the authors include the low prediction accuracy, the heterogeneous nature of some networks, and the scalability of the proposed frameworks. Moreover, a relevant observation made by the authors is related to the poor availability of approaches that can analyze networks where both the set of nodes and the set of edges evolve over time [18]. Indeed, most of the methods [37, 48] try to predict new links in networks characterized by a static set of nodes, i.e., the full set is known a priori and does not change over time. This strong assumption does not reflect real-world scenarios, such as social networks, where at every second new users might join or old users might leave. In these cases, such approaches have to re-learn their models from scratch.

2.2. Node and link embedding in dynamic networks

One of the major challenges related to the study and the analysis of networks concerns the representation of nodes, links and whole networks when they are treated as training examples of a machine learning method. In the literature, we can find several approaches [48, 7] that directly analyze the network structure represented through a binary adjacency matrix. However, this representation leads to inefficiencies when the network under analysis exhibits high levels of sparsity. To deal with this issue, in the literature we can find several techniques for dimensionality reduction, such as Principal Component Analysis (PCA), Singular Value Decomposition (SVD), and (Non-negative) Matrix Factorization [4, 19].

Recently, the concept of *embedding* has received attention in the research community. It consists in representing complex structures according to (relatively) low-dimensional numerical feature vectors. In the context of networks, we can distinguish between node embedding, edge embedding, and graph embedding, according to the unit of analysis the numerical feature vectors are associated with.

Approaches for node embedding are well established in the literature. In [36], the authors proposed DeepWalk, a framework that is able to generate node embeddings by exploiting random walks and the Skip-Gram model. Subsequently, an evolution of DeepWalk, called Node2Vec, has been proposed [16]. The main difference with respect to its predecessor is the ability to bias the random walks in order to adopt different (breadth-first, depth-first or hybrid) search strategies.

A different approach has been adopted in LINE [43]. It generates the node representation in the latent space by optimizing a function that takes into account the first-order and the second-order proximity, i.e., the direct connections between nodes and the similarity of their neighborhood.

However, all the above-mentioned methods work only on static networks, where the set of nodes and the set of edges do not change over time. One attempt towards handling the dynamicity of the network has been done in [11], where the authors propose a general approach to extend skip-gram based network embedding models to the dynamic context. Alternative approaches have been subsequently proposed, including [29], where the authors proposed a dynamic version of Node2Vec, [32] where the authors described a framework to incorporate the temporal dimension into node embeddings by exploiting temporal random walks, and [17], where the authors proposed a method that jointly learns the embedding for different time steps exploiting Node2Vec.

In [44], the authors proposed the system DyRep, which learns latent representations of the nodes by considering both the *dynamics of* the network, i.e., how the structure of the network evolves over time, and the *dynamics on* the network, i.e, the activities between nodes.

In [49] the authors proposed DynamicTriad, a framework that constructs node embeddings by exploiting closed triads, i.e., fully-connected triples of nodes. However, the proposed framework is applicable only on undirected graphs characterized by a pre-defined set of nodes.

In [26], the authors proposed JODIE, a framework that exploits two recurrent neural networks to compute the static and the dynamic embeddings of a node, and adopts a projection operator to predict the evolution of the embedding of the nodes in future snapshots. However, this method is strictly tailored to bipartite networks.

Lastly, in a recent work [15] the authors proposed the system *dyngraph2vec*. It implements three different approaches for dynamic node embedding, that take into account the evolution of the network structure. It also implements a link prediction method which can predict the next snapshot. However, the system needs to know the whole set of nodes in advance, i.e., it cannot adapt to the possible introduction of new nodes in the network over time.

As regards the identification of a numerical vector for links (link embedding), several techniques have been proposed in the literature. For example, the authors of Node2Vec [16] proposed four different operators to construct a link embedding starting from the embeddings of the involved nodes: average, Hadamard, weighted-L1 and weighted-L2. Other works [45] compared the performances obtained using these operators with those achieved by concatenating the feature vectors of the involved nodes, and proved that the concatenation generally leads to better results. For this reason, in this work, we adopt such a strategy.

In conclusion, even if several methods have been proposed in the literature to compute node or link embeddings in a dynamic scenario, they generally exhibit significant limitations: they either assume that the set of nodes does not change over time [49, 17] or are limited to undirected networks [49, 44]. On the contrary, the approach proposed in this paper, although based on random walks, like [36, 16], can also work *i)* when the set of nodes evolves over time; *ii)* in the case of directed networks; *iii)* in an incremental fashion, i.e. contrary to the classical batch learning mode, our approach does not re-learn the representation from scratch when new data arrives.

3. The LP-ROBIN method

In the following, we first introduce some useful definitions and the task we aim to solve. Next, we describe in detail the proposed approach.

Definition 3.1. *Dynamic network.* A dynamic network is a network that evolves over time. It can be represented as a sequence of snapshots $[G^1, G^2, ..., G^T]$, where $G^t = (V^t, E^t)$ is the state of the network at time t, characterized by the set of nodes V^t and the set of links E^t.

In this paper, we assume that nodes and links can appear over time, but cannot disappear. Formally, $\forall_{t=1,2,...,T-1}, V^t \subseteq V^{t+1}$ and $E^t \subseteq E^{t+1}$. It is noteworthy that this assumption does not limit the applicability of the proposed method since it is coherent with the considered task, namely, the prediction of new links appearing over time.

Definition 3.2. *Link prediction in dynamic networks.* Given a dynamic network $[G^1, G^2, ..., G^T]$, the goal of the link prediction task is to predict the set of new links $\hat{E}^{T+1}$ that will appear at time *T+1.*

It is worth clarifying that $\hat{E}^{T+1} \subseteq E^{T+1} - E^T$, since links that are actually predictable are those involving nodes that have been already observed during the training stage, i.e., V^T. More formally, $\hat{E}^{T+1} = \{\langle u, v\rangle \in E^{T+1} | \langle u, v\rangle \notin E^T \wedge u \in V^T \wedge v \in V^T\}$. However, the whole set of nodes V^{T+1} and links E^{T+1} are considered in the subsequent steps of the link prediction task, namely for the prediction of $\hat{E}^{T+2}$.

We model the link prediction task as a supervised learning problem. In particular, we treat it as a binary classification task, where a link is classified as 1 if it is expected to appear in the next snapshot of the network (i.e., it does not exist at time T, but will exist at time $T+1$), 0 otherwise (i.e., it does not exist at both time T and $T+1$ or it already exists at time T).

Differently from most of the existing methods, LP-ROBIN can work with directed dynamic networks, where two nodes may be connected through two different links, one for each direction. In any case, LP-ROBIN can also work on undirected dynamic networks, by representing each undirected link through a pair of directed links. Therefore, in the following, we will assume to work in the more general case of directed networks.

Methodologically, for each snapshot $G^t, t = 1, 2, ...T$, LP-ROBIN performs the following steps:

- Incremental construction and update (for nodes already observed in previous snapshots) of node embeddings, on the basis of random walks and of a weighting strategy that considers the position of nodes in the random walks;
- Generation of the link embeddings for new links observed in G^t, by concatenating the embedding of the involved nodes;
- Incremental fit and update of a binary classification model, on the basis of the embedding of the new links observed in G^t.

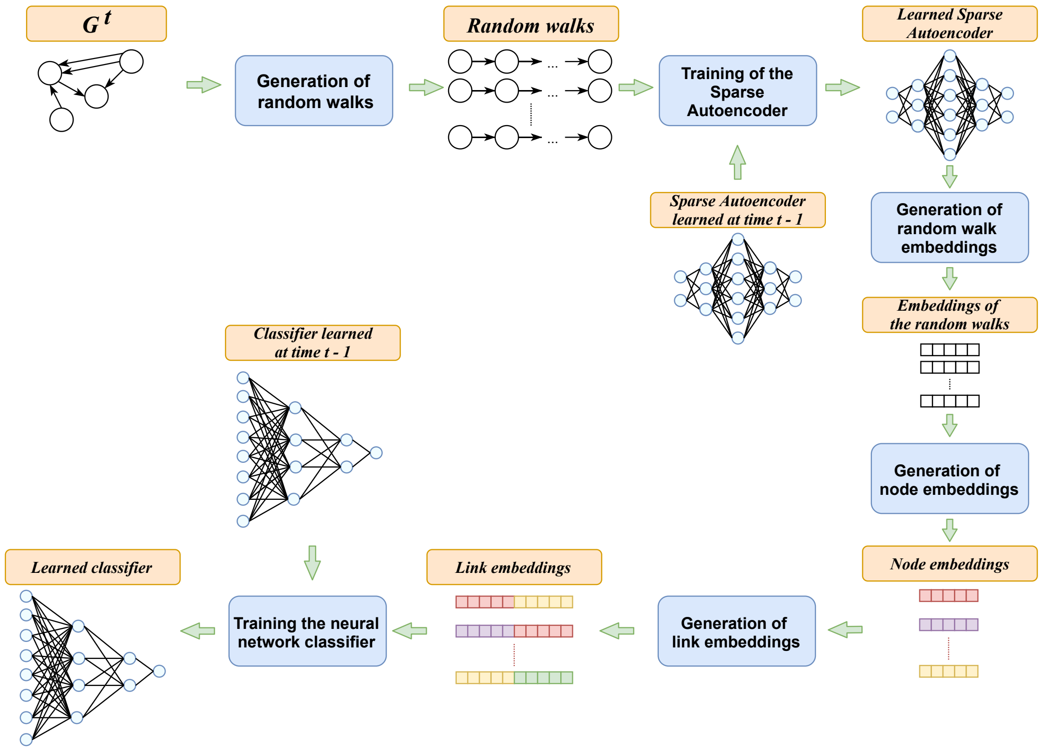
The trained model is then used to solve the link prediction task for G^{T+1}.

The whole workflow is depicted in Figure 2, while each step will be detailed in the following subsections.

3.1. Node embedding

Definition 3.3. *Node embedding in dynamic networks.* Identifying a node embedding in a dynamic network means to learn, $\forall t \in 1, ..., T$, a function $f^t : V^t \rightarrow \mathbb{R}^d$, with $d \ll |V^t|$. Considering $d \ll |V^t|$ means that the dimensionality of the identified feature space is much lower than the number of nodes, since a naive embedding approach would be that of directly considering the adjacency matrix.

As introduced in Section 2.2, in the literature many approaches have been proposed to compute node embedding on dynamic networks, but they



G^t
Generation of random walks
Random walks
Training of the Sparse Autoencoder
Learned Sparse Autoencoder
Sparse Autoencoder learned at time t - 1
Generation of random walk embeddings
Embeddings of the random walks
Generation of node embeddings
Node embeddings
Classifier learned at time t - 1
Learned classifier
Training the neural network classifier
Link embeddings
Generation of link embeddings

Figure 2: Incremental learning and update of the SAE and of the classifier learned at time $t-1$, on the basis of new data observed at time t, in order to predict new links for the time $t+1$.

generally assume that the set of nodes does not change over time (i.e., they assume to know the whole set of nodes a priori) [49, 17], or are able to work only on undirected networks [44, 49].

In the following, we describe our novel approach for node and link embedding, that overcomes such limitations and works incrementally. Since it is based on the concept of *random walks* in dynamic networks, in the following we first formally define it.

Definition 3.4. *Random walks in dynamic networks.* Given a snapshot G^t, a random walk is defined as a sequence of nodes generated starting from a given node and taking a given number of random steps, following the links of the network observed at time t, i.e., links in E^t.

When a new snapshot of the network G^t is observed, LP-ROBIN constructs n_rw random walks of length l_rw starting from every new node in V^t and every node appearing as a source node of new links in E^t. Formally, for $t = 1$, random walks are computed starting from all the nodes in V^1, while for $t > 1$, they are computed starting from the following set of nodes:

$$\{v | v \in (V^t - V^{t-1}) \vee \exists v' \in V^t s.t. \langle v, v' \rangle \in (E^t - E^{t-1})\} \tag{1}$$

This strategy allows us to take into account the effect of new nodes on the embedding of other nodes also when such new nodes do not have any outlink.

The identified random walks are represented as l_rw - dimensional vectors of node IDs. If a random walk is shorter than l_rw, because it reached a node without outlinks, the corresponding vector is 0-padded. These vectors are subsequently given as input to a sparse autoencoder (SAE), which goal is to construct a representation of each random walk in the latent space.

An autoencoder is a specific kind of artificial neural network, that has already been fruitfully adopted in the literature to generate embeddings [21, 47]. It aims at learning an encoding function, which encodes the input into a *code* (or *latent representation*), and a decoding function that, given the code, approximately reconstructs the input. The autoencoder is trained aiming to minimize a loss function based on the difference between the original input and its reconstructed version. In our case, we use a sparse autoencoder with l1-regularization [14], which allows us to possibly learn an *over-complete* representation of the input vector, i.e., the number of neurons in the hidden layer can be higher than that of the input layer, without incurring overfitting issues [39].

Different architectures can be designed to solve this task. In this paper, we design the architecture as follows:

- an input layer, consisting of a number of neurons equal to the length of the random walks l_rw;
- one encoding hidden layer with $d/2$ neurons, which activation is penalized through l1-regularization;
- one encoding hidden layer with d neurons, that represents the d-dimensional encoded representation of the input, which activation is penalized through l1-regularization;
- one decoding hidden layer having $d/2$ neurons;
- an output layer having a number of neurons equal to that of the input layer (i.e., l_rw).

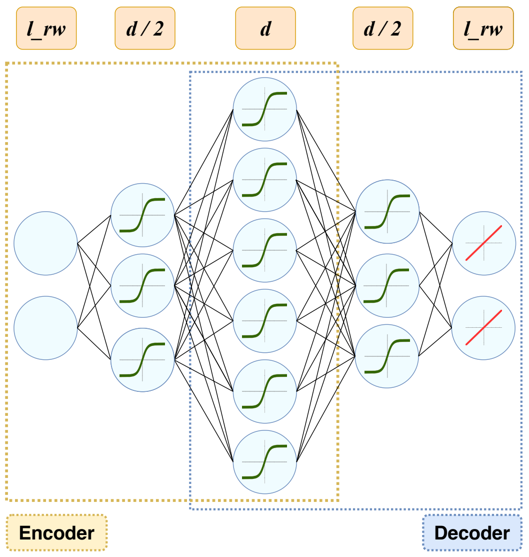


Figure 3: The architecture of the sparse autoenconder used to learn an embedding for random walks.

All the internal layers use *tanh* as activation function, while the output layer uses a *linear* activation function. The adopted loss function is the mean squared error. A graphical overview of the proposed architecture is shown in Figure 3.

After training the sparse autoencoder, we can use its d-dimensional hidden layer to obtain the latent representation of a given random walk. Formally, considering that each node in V^t is assigned to a unique ID in $\{1, 2, ..., |V^t|\}$, we define the function $g^t : \{1, 2, ..., |V^t|\}^{l\_rw} \rightarrow \mathbb{R}^d$, that takes as input a random walk represented as a l_rw-dimensional vector of node IDs and returns the d-dimensional real-valued vector observed in the second hidden layer once the random walk has been fed to the SAE.

Subsequently, LP-ROBIN computes the embedding for each node v in V^t by aggregating the embedding of the random walks that traverse v. In particular, we propose the following different aggregations.

Weighted sum: inspired by Word2vec [30], where the embedding of a textual document is computed as the sum of the embedding of its words, we compute the embedding of a node as the sum of the embedding of the random walks in which it appears. However, we weight the contribution of each random walk in the sum according to the position of the considered node (i.e., the distance, in terms of the number of steps, from the source node). Formally, given a node $v \in V^t$, its embedding at time t is computed as:

$$\widetilde{emb}^t(v) = \sum_{rw \in RW_v^t} \frac{1}{pos(v, rw)} \cdot g^t(rw) \tag{2}$$

where RW_v^t is the set of the random walks generated at time t involving the node v, $g^t(rw)$ is the function that returns the latent representation of the random walk rw through the SAE, and $pos(v, rw)$ returns the position of the node v in the random walk rw.

Weighted average. In this case, the embedding of a node is computed as the weighted average of the embedding of the random walks in which it appears. The main difference with respect to the weighted sum is that the embedding is normalized according to the position of the node in the random walks. Formally, given a node $v \in V^t$:

$$\widetilde{emb}^t(v) = \frac{1}{\sum\limits_{rw \in RW_v^t} \frac{1}{pos(v,rw)}} \cdot \left(\sum_{rw \in RW_v^t} \frac{1}{pos(v, rw)} \cdot g^t(rw) \right) \tag{3}$$

It is noteworthy that in both cases, weighted sum and weighted average, $\widetilde{emb}^t(v)$ represents the node v according to the state of the network in the last snapshot. In order to represent the state of the network already observed for previous snapshots, we introduce the vector $emb^t(v)$ defined as follows:

$$\begin{cases} emb^t(v) = \widetilde{emb^t}(v) & \text{if } t = 1 \lor v \in (V^t - V^{t-1}) \\ emb^t(v) = \alpha \cdot \widetilde{emb^t}(v) + (1 - \alpha) \cdot emb^{t-1}(v) & \text{otherwise} \end{cases} \tag{4}$$

where $\alpha \in [0, 1]$ is a user-defined parameter that represents the weight of the embedding computed according to the new random walks with respect to that computed according to previous snapshots.

We remark that the approach we propose is significantly different from that adopted by Node2Vec. Indeed, Node2Vec directly aims at learning a representation of each node according to its neighborhood, "observed" through random walks. On the contrary, LP-ROBIN learns a representation for random walks (by means of $g^t(\cdot)$) and constructs node embedding through aggregation. Intuitively, the approach followed by LP-ROBIN aims at optimizing the representation of random walks, rather than that of nodes, meaning that it is able to better preserve and represent the network structure more globally. The representation of nodes, obtained by aggregation, can therefore be considered as a latent representation of their roles in the network.

Moreover, the adoption of a sparse autoencoder with the proposed architecture provides two additional advantages:

- apart from the assumptions $d \ll |V^t|$, and $l\_rw \leq d/2$, we do not impose any additional constraints on the length l_rw of the considered random walks[3];
- training the SAE from a representation of random walks, rather than from a representation of nodes, gives us the advantage of being independent on the set of nodes, namely, if new nodes appear over time, the SAE can be incrementally fed with new random walks, without restructuring its architecture nor retraining it from scratch.

In this way, LP-ROBIN is able to adapt the learned model to the changes of the network, combining both the information learned in previous snapshots and that conveyed from the new snapshot, in an incremental and efficient manner.

The proposed algorithms for constructing node embeddings and random walks are formalized in Algorithms 1 and 2. From Algorithm 1, it is possible to see that, at each new snapshot, LP-ROBIN incrementally updates the SAE by exploiting the inherent incremental learning strategy of autoencoders (see the instruction "feed SAE with RW^t" at line 11) and incrementally updates the embedding according to Equation (4) (see line 18).

Finally, in order to represent links for the link prediction task, as introduced in Section 2.2, we build the embedding for new links observed in $E^t - E^{t-1}$ by concatenating the embedding of the involved nodes. Therefore, each link is represented through a $2d$-dimensional feature vector.

[3]The constraint $l\_rw \leq d/2$ can in practice be relaxed: if $l\_rw > d/2$, we can adopt a simpler architecture with a single hidden layer with d neurons.

Algorithm 1: Node embedding

```
Data:
· $G^t = (V^t, E^t)$: the network snapshot at time $t$;
· $G^{t-1} = (V^{t-1}, E^{t-1})$: the network snapshot at time $t-1$ (if $t > 1$);
· $SAE$: learned sparse autoencoder (if $t > 1$);
· $embs^{t-1}$: node embeddings computed at time $t-1$ (if $t > 1$);
· $n\_rw$: number of random walks to perform;
· $l\_rw$: length of random walks;
· $d$: dimensionality of the embedding;
· $\alpha \in [0, 1]$: weight of the random walks of the current snapshot;
· $agg \in \{weighted\_sum, weighted\_avg\}$: aggregation function
Result:
· $embs^t$: embedding learned for each node in $V^t$.
/* Identification of new nodes and links                                 */
1  $newV^t \leftarrow V^t$;     $newE^t \leftarrow E^t$;
2  if $t > 1$ then
3      $newV^t \leftarrow newV^t - V^{t-1}$;     $newE^t \leftarrow newE^t - E^{t-1}$;
/* Identification of source nodes for random walks, see Eq.(1)           */
4  $nodesToEmb \leftarrow \{v | v \in newV^t \vee \exists v' \in V^t s.t. \langle v, v' \rangle \in newE^t\}$;
/* Generation of random walks                                             */
5  $RW^t \leftarrow \emptyset$;
6  foreach $v \in nodesToEmb$ do
7      $rws = randomwalks(G^t, v, n\_rw, l\_rw)$;
8      $RW^t \leftarrow RW^t \cup rws$;
/* Incremental learning of the SAE to represent random walks              */
9  if $SAE$ does not exist then
10     $SAE \leftarrow createSparseAutoencoder(d)$;
11 feed SAE with $RW^t$;
/* Construction of node embedding                                         */
12 $embs^t \leftarrow \emptyset$;
13 foreach $v \in nodesToEmb$ do
14     if $agg = weighted\_sum$ then
15         compute $\widetilde{emb}^t(v)$ according to Eq. (2);
16     else if $agg = weighted\_avg$ then
17         compute $\widetilde{emb}^t(v)$ according to Eq. (3);
18     compute $emb^t(v)$ according to Eq. (4), on the basis of $emb^{t-1}(v)$
         and $\alpha$;
19     $embs^t = embs^t \cup \{emb^t(v)\}$;
20 return $embs^t$;
```

Algorithm 2: *randomwalks (G^t, v, n_rw, l_rw)*

```
Data:
· $G^t = (V^t, E^t)$: the network snapshot at time $t$;
· $v \in V^t$: the source node for random walks;
· $n\_rw$: number of random walks to perform;
· $l\_rw$: length of random walks.
Result:
· $rws$: $n\_rw$ random walks of length $l\_rw$ generated starting from $v$.
1 $rws \leftarrow \emptyset$;
2 for $i \leftarrow 1$ to $n\_rw$ do
3     $rw \leftarrow [v]$;
4     $lastNode \leftarrow v$;
5     for $k \leftarrow 2$ to $l\_rw$ do
          /* A random neighbor of $lastNode$, according to $E^t$; 0 if
             there is no outlink                                    */
6         $lastNode \leftarrow rndNeighbor(lastNode, E^t)$;
7         $rw = rw.append(lastNode)$;
8     $rws \leftarrow rws \cup \{rw\}$;
9 return $rws$;
```

3.2. Prediction of new links

As introduced in Section 1, LP-ROBIN considers the link prediction task as a binary classification problem. In particular, given a dynamic network represented by a sequence of snapshots $[G^1, G^2, ..., G^T]$, we aim at predicting the set of new links that will appear in the next snapshot G^{T+1}.

Methodologically, we train a classifier that is able to distinguish between existing and non-existing links. These links are represented according to the strategy described in Section 3.1 and given in input to a neural network classifier used to solve the link prediction task. The architecture of such a neural network is:

- the input layer, consisting of $2d$ neurons, able to take as input $2d$-dimensional vectors associated to links;
- one hidden layer with d neurons;
- one hidden layer with $d/2$ neurons;

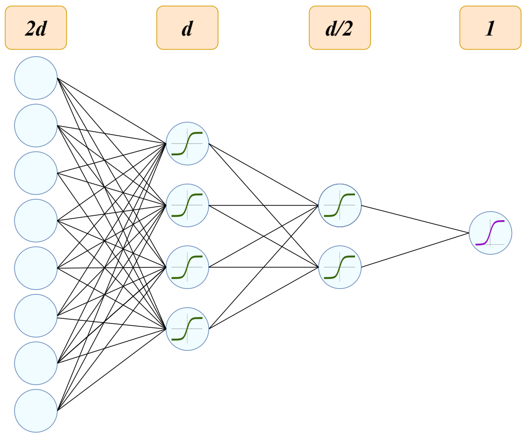


Figure 4: The neural network architecture of the classifier for the link prediction phase.

- the output layer, with 1 neuron, that represents the predicted label: instances (i.e., links) are labeled as positive if the output is greater or equal to 0.5, negative otherwise.

All the internal layers use *tanh* as activation function, while the output layer uses a *sigmoid* activation function. The adopted loss function is the binary cross-entropy, which is appropriate for binary classification tasks. A graphical overview of the architecture is depicted in Figure 4.

It is worth mentioning that the neural network classifier is not retrained from scratch when a new snapshot is observed. Given the classifier learned at time $t-1$, we feed it with a new set of examples of links, namely: all the links that appeared in the last snapshot $newE^t = E^t - E^{t-1}$, labeled as positive examples, and $|newE^t|$ examples randomly selected from $V^t \times V^t - E^t$, labeled as negative examples. This approach is motivated by the huge number of possible non-existing links, which may cause problems of class imbalance [3].

In this way, the neural network exploits latent patterns learned in previous snapshots, together with the information conveyed by new examples, possibly leading to better performances. The fitted model is finally adopted to predict the existence of links for the subsequent snapshot G^{t+1}.

We stress that this approach allows us to handle possible issues caused by the *concept drift* phenomenon. Indeed, although LP-ROBIN is able to take into account the evolution of the network observed in previous snapshots, it also incrementally adapts to newly observed nodes and links over time, thus making the predictive model robust to changes of the network structure.

4. Time Complexity

To estimate the time complexity of LP-ROBIN, we analyze the complexity of each step of its workflow.

Let n_t be the number of newly introduced nodes at time t, i.e., $n_t = |nodesToEmb|$, that LP-ROBIN has to embed (see Equation (1) and Algorithm 1, line 4). We recall that l_rw and n_rw are the length and the number of random walks, respectively, to be identified for each node. Since we can randomly select a neighbor of a given node in constant time (this information is already available in the network structure), the time complexity at time t for the identification of random walks for all the nodes of $nodesToEmb$ corresponds to:

$$O(n_t \cdot n\_rw \cdot l\_rw) \tag{5}$$

Following the estimation of the time complexity for AutoEncoders in [5], the complexity of the identification of d-dimensional embeddings for $n_t \cdot n\_rw$ random walks by means of the SAE can be approximated to:

$$O(n_t \cdot n\_rw \cdot d^2) \tag{6}$$

The time complexity of the computation of the embedding for a given node depends on the existence or absence of a representation already learned in a previous snapshot for the same node. The difference would actually consist, in the case the node was already present, of one additional operation to linearly combine the existing embedding with the current one (see Equation (4)). Therefore, we can conclude that the overall time complexity of this step, in this case, corresponds to:

$$O(n_t \cdot n\_rw) \tag{7}$$

The next step consists of the identification of link embeddings. We remind that it is obtained by concatenating the latent representation of the involved nodes. Let e_t be the number of edges to embed at time t, i.e., $e_t = |E^t - E^{t-1}|$. Since we can retrieve the embedding of a node in constant time, the complexity of the construction of e_t link embeddings is equal to:

$$O(2 \cdot e_t) \tag{8}$$

Finally, we estimate the complexity of the link prediction step. Following the same assumptions made for AutoEncoders, and considering that the largest layer has $2d$ neurons (see Figure 4), the complexity of this step can be approximated to:

$$O(e_t \cdot (2d)^2) \tag{9}$$

Therefore, by summing up the time complexity of all the phases performed by LP-ROBIN, we can conclude that the overall time complexity at each time t corresponds to:

$$O(n_t \cdot n\_rw \cdot l\_rw) + O(n_t \cdot n\_rw \cdot d^2) + O(n_t \cdot n\_rw) + O(2 \cdot e_t) + O(e_t \cdot (2d)^2) \tag{10}$$

Considering that $l\_rw < d^2$, the time complexity of LP-ROBIN can be approximated to:

$$O(n_t \cdot n\_rw \cdot d^2) + O(e_t \cdot d^2) \tag{11}$$

This means that the time complexity of LP-ROBIN is linear in the number of nodes added at the last snapshot, linear in the number of random walks considered, linear in the number of new edges added at the last snapshot, and quadratic in the dimensionality of the embedding.

5. Experimental evaluation

We evaluated the performances achieved by LP-ROBIN with the following parameter values: $n\_rw \in \{10, 20\}$ (number of random walks for each node), $l\_rw \in \{10, 20\}$ (length of random walks), $d \in \{64, 128, 256\}$ (size of the node embedding), $\alpha \in \{0.0, 0.5, 1.0\}$ (weight of the last snapshot), $agg \in \{weighted\_sum, weighted\_avg\}$ (function to aggregate the embeddings of random walks). The best values have been selected through a grid search.

We implemented the artificial neural networks used by LP-ROBIN with Tensorflow 2.0. The weights of the neural networks have been initialized using the Xavier scheme [13]. Moreover, the optimization has been performed using the Adam optimizer [23], iterating for a maximum number of epochs equal to 200. We tuned the hyper-parameters of the neural networks (batch size and learning rate) by exploiting the Hyperopt library [1] and using the 20% of the training set as validation set. For this evaluation the following values have been considered: batch size $\in \{2^7, 2^8, 2^9, 2^{10}\}$, learning rate $\in [0.00001, 0.01]$, the parameter used for the l1-regularization $\lambda \in [0.00001, 0.01]$.

LP-ROBIN has been compared with 8 baselines and state-of-the-art competitors. The evaluation measures we collected for the comparison are the Area under the ROC curve (AUC) and the F1-score. For the AUC we directly considered the scores returned by the predictors (for a fair comparison, the same procedure is applied for LP-ROBIN and to its competitors) in order to identify the (FP-rate, TP-rate) points of the ROC curve and properly plot them. For the F1-score we considered the example as positive if the returned score(s) suggest that the likelihood of being positive is higher than that of being negative, negative otherwise. All the experiments have been performed on a server equipped with a 6 cores CPU @ 3.50 GHz, 64 GB of RAM, and a NVIDIA GeForce GTX Titan X.

5.1. Datasets

We performed our experiments on three public datasets:

CollegeMsg[4] **[34].** This dataset contains information about messages exchanged on an online social network at the University of California, Irvine. Each message is characterized by a source user, a destination user, and an associated timestamp ranging from April 2004 to October 2004. We constructed seven monthly snapshots, namely G^0:04/2004, G^1:05/2004, G^2:06/2004, G^3:07/2004, G^4:08/2004, G^5:09/2004, G^6:10/2004.

BitcoinOTC[5] **[25, 24].** This dataset contains timestamped links between November 2010 and December 2016 related to a who-trust-whom network of people that use the platform BitcoinOTC. We considered only links associated with a trust value greater than 0, i.e., excluding links representing the fact that someone does not trust someone else. We divided the dataset into 7 yearly snapshots, namely G^0:2010, G^1:2011, G^2:2012, G^3:2013, G^4:2014, G^5:2015, G^6:2016.

Wiki-Talk[6] **[35, 27]**. This dataset contains timestamped links collected between November 2001 and August 2007 related to editing activities of a user's Talk page. Each node represents a user's Talk page, while each link represents an editing activity performed by the source node on the destination node's Talk page. We divided the dataset into seven yearly snapshots, namely G^0:2001, G^1:2002, G^2:2003, G^3:2004, G^4:2005, G^5:2006, G^6:2007.

In Table 1, we show the number of nodes of each snapshot for all the considered datasets. For training-prediction purposes, we considered temporally adjacent snapshots (see Table 2). We remind that, for the link prediction

[4]https://snap.stanford.edu/data/CollegeMsg.html
[5]https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html
[6]https://snap.stanford.edu/data/wiki-talk-temporal.html

Table 1: The Number of nodes in each snapshot. We recall that the nodes are incrementally added to the network when a new snapshot arrives.

	G^0	G^1	G^2	G^3	G^4	G^5	G^6
CollegeMsg	504	1,514	1,731	1,780	1,827	1,875	1,899
BitcoinOTC	43	1,631	3,107	4,985	5,453	5,570	5,881
WikiTalk	17	1,161	5,681	25,255	112,654	513,138	1,140,149

task, instances correspond to links. Specifically, for each prediction task, the training instances are the links belonging to previous snapshots, with respect to the snapshot subject of the prediction. For example, for the prediction of snapshot G^2, the training instances (i.e., links) of the snapshots G^0 and G^1 are used. Note that, coherently with Definition 3.2, the capability of predicting new links appearing in G^6 (involving nodes already known in G^5) is evaluated when G^5 is included in the training snapshots. Note that LP-ROBIN works in an incremental fashion, whereas all the competitors work in batch mode and require a full retrain when a new snapshot arrives (see *Incr. Training Instances* and *Batch Training Instances* in Table 2). The examples actually considered in the prediction phase are those involving nodes already known during the training phase. This explains why the examples to predict in G^j are less than the training examples added when predicting G^{j+1}.

5.2. Competitor methods

We compared LP-ROBIN against the following competitors:

Adamic-Adar Index (AAI) [28]: the Adamic-Adar Index of a pair of nodes $\langle u, v\rangle$ is computed as $\sum_{w\in\Gamma(u)\cap\Gamma(v)} \frac{1}{log|\Gamma(w)|}$, where $\Gamma(x)$ is the neighborhood of x. This index tends to give more importance to common neighbors that have a small number of connections.

Common Neighbors (CN) [41]: given two nodes u and v, the predicted score associated with the link $\langle u, v\rangle$ is computed on the basis of their common neighbors, that is, $|\Gamma(u) \cap \Gamma(v)|$, where $\Gamma(x)$ is the set of neighbors of x.

Jaccard coefficient (JACCARD) [28]: the Jaccard coefficient of an edge $\langle u, v\rangle$ is computed as the ratio between the number of neighbors of u and v in common and the cardinality of the union of the sets of neighbors, i.e., $\frac{|\Gamma(u)\cap\Gamma(v)|}{|\Gamma(u)\cup\Gamma(v)|}$, where $\Gamma(x)$ represents the set of neighbors of x.

Resource Allocation Index (RAI) [50]: this index is based on the concept of resource allocation process and it is computed as $\sum_{w\in\Gamma(u)\cap\Gamma(v)} \frac{1}{|\Gamma(w)|}$,

Table 2: Summary of snapshots, number of training examples (in both incremental and batch settings), and prediction examples.

	Training Snapshots	**Prediction Snapshot**	**Incr. Training Examples**	**Batch Training Examples**	**Prediction Examples**
CollegeMsg	G^0	G^1	1,906	1,906	2,109
	G^0-G^1	G^2	12,725	14,631	1,531
	G^0-G^2	G^3	2,975	17,606	916
	G^0-G^3	G^4	1,131	18,737	563
	G^0-G^4	G^5	769	19,506	376
	G^0-G^5	G^6	517	20,023	180
BitcoinOTC	G^0	G^1	114	114	83
	G^0-G^1	G^2	7,629	7,743	1,149
	G^0-G^2	G^3	8,556	16,299	2,052
	G^0-G^3	G^4	11,139	27,438	1,728
	G^0-G^4	G^5	3,596	31,034	625
	G^0-G^5	G^6	953	31,987	39
Wiki-Talk	G^0	G^1	13	13	7
	G^0-G^1	G^2	3,412	3,425	2,011
	G^0-G^2	G^3	18,019	21,444	12,242
	G^0-G^3	G^4	76,541	97,985	50,961
	G^0-G^4	G^5	282,016	380,001	225,344
	G^0-G^5	G^6	1,157,534	1,537,535	458,292

where $\Gamma(x)$ is the neighborhood of node x. Like the Adamic-Adar Index, it gives more importance to neighbors involved in a small number of links.

Node2Vec [16]: we specifically compared our embedding approach with Node2Vec since they are, in principle, both based on random walks. In order to specifically evaluate the contribution of the node embedding mechanism implemented in LP-ROBIN, we plugged Node2Vec in place of our embedding approach in LP-ROBIN. Since Node2Vec works in batch mode, in this case also the link prediction algorithm works in batch mode.

DynAE [15]: DynAE is the *dyngraph2vec* variant, which is based on adapting autoencoders to the dynamic context.

DynRNN [15]: DynRNN is the *dyngraph2vec* variant that is based on recurrent neural networks in order to capture and exploit the temporal dimension.

DynAERNN [15]: DynAERNN is the *dyngraph2vec* variant that combines both autoencoders and recurrent neural networks, exploiting the former to reduce the dimensionality of the input and the latter to capture temporal information.

For Node2Vec, we considered different values of its parameters, i.e., $p \in \{0.25, 1, 4\}$ and $q \in \{0.25, 0.50, 1\}$, suggested by the authors according to the results of the experiments reported and discussed in [16], while the considered embedding size, number of random walks and their length have been set to the same values adopted for LP-ROBIN.

Since all the variants of *dyngraph2vec* are based on neural networks, analogously to LP-ROBIN and for a fair comparison, we set the number of epochs equal to 200. Moreover, we set its lookback parameter (i.e., the number of previous snapshots to consider during the training phase) equal to the number of previous snapshots (as shown in Table 2). Moreover, in order to properly compare it with LP-ROBIN, and exclude possible influences deriving from the architecture of the neural network in the embedding phase, we set the number of neurons of its hidden layers equal to those considered for LP-ROBIN in the embedding phase.

5.3. Results and discussion

In this section, we show and discuss the results obtained by LP-ROBIN and its competitors on the considered datasets.

The first analysis is aimed at investigating the sensitivity of LP-ROBIN to the values of its parameters. In Figure 5 we report the results in terms of AUC of an ablation study. From the graphs, it is possible to observe that none of the parameters has a strong influence on the AUC performance, especially the number (i.e., n_rw) and the length (i.e., l_rw) of the random walks. This is a positive aspect that indicates that LP-ROBIN identifies reliable embeddings even for a small number of short walks. An additional observation is that there are no parameter configurations that lead to the best results over all the datasets. One exception is represented by the embedding size (i.e., d), which typically shows the best results for $d = 256$. This means that the number of nodes in the hidden layers influences the effectiveness of the method. As for the aggregation method, the best results are achieved with the "weighted sum", especially for BitcoinOTC and WikiTalk. This indicates that the normalization introduced in the "weighted average" is not always beneficial and can lead to flattening the representation of the links.

Table 3: Parameter values selected by the grid search.

System	**Parameter**	**CollegeMsg**	**BitcoinOTC**	**WikiTalk**
LP-ROBIN	n_rw	20	20	10
	l_rw	10	10	20
	d	128	256	256
	agg	$w\_average$	w_sum	w_sum
	α	0.5	1.0	0.5
Node2Vec	n_rw	10	20	10
	l_rw	20	10	10
	d	256	256	64
	p	4	1	1
	q	0.25	0.25	1
DynAE	d	256	128	64
DynRNN	d	64	64	64
DynAERNN	d	256	256	64

As for α, the best results are obtained with $\alpha = 0.5$ and $\alpha = 1$. This is probably due to the speed of evolution of the network structure over time, which requires giving more importance to the most recent data[7].

We performed a similar ablation study for competitor systems (see Figures 6-9). For Node2Vec, we can draw conclusions similar to those of LP-ROBIN. This is a clear indication that the subsequent link prediction task is not negatively affected by the values of the parameters adopted for the embedding strategy. This observation also holds when comparing the learning strategies adopted for LP-ROBIN (incremental) and Node2Vec (batch): we do not observe any correlation between the parameter values and the adopted learning strategy.

In Table 3 we report the combination of the parameter values that led to the best results in terms of AUC. In the remaining of this section, we will use such configurations for the comparative evaluation.

[7]We remark that with $\alpha = 1$ we do not forget the past, but we represent the nodes (and links) only according to the random walks observed at the current snapshot.

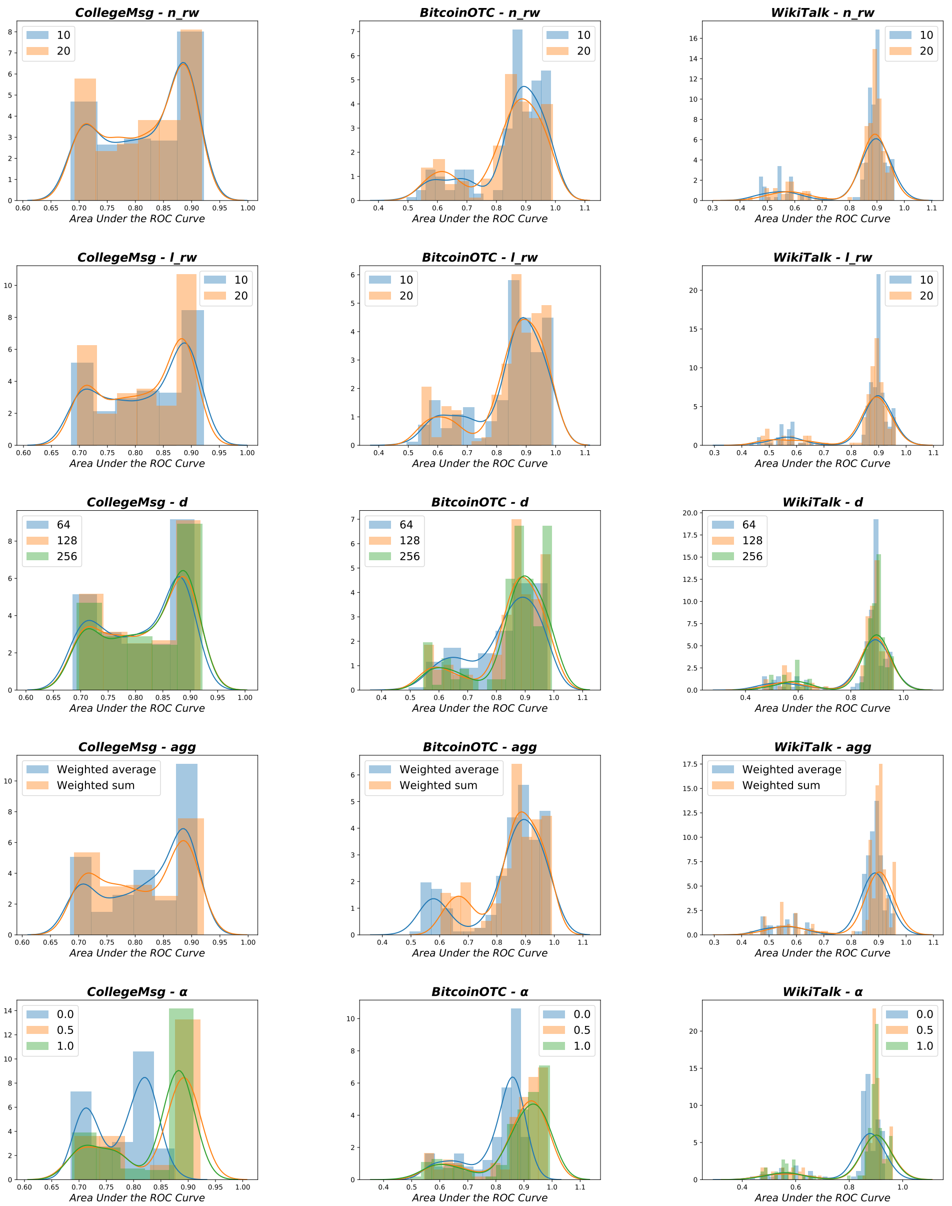


Figure 5: Results of an ablation study over LP-ROBIN parameters in terms of AUC.

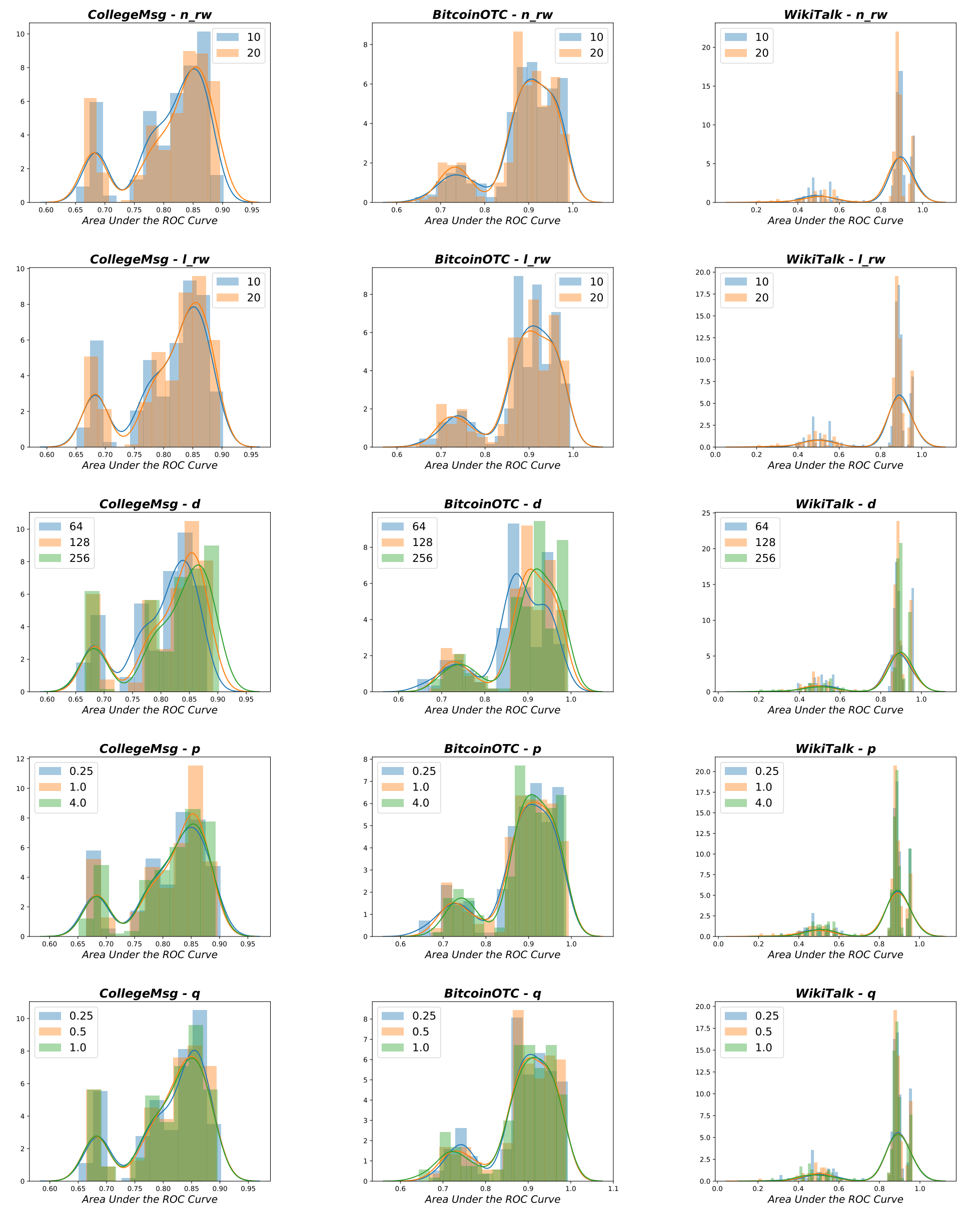


Figure 6: Results of an ablation study over Node2Vec parameters in terms of AUC.

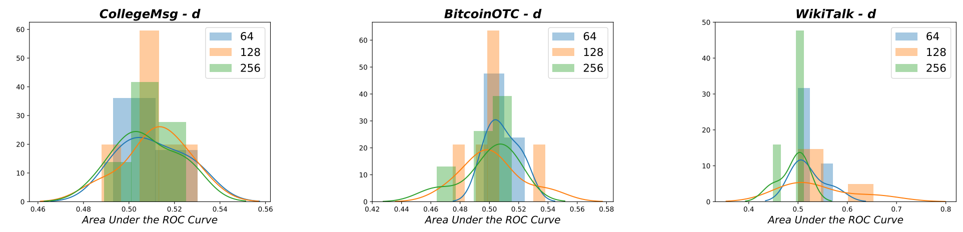


Figure 7: Results of an ablation study over DynAE parameters in terms of AUC.

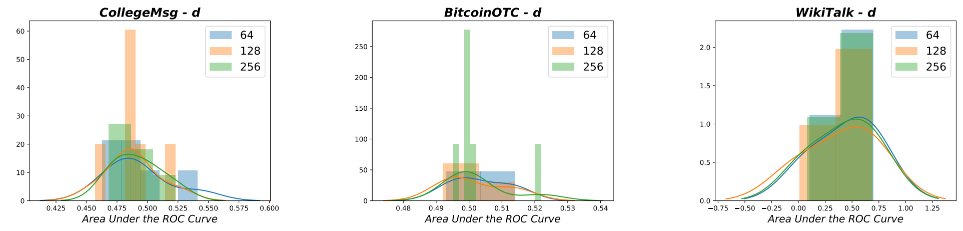


Figure 8: Results of an ablation study over DynRNN parameters in terms of AUC.

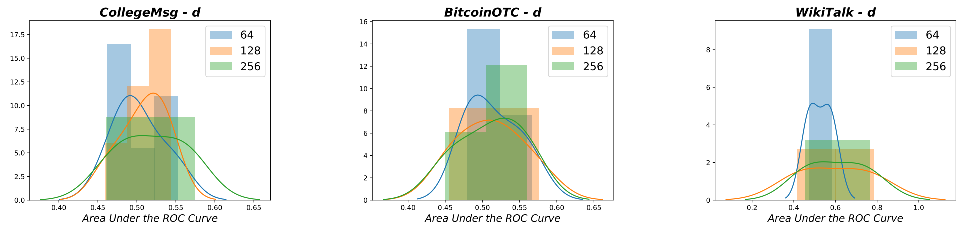


Figure 9: Results of an ablation study over DynAERNN parameters in terms of AUC.

We report the AUC and F1-score results in Table 4 and in Figure 10. It is possible to observe that LP-ROBIN leads to the best overall results, both in terms of AUC and in terms of F1-Score. There are few cases in which Node2Vec is able to outperform LP-ROBIN, but only for the first snapshots, while on the subsequent snapshots, LP-ROBIN clearly shows its superiority. This phenomenon emphasizes that the dynamic nature of LP-ROBIN (in both the construction of the embedding and in the learning strategy) is beneficial, since it is able to properly weight the contribution of new data with respect to past data. On the contrary, the static nature of Node2Vec gives the same importance to all the data, without giving more importance to recent data and without properly dealing with possible concept drift phenomena.

LP-ROBIN and state-of-the-art competitors generally perform significantly better than baselines. This is due to the fact that baselines have a local view of the network structure, since they focus on the neighborhood of nodes. On the contrary, relying on random walks, generated from specific nodes and links, leads to having a wider view of the network structure. For this reason, LP-ROBIN can be considered a *semi-local* approach since, on the other hand, it does not incur the strong computational issues typical of *global* approaches that extensively analyze the whole network (e.g., [20, 6]).

Compared with DynAE, DynRNN, and DynAERNN, we can observe a clear superiority of LP-ROBIN. Although all the methods exploit neural networks for the embedding step, the strategy we adopt in LP-ROBIN appears to be much more effective. This might be due to the fact that DynAE, DynRNN, and DynAERNN directly compress the node representation into d features starting from a full row of the adjacency matrix. On the contrary, in LP-ROBIN we reduce this problem since we start from l_rw (with $l\_rw \ll |V^t|$, where $|V^t|$ is the number of nodes at the t-th snapshot). Moreover, this aspect also makes DynAE, DynRNN, and DynAERNN much less efficient. Indeed, they were not able to complete the experiments causing out-of-memory errors for the last few snapshots of the WikiTalk dataset.

As for the learning times (see Table 5), while the baselines are very efficient at the price of poor accuracy, the comparison with Node2Vec shows that the incremental approach, in addition to providing accurate results, is also more efficient. An exception is represented by WikiTalk, where the lower Node2Vec running times are motivated by the shorter random walks of its best configuration (see Table 3). Finally, DynAE, DynRNN and DynAERNN do not show competitive running times, probably for the same reason mentioned before, i.e., they exploit high-dimensional node representations.

Table 4: F1-score and AUC results for all the considered snapshots. The best F1-score and AUC results for each dataset and snapshot are reported in bold.

		G^1		G^2		G^3		G^4		G^5		G^6	
	System	F1	AUC	F1	AUC	F1	AUC	F1	AUC	F1	AUC	F1	AUC
CollegeMsg	*LP-ROBIN*	0.550	**0.701**	0.656	0.769	**0.813**	**0.890**	**0.769**	**0.905**	0.734	**0.896**	**0.773**	**0.904**
	AAI	0.002	0.577	0.004	0.665	0.026	0.698	0.011	0.693	0.042	0.683	0.022	0.612
	CN	0.000	0.572	0.004	0.658	0.036	0.694	0.004	0.692	0.031	0.681	0.022	0.607
	JACCARD	0.002	0.559	0.000	0.634	0.000	0.665	0.000	0.665	0.055	0.655	0.000	0.581
	RAI	0.000	0.577	0.003	0.666	0.017	0.699	0.011	0.692	0.031	0.682	0.022	0.615
	Node2Vec	0.514	0.685	0.648	**0.794**	0.750	0.859	0.731	0.873	**0.745**	0.865	0.768	0.868
	DynAE	**0.667**	0.503	**0.667**	0.489	0.667	0.523	0.667	0.503	0.667	0.504	0.667	0.525
	DynRNN	0.646	0.463	0.612	0.491	0.634	0.478	0.663	0.497	0.662	0.484	0.667	0.541
	DynAERNN	0.609	0.460	0.556	0.498	0.402	0.490	0.459	0.574	0.471	0.557	0.485	0.535
BitcoinOTC	*LP-ROBIN*	0.573	0.674	**0.777**	0.857	**0.853**	0.913	**0.866**	**0.929**	**0.939**	**0.981**	**0.901**	0.992
	AAI	0.114	0.588	0.031	0.763	0.017	0.763	0.029	0.794	0.035	0.907	0.143	0.921
	CN	0.337	0.584	0.077	0.761	0.037	0.761	0.025	0.792	0.086	0.904	0.143	0.921
	JACCARD	0.000	0.522	0.000	0.739	0.000	0.746	0.007	0.777	0.000	0.884	0.098	0.917
	RAI	0.092	0.584	0.019	0.762	0.004	0.763	0.017	0.793	0.013	0.907	0.098	0.921
	Node2Vec	**0.667**	**0.761**	0.769	**0.882**	0.802	**0.920**	0.841	0.927	0.916	0.970	0.870	**0.993**
	DynAE	**0.667**	0.502	0.667	0.494	0.667	0.500	0.667	0.475	0.667	0.503	0.667	0.538
	DynRNN	0.656	0.514	0.655	0.493	0.634	0.497	0.662	0.504	0.666	0.499	0.667	0.513
	DynAERNN	0.664	0.515	0.600	0.473	0.379	0.518	0.412	0.554	0.366	0.450	0.500	0.560
WikiTalk	*LP-ROBIN*	0.571	0.551	**0.822**	**0.909**	**0.780**	**0.889**	**0.794**	**0.894**	**0.805**	**0.918**	**0.887**	**0.963**
	AAI	0.000	0.500	0.007	0.789	0.005	0.797	0.001	0.762	0.001	0.692	0.000	0.735
	CN	**0.923**	**0.929**	0.009	0.770	0.009	0.790	0.002	0.767	0.000	0.700	0.000	0.740
	JACCARD	**0.923**	**0.929**	0.055	0.477	0.055	0.720	0.062	0.748	0.054	0.697	0.046	0.732
	RAI	0.444	**0.929**	0.011	0.792	0.000	0.801	0.000	0.769	0.000	0.700	0.000	0.742
	Node2Vec	0.556	0.510	0.755	0.899	0.724	0.862	0.684	0.869	0.738	0.884	0.846	**0.963**
	DynAE	0.667	0.571	0.667	0.513	0.667	0.502	0.667	0.500	-	-	-	-
	DynRNN	0.471	0.102	0.667	0.699	0.639	0.529	-	-	-	-	-	-
	DynAERNN	0.182	0.582	0.574	0.566	0.496	0.472	0.486	0.483	-	-	-	-

Table 5: Running times (in seconds) for the training phase.

	System	G^1	G^2	G^3	G^4	G^5	G^6
CollegeMsg	*LP-ROBIN*	39	65	23	11	7	7
	AAI	1	2	1	1	1	1
	CN	1	2	1	1	1	1
	JACCARD	1	2	1	1	1	1
	RAI	1	1	1	1	1	1
	Node2Vec	18	72	86	91	90	95
	DynAE	66	95	185	245	324	424
	DynRNN	268	278	943	1,751	3,112	4,788
	DynAERNN	86	132	426	811	1,404	2,167
BitcoinOTC	*LP-ROBIN*	16	58	22	28	31	9
	AAI	1	1	1	1	2	1
	CN	1	2	2	2	1	1
	JACCARD	1	1	1	2	1	1
	RAI	1	1	1	2	1	1
	Node2Vec	6	85	151	240	268	270
	DynAE	322	359	638	962	1,437	1,748
	DynRNN	4,534	5,674	18,102	36,713	63,796	84,698
	DynAERNN	420	478	1,597	3,077	5,368	28,072
WikiTalk	*LP-ROBIN*	12	37	145	460	3,238	24,927
	AAI	2	2	16	75	384	4,926
	CN	1	1	3	18	108	1,876
	JACCARD	1	2	4	21	119	1,994
	RAI	1	1	4	18	107	1,902
	Node2Vec	5	15	53	238	1,092	5,846
	DynAE	4,445	7,965	11,195	15,677	-	-
	DynRNN	4,792	13,146	25,556	-	-	-
	DynAERNN	4,983	13,615	26,184	43,948	-	-

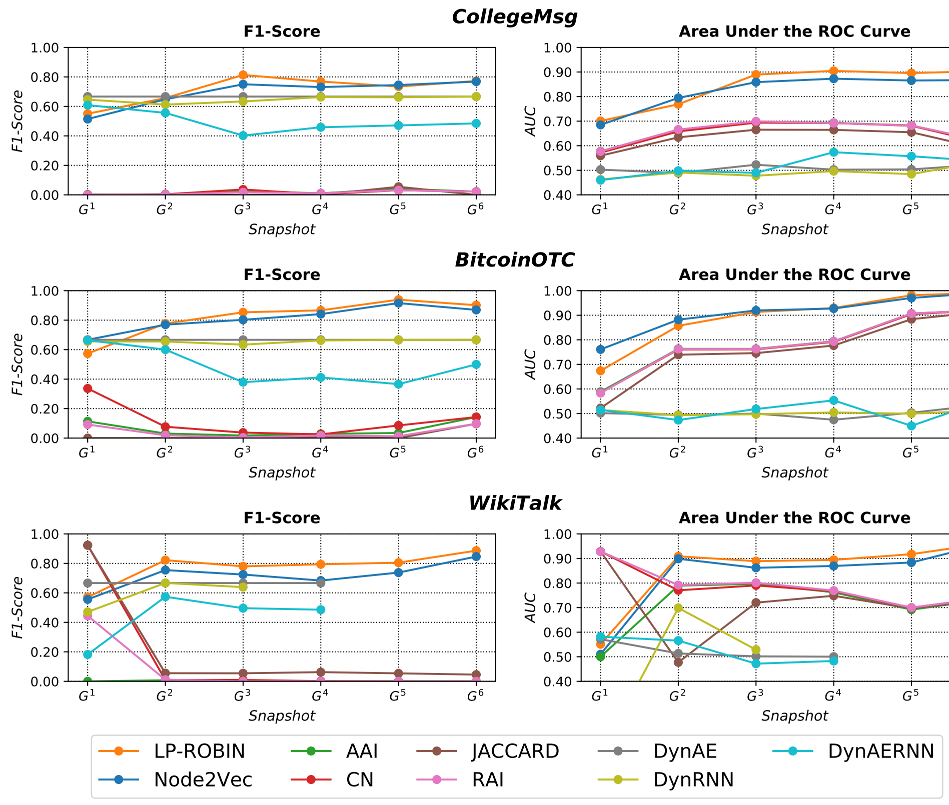


Figure 10: Results obtained with the different datasets in terms of F1-score and AUC, with the best configurations.

6. Conclusion

In this paper, we proposed LP-ROBIN, a novel method that is able to solve link prediction tasks on dynamic networks. LP-ROBIN is able to predict new links that will appear in the next snapshot of the network, exploiting an embedding technique based on random walks.

LP-ROBIN works incrementally, in order to dynamically capture the changes. The innovative aspect is that the network embedding and the link prediction model are incrementally and coherently updated when new snapshots arrive, with the advantage of adapting to the possible structural changes in the network.

Our experimental evaluation showed that LP-ROBIN is able to outperform baseline and state-of-the-art competitors, achieving better results in terms of both AUC and F1-score. Moreover, LP-ROBIN effectively and efficiently exploits the different contributions coming from new data with respect to past data, in an incremental and dynamic fashion.

For future work, we will extend LP-ROBIN to predict not only the appearance of new links, but also their removal over time. A possible solution could consist in constructing random walks virtually traversing non-existent links, and feeding the sparse autoencoder with them to generate their latent representations. Then, we could exploit their *complementary/negated* representation to represent nodes and, accordingly, links.

Moreover, we will aim to extend the applicability of LP-ROBIN to a more general setting, involving heterogeneous types of nodes and links as well as possible weights on the links. This would give us the opportunity to apply LP-ROBIN to more complex network data, such as biological networks.

Availability

The system, the datasets and all the results are publicly available at: `https://figshare.com/projects/LP-ROBIN/87188`.

Acknowledgment

The authors acknowledge the support of the European Commission through the H2020 Projects "CounteR - Privacy-First Situational Awareness Platform for Violent Terrorism and Crime Prediction, Counter Radicalisation and Citizen Protection" (Grant N. 101021607), and "IMPETUS - Intelligent

Management of Processes, Ethics and Technology for Urban Safety" (Grant. N. 883286). Dr. Gianvito Pio acknowledges the support of Ministry of Education, Universities and Research (MIUR) through the project "Big Data Analytics", AIM 1852414, activity 1, line 1.

References

[1] Bergstra, J., Yamins, D., Cox, D.D., 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in: Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013, JMLR.org. pp. 115–123.

[2] Bifet, A., Gavaldà, R., 2007. Learning from time-changing data with adaptive windowing, in: In SIAM International Conference on Data Mining, pp. 443–448.

[3] Brzezinski, D., Stefanowski, J., 2018. Ensemble classifiers for imbalanced and evolving data streams, in: Data Mining in Time Series and Streaming Databases, Chapter 3, pp. 44–68.

[4] Buono, N.D., Pio, G., 2015. Non-negative matrix tri-factorization for co-clustering: An analysis of the block matrix. Information Sciences 301, 13 – 26.

[5] Ceci, M., Corizzo, R., Japkowicz, N., Mignone, P., Pio, G., 2020. ECHAD: embedding-based change detection from multivariate time series in smart grids. IEEE Access 8, 156053–156066.

[6] Chebotarev, P.Y., Shamis, E., 1997. The matrix-forest theorem and measuring relations in small social group. Automation and Remote Control 58, 1505–1514.

[7] Chi, K., Yin, G., Dong, Y., Dong, H., 2019. Link prediction in dynamic networks based on the attraction force between nodes. Knowledge-Based Systems 181, 104792.

[8] Chiu, C., Zhan, J., 2018. Deep learning for link prediction in dynamic networks using weak estimators. IEEE Access 6, 35937–35945.

[9] Comito, C., 2020. Next: A framework for next-place prediction on location based social networks. Knowledge-Based Systems 204, 106205.

[10] Dakiche, N., Tayeb, F.B.S., Slimani, Y., Benatchba, K., 2019. Tracking community evolution in social networks: A survey. Information Processing & Management 56, 1084 – 1102.

[11] Du, L., Wang, Y., Song, G., Lu, Z., Wang, J., 2018. Dynamic Network Embedding : An Extended Approach for Skip-gram based Network Embedding, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, Stockholm, Sweden. pp. 2086–2092.

[12] Fakhraei, S., Foulds, J., Shashanka, M., Getoor, L., 2015. Collective spammer detection in evolving multi-relational social networks, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, New York, NY, USA. p. 1769–1778.

[13] Glorot, X., Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks, in: AISTATS, pp. 249–256.

[14] Glorot, X., Bordes, A., Bengio, Y., 2011. Deep sparse rectifier neural networks, in: Gordon, G.J., Dunson, D.B., Dudík, M. (Eds.), Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011, JMLR.org. pp. 315–323.

[15] Goyal, P., Chhetri, S.R., Canedo, A., 2020. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. Knowl. Based Syst. 187.

[16] Grover, A., Leskovec, J., 2016. node2vec: Scalable feature learning for networks. KDD : proceedings. International Conference on Knowledge Discovery & Data Mining 2016, 855–864.

[17] Haddad, M., BOTHOREL, C., Lenca, P., Bedart, D., 2019. TemporalNode2vec: Temporal Node Embedding in Temporal Networks, in: COMPLEX NETWORKS 2019 : 8th International Conference on Complex Networks and their Applications, Lisbon, Portugal. pp. 891–902.

[18] Haghani, S., Keyvanpour, M.R., 2017. Temporal link prediction: Techniques and challenges. Computer science and information technologies .

[19] Hess, S., Pio, G., Hochstenbach, M.E., Ceci, M., 2021. BROCCOLI: overlapping and outlier-robust biclustering through proximal stochastic gradient descent. Data Min. Knowl. Discov. 35, 2542–2576.

[20] Jeh, G., Widom, J., 2002. Simrank: A measure of structural-context similarity, in: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA. pp. 538–543.

[21] Jin, D., Li, B., Jiao, P., He, D., Zhang, W., 2019. Network-Specific Variational Auto-Encoder for Embedding in Attribute Networks, in: IJCAI, pp. 2663–2669.

[22] Kim, J., Hastak, M., 2018. Social network analysis: Characteristics of online social networks after a disaster. International Journal of Information Management 38, 86–96.

[23] Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. CoRR abs/1412.6980.

[24] Kumar, S., Hooi, B., Makhija, D., Kumar, M., Faloutsos, C., Subrahmanian, V., 2018. REV2: Fraudulent User Prediction in Rating Platforms, in: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining - WSDM '18, ACM Press, Marina Del Rey, CA, USA. pp. 333–341.

[25] Kumar, S., Spezzano, F., Subrahmanian, V.S., Faloutsos, C., 2016. Edge Weight Prediction in Weighted Signed Networks, in: 2016 IEEE 16th International Conference on Data Mining (ICDM), IEEE, Barcelona, Spain. pp. 221–230.

[26] Kumar, S., Zhang, X., Leskovec, J., 2019. Predicting dynamic embedding trajectory in temporal interaction networks, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Association for Computing Machinery, New York, NY, USA. p. 1269–1278.

[27] Leskovec, J., Huttenlocher, D.P., Kleinberg, J.M., 2010. Governance in social media: A case study of the wikipedia promotion process, in: Cohen, W.W., Gosling, S. (Eds.), Proceedings of the Fourth International Conference on Weblogs and Social Media, ICWSM 2010, Washington, DC, USA, May 23-26, 2010, The AAAI Press. pp. 98–105.

[28] Liben-Nowell, D., Kleinberg, J., 2007. The Link Prediction Problem for Social Networks. Journal of the American Society for Information Science and Technology , 19.

[29] Mahdavi, S., Khoshraftar, S., An, A., 2018. dynnode2vec: Scalable Dynamic Network Embedding, in: 2018 IEEE International Conference on Big Data (Big Data), IEEE, Seattle, WA, USA. pp. 3762–3765.

[30] Mikolov, T., Chen, K., Corrado, G.S., Dean, J., 2013. Efficient estimation of word representations in vector space. CoRR abs/1301.3781.

[31] Mohamed, E.A., Zaki, N., Marjan, M., 2019. Current Trends and Challenges in Link Prediction Methods in Dynamic Social Networks : A Literature Review. Advances in Science, Technology and Engineering Systems Journal 4, 244–254.

[32] Nguyen, G.H., Lee, J.B., Rossi, R.A., Ahmed, N.K., Koh, E., Kim, S., 2018. Continuous-Time Dynamic Network Embeddings, in: Companion of the The Web Conference 2018 on The Web Conference 2018 - WWW '18, ACM Press, Lyon, France. pp. 969–976.

[33] Noble, J., Adams, N., 2018. Real-time dynamic network anomaly detection. IEEE Intelligent Systems 33, 5–18.

[34] Panzarasa, P., Opsahl, T., Carley, K.M., 2009. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. Journal of the American Society for Information Science and Technology 60, 911–932.

[35] Paranjape, A., Benson, A.R., Leskovec, J., 2017. Motifs in temporal networks, in: de Rijke, M., Shokouhi, M., Tomkins, A., Zhang, M. (Eds.), Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM 2017, Cambridge, United Kingdom, February 6-10, 2017, ACM. pp. 601–610.

[36] 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. pp. 701–710.

[37] Rahman, M., Hasan, M.A., 2016. Link prediction in dynamic networks usingÂ graphlet, in: Frasconi, P., Landwehr, N., Manco, G., Vreeken, J. (Eds.), Machine Learning and Knowledge Discovery in Databases, Springer International Publishing, Cham. pp. 394–409.

[38] Ramesh, A., Rodriguez, M., Getoor, L., 2017. Multi-relational influence models for online professional networks, in: Proceedings of the International Conference on Web Intelligence, Association for Computing Machinery, New York, NY, USA. p. 291–298.

[39] Ranzato, M., Boureau, Y., LeCun, Y., 2007. Sparse feature learning for deep belief networks, in: Platt, J.C., Koller, D., Singer, Y., Roweis, S.T. (Eds.), Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007, Curran Associates, Inc.. pp. 1185–1192.

[40] Rekatsinas, T., Ghosh, S., Mekaru, S.R., Nsoesie, E.O., Brownstein, J.S., Getoor, L., Ramakrishnan, N., 2017. Forecasting rare disease outbreaks from open source indicators. Statistical Analysis and Data Mining: The ASA Data Science Journal 10, 136–150.

[41] Soundarajan, S., Hopcroft, J., 2012. Using community information to improve the precision of link prediction methods, in: Proceedings of the 21st International Conference on World Wide Web, Association for Computing Machinery, Lyon, France. pp. 607–608.

[42] Stojanova, D., Ceci, M., Appice, A., Dzeroski, S., 2012. Network regression with predictive clustering trees. Data Min. Knowl. Discov. 25, 378–413.

[43] 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, International

World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE. p. 1067–1077.

[44] Trivedi, R., Farajtabar, M., Biswal, P., Zha, H., 2019. DyRep: Learning Representations over Dynamic Graphs, in: ICLR, pp. 1–25.

[45] Verma, J., Gupta, S., Mukherjee, D., Chakraborty, T., 2019. Heterogeneous Edge Embedding for Friend Recommendation, in: Azzopardi, L., Stein, B., Fuhr, N., Mayr, P., Hauff, C., Hiemstra, D. (Eds.), Advances in Information Retrieval. Springer International Publishing, Cham. volume 11438, pp. 172–179.

[46] Wang, D., Pedreschi, D., Song, C., Giannotti, F., Barabasi, A.L., 2011. Human mobility, social ties, and link prediction, in: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, New York, NY, USA. p. 1100–1108.

[47] Yang, Y., Chen, H., Shao, J., 2019. Triplet Enhanced AutoEncoder: Model-free Discriminative Network Embedding, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, Macao, China. pp. 5363–5369.

[48] Yu, W., Cheng, W., Aggarwal, C.C., Chen, H., Wang, W., 2017. Link prediction with spatial and temporal consistency in dynamic networks, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence, AAAI Press. p. 3343–3349.

[49] kui Zhou, L., Yang, Y., Ren, X., Wu, F., Zhuang, Y., 2018. Dynamic network embedding by modeling triadic closure process, in: AAAI, pp. 571–578.

[50] Zhou, T., Lu, L., Zhang, Y.C., 2009. Predicting Missing Links via Local Information. The European Physical Journal B 71, 623–630. ArXiv: 0901.0553.