Scalable Representation Learning for Dynamic Heterogeneous Information Networks via Metagraphs

YANG FANG, XIANG ZHAO, PEIXIN HUANG, and WEIDONG XIAO, National University of Defense Technology, China
MAARTEN DE RIJKE, University of Amsterdam, The Netherlands

Content representation is a fundamental task in information retrieval. Representation learning is aimed at capturing features of an information object in a low-dimensional space. Most research on representation learning for heterogeneous information networks (HINs) focuses on static HINs. In practice, however, networks are dynamic and subject to constant change. In this article, we propose a novel and scalable representation learning model, M-DHIN, to explore the evolution of a dynamic HIN. We regard a dynamic HIN as a series of snapshots with different time stamps. We first use a static embedding method to learn the initial embeddings of a dynamic HIN at the first time stamp. We describe the features of the initial HIN via metagraphs, which retains more structural and semantic information than traditional path-oriented static models. We also adopt a complex embedding scheme to better distinguish between symmetric and asymmetric metagraphs. Unlike traditional models that process an entire network at each time stamp, we build a so-called *change dataset* that only includes nodes involved in a triadic closure or opening process, as well as newly added or deleted nodes. Then, we utilize the above metagraph-based mechanism to train on the change dataset. As a result of this setup, M-DHIN is scalable to large dynamic HINs since it only needs to model the entire HIN once while only the changed parts need to be processed over time. Existing dynamic embedding models only express the existing snapshots and cannot predict the future network structure. To equip M-DHIN with this ability, we introduce an LSTM-based deep autoencoder model that processes the evolution of the graph via an LSTM encoder and outputs the predicted graph. Finally, we evaluate the proposed model, M-DHIN, on real-life datasets and demonstrate that it significantly and consistently outperforms state-of-the-art models.

CCS Concepts: • **Information systems** → **Network data models;**

Additional Key Words and Phrases: Dynamic heterogeneous information network, network representation learning, metagraph

This work was partially supported by NSFC under grant Nos. 61872446 and U19B2024, NSF of Hunan Province under grant No. 2019JJ20024, the Science and Technology Innovation Program of Hunan Province under grant No. 2020RC4046, and the Hybrid Intelligence Center, a 10-year program funded by the Dutch Ministry of Education, Culture and Science through the Netherlands Organisation for Scientific Research, https://hybrid-intelligence-centre.nl. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.
Authors' addresses: Y. Fang, X. Zhao (corresponding author), P. Huang, and W. Xiao, National University of Defense Technology, Changsha, 410072, China; emails: {fangyang12, xiangzhao, huangpeixin15, wdxiao}@nudt.edu.cn; M. de Rijke, University of Amsterdam, Amsterdam, 1090GH, The Netherlands; email: m.derijke@uva.nl.

1046-8188/2022/03-ART64 $15.00
https://doi.org/10.1145/3485189

ACM Reference format:
Yang Fang, Xiang Zhao, Peixin Huang, Weidong Xiao, and Maarten de Rijke. 2022. Scalable Representation Learning for Dynamic Heterogeneous Information Networks via Metagraphs. *ACM Trans. Inf. Syst.* 40, 4, Article 64 (March 2022), 27 pages.
https://doi.org/10.1145/3485189

1 INTRODUCTION

A heterogeneous information network (HIN) is a network that evolves constantly, with nodes and edges of multiple types. In practice, most networks are dynamic HINs, such as social networks and bibliographic networks. Hence, dynamic HINs serve as a more expressive tool to model information-rich problems than static networks.

To be able to process a network in a machine learning context, *network representation learning*, also known as network embedding learning, which is aimed at obtaining an embedding of a network into a low-dimensional space, has been investigated extensively. Most research focuses on static information networks. Classic network embedding models [e.g., 15, 22, 29] utilize random walks to explore static homogeneous networks. To represent static heterogeneous networks, many models have been proposed based on *metapaths*, using different mechanisms to model the relationships between HIN nodes [7, 11, 16]. Unlike static network embeddings, the techniques for dynamic HINs need to be incremental and scalable so as to be able to handle network evolutions effectively. This renders most existing static embedding models, which need to process the entire network step by step, unsuitable and inefficient. To address this issue, we propose a novel dynamic HIN embedding model, named M-DHIN, which provides a scalable method to capture the features of a dynamic HIN via *metagraphs*. We first learn the initial embeddings of the whole HIN at the first time stamp. Leveraging random walks or metapaths as is done by traditional network embedding methods is insufficient to describe the neighborhood structure of a node in a complete manner [10]. Here, we propose metagraphs to capture the structural information of a HIN. A metagraph is a subgraph of node types with edge types in between, which captures the connecting subgraph of two node types, and hence, completely retains the neighborhood structure of nodes; some toy examples are provided in Figure 1. When training the model, we observe that the structure of metagraphs can be either symmetric or asymmetric, as illustrated in Figures 1(a) and 1(b), respectively. To better represent HIN nodes, we incorporate a complex space-oriented embedding scheme [30] to handle symmetric and asymmetric relationships between nodes. In complex space, the entries in the embedding vectors of nodes are complex-valued; that is, we separate the vectors of nodes into real and imaginary parts.

We employ triadic structures, i.e., sets of three nodes, to capture evolutions in a dynamic HIN. Triadic structures are fundamental building blocks in a network [33] and we describe their evolution as *triadic closure* and *opening processes* [37]. Figure 2 is a toy example of a dynamic HIN, where U denotes users and T denotes topics subscribed to by users. From time stamp t to $t+1$, we observe a newly arriving edge from U_2 to T_1, meaning that U_2 now subscribes to T_1, which is a triadic closure process. Moving from time stamp $t+1$ to $t+2$, the edge from U_2 to U_3 disappears, meaning that U_2 unfollows U_3, which is a triadic opening process. Unlike traditional static methods that process the entire network, we introduce a changed training dataset that only includes the nodes involved in triadic closure or opening processes in between two consecutive time points, as well as newly added and deleted nodes with their neighbor nodes. That is, after obtaining the initial complex embeddings, we only update the representations of nodes in the change dataset. When training, we process the metagraph instances in the change dataset. For example, in Figure 2, from time t to $t+1$, only U_1, U_2, and T_1 will be included in the change dataset, and they will become

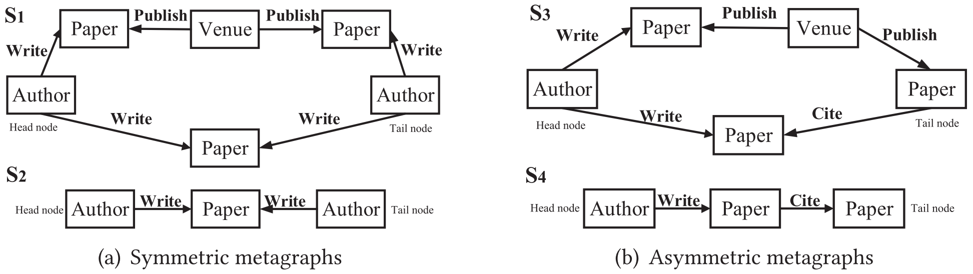


Fig. 1. Examples of metagraphs.

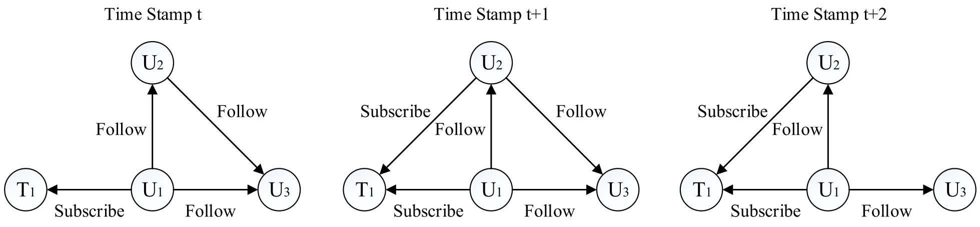


Fig. 2. An example of a dynamic HIN model of a social network.

a metagraph instance to be trained. From time $t + 1$ to $t + 2$, only U_1, U_2, and U_3 will be included. Then we apply the complex embedding mechanism again on the change datasets to update the network embeddings. By doing so, we represent every snapshot of a dynamic HIN over time, thus capturing its dynamic features. Note that metagraphs will not change along with the HIN, since they only contain node and edge types acting like a relation. We only update metagraph instances.

The network embeddings generated above can be leveraged to predict the future network structure. Most existing methods only use the current snapshot to predict a future network, ignoring information about the network's historical evolution. We introduce an LSTM-based deep autoencoder to utilize all history snapshots for prediction. Concretely, we use a metagraph to form the adjacency matrix of a node and then collect all the history metagraphs as inputs. The LSTM encoder helps the model to capture the history of nodes. Finally, the decoder outputs the predicted graph.

To evaluate the proposed M-DHIN, we perform experiments on six tasks: link prediction, changed link prediction, node classification, node prediction, graph reconstruction, and anomaly detection. The experimental results we obtain show that M-DHIN consistently and significantly outperforms state-of-the-art models, which verifies the effectiveness of M-DHIN on representing a dynamic HIN.

This article is an extension of our previous work [10]. In our previous work, we focused on representation learning for static HINs via a metagraph-based complex mechanism. Here, we make the non-trivial extension to dynamic HINs. The main contributions of the article can be summarized as follows:

- We propose a novel dynamic network embedding model, M-DHIN, that learns representations of every snapshot of a dynamic HIN via metagraph-based complex embeddings on so-called change datasets.
- We propose an LSTM-based deep autoencoder mechanism to enable M-DHIN to predict the future network via history structure evolutions.

- We evaluate the performance of M-DHIN on six tasks and compare it with state-of-the-art models; the experimental results show that M-DHIN consistently and significantly outperforms the state of the art.

The rest of the article is structured as follows. We introduce related work in Section 2 and present the preliminaries in Section 3. Then we present M-DHIN, together with a theoretical analysis, in Section 4. Next, we introduce our experimental setup in Section 5 and present our experimental results in Section 6. We conclude in Section 7.

2 RELATED WORK

2.1 Static Network Embeddings

Most previous research on representation learning for information networks focuses on representation learning for static networks. Network embedding methods originated from dimensionality reduction approaches [3, 6, 25, 31], which compute eigenvectors of the affinity graph constructed via feature vectors for nodes. The graph factorization model [1] generates network embeddings by factorizing the adjacency matrix. These models cannot capture the global network structure, they have problems with data sparsity, and they come with high computational costs [29].

To preserve the local and global structure of a network, some research proposes to utilize random walks or paths of a network. DeepWalk [22] first extracts random walks from a network and then applies the SkipGram model on random walks to generate network embeddings. LINE [29] employs both first-order and second-order proximities so as to capture both local and neighboring structures. node2vec [15] harnesses a biased random walk sampling strategy to represent the network structure, which is more expressive than DeepWalk. GrapRep [5] explores the high-order proximities of DeepWalk but suffers from the expensive computation of the power of a matrix and **singular value decomposition (SVD)** during the training process. SDNE [31] preserves the non-linear structural information of a network by using a semi-supervised deep model.

The aforementioned approaches are designed for homogeneous networks. There are many dedicated methods specifically expressing features of heterogeneous networks. PTE [28] utilizes the conditional probability of nodes of one specific type generated by nodes of another type, then computes the conditional distribution, which should be close to its empirical distribution. metapath2vec [7] uses a heterogeneous SkipGram by setting the context window limited to one type. HINE [16] adopts a proximity measure based on metapaths and preserves proximity by minimizing the distance between nodes' joint probability defined by `sigmoid` and empirical probabilities. Esim [27] proposes a metapath-guided sampling strategy to improve the model efficiency while its objective function is partially defined. HIN2Vec [11] utilizes Hadamard multiplication of nodes and metapaths to exploit features of a HIN, but the symmetric and asymmetric structures inside a HIN are largely overlooked.

In a recent publication the term MetaGraph2Vec [35] was coined. However, this publication utilizes heterogeneous SkipGrams and combines metapaths to form so-called "metagraphs," which, in essence, is a path-oriented model. Additionally, many recent models harness graph neural networks to explore the structural information of networks [12, 18, 34]. HetGNN [34] groups nodes according to their type and applies Bi-LSTM on each group. GEM [20] learns representations from heterogeneous account-device graphs to detect the malicious account in a static setting. HAN [32] proposes to represent a heterogeneous network based on the hierarchical attention, including node-level and semantic-level attentions. Meta-GNN [26] makes use of metagraphs to extract diverse semantics; however, it does not take symmetric and asymmetric metagraphs into consideration. PIdentifier [9] designs different metagraphs to formulate the relatedness between nodes, which are task specific. It does not consider the symmetric and asymmetric attributes of metagraphs either, which may hinder its performance.

In our previous work [10] we propose a static HIN embedding model that first adopts the GRAMI algorithm to generate metagraphs and then harnesses the complex embedding scheme to explore the features of a HIN. In contrast, in this work, we focus on embedding dynamic HINs and use the method from our previous work to train the initial snapshot of a HIN.

2.2 Dynamic Network Embeddings

Recently, dynamic network embeddings have begun to draw more attention. Some early research tries to extend static network embedding models by adding regularizations [36, 38]. Li et al. [19] borrow the idea of spectral clustering, which is widely used in computer vision tasks. It learns a robust dynamic affinity matrix using multiple features and decides a set of optimal projection matrices. Luo et al. [21] realize unsupervised feature selection by building an adaptive reconstruction graph meanwhile considering its multiconnected-components (multi-cluster) structure. DynGEM [14] proposes an autoencoder mechanism to capture the dynamics, but it only utilizes one previous time step snapshot to generate the current graph embeddings. TIMERS [36] leverages incremental SVD to update the network embeddings and returns SVD values when the error is beyond the threshold. DynamicTraid [37] only uses the triadic closure process to describe the evolution of a dynamic network and is designed for homogeneous networks. DYLINK2VEC [23] focuses on learning link embeddings in a dynamic network, that is, learning node pair representations instead of node representations. Then it adopts temporal functions to learn dynamic patterns over time. Furthermore, change2vec [4] utilizes a changed training dataset to only train the changed part of a dynamic network. It uses metapath2vec to model the snapshot and is unable to predict the future network. Dyngraph2vec [13] utilizes an LSTM-based deep autoencoder to process the history snapshots; however, it uses an adjacency matrix to represent the network structure, which is not very expressive, thus missing relation information between nodes. It is also not scalable to large networks as the dimension of the adjacency matrix could be large, causing high computational costs. In addition, it can only process at most 10 time stamp snapshots as reported in its original paper, since it has to process the complete history information at a time. However, the model we propose, M-DHIN, is able to process history information step by step, and thus the sequence length of history embeddings is not limited.

To the best of our knowledge, our model, M-DHIN, is the first to not only represent every snapshot of the dynamic HIN by modeling the triadic evolution process but also predict the future structure of the HIN via an LSTM-based deep autoencoder.

3 PRELIMINARIES

In the following, we introduce the notation and definitions of a dynamic HIN and metagraph. Then we formulate the problem of dynamic network representation learning for HINs. Table 1 lists the main terms and notation used.

Definition 3.1 (Dynamic (HIN)). Let $G = (V, E, T)$ be a directed graph, in which V denotes the set of nodes and E denotes the set of edges between nodes. Each node and edge are associated with a type mapping function, $\phi : V \rightarrow T_V$ and $\varphi : E \rightarrow T_E$, respectively. T_V and T_E denote the sets of node and edge types. A *heterogeneous information network* (HIN) is a network where $|T_V| > 1$ or $|T_E| > 1$; otherwise, it is a homogeneous network. Moreover, a *dynamic* HIN is a series of network snapshots denoted as $\{G^1, \ldots, G^{Time}\}$. For two consecutive time stamps t and $t+1$, the following conditions should be satisfied: $|V^{t+1}| \neq |V^t|$ or $|E^{t+1}| \neq |E^t|$, where $|V^t|$ and $|E^t|$ denote the number of nodes and edges at time stamp t, respectively. We assume that T_V and T_E

Table 1. Terms and Notation

Symbol	Definition
n	Number of nodes
(u, s, v)	HIN triplet containing head node u, tail node v, and metagraph s
$\mathbf{u}, \mathbf{s}, \mathbf{v}$	Embeddings of u, s, and v, respectively
$\mathbf{X}_{uv}$	Score matrix
Y^t	Node embeddings at time stamp t
Z^t	Metagraph embeddings at time stamp t
$a_u = [a_u^1, \ldots, a_u^t]$	Time sequential adjacency matrix of node u based on metagraphs
$y^{(k)}$	Hidden layers of LSTM-based autoencoder
$\theta = \{W^{(k)}, b^{(k)}\}$	Autoencoder weights and biases

remain unchanged during the entire evolution of the network. Figure 2 illustrates an example of a dynamic HIN.

Definition 3.2 (Metagraph). A *metagraph* is a subgraph of compatible node types and edge types, denoted as $S = (T_{VS}, T_{ES})$, where T_{VS} and T_{ES} denote the sets of node types and edge types in a metagraph, respectively.

As shown in Figure 1, metagraphs can be classified as symmetric and asymmetric ones; we will deal with both conditions in the proposed model.

PROBLEM 3.1 (DYNAMIC HIN REPRESENTATION LEARNING). *Given a series of dynamic networks* $\{G^1, \ldots, G^{Time}\}$, dynamic HIN representation learning *is to learn the dynamic low-dimensional vectors of nodes* $Y^t \in \mathbb{C}^{|V^t| \times d}$*, and* d *is the dimension of the node representation. In particular, our proposed method should also obtain the representations of metagraphs* $Z^t \in \mathbb{C}^{|S^t| \times d}$*. These representations are able to capture the evolving structural properties in a dynamic network.*

4 PROPOSED MODEL

In this section, we detail our proposed model M-DHIN in detail. An overview of M-DHIN is provided in Figure 3. We first introduce a complex embedding mechanism to represent a given dynamic HIN at the initial time stamp. Then we apply a triadic metagraph dynamic embedding mechanism to learn the dynamic HIN from time step 2 to time stamp t. Finally, we propose an LSTM-based deep autoencoder to perform graph prediction at time stamp $t + 1$.

4.1 Initial Complex Embedding Mechanism

Traditional HIN representation learning methods like DeepWalk, node2vec, and Metapath2vec need to process the complete HIN at every time stamp for dynamic generation of up-to-date node vectors, which is time-consuming, inefficient, and not scalable to large dynamic HINs. To address this issue, we propose a novel model named M-DHIN that is able to capture the main changes of dynamic networks. The initial step of M-DHIN is similar to static HIN embedding methods, and it represents the whole network (at the first time stamp) via a complex embedding scheme based on metagraphs.

Given the initial HIN at time stamp 1, $G^1 = (V^1, E^1)$, to represent the relationship between nodes and metagraphs, we introduce the concept of HIN triplets. A *HIN triplet* is denoted as (u, s, v), in which u is the first node generated in a metagraph, v the last, and s is the metagraph connecting

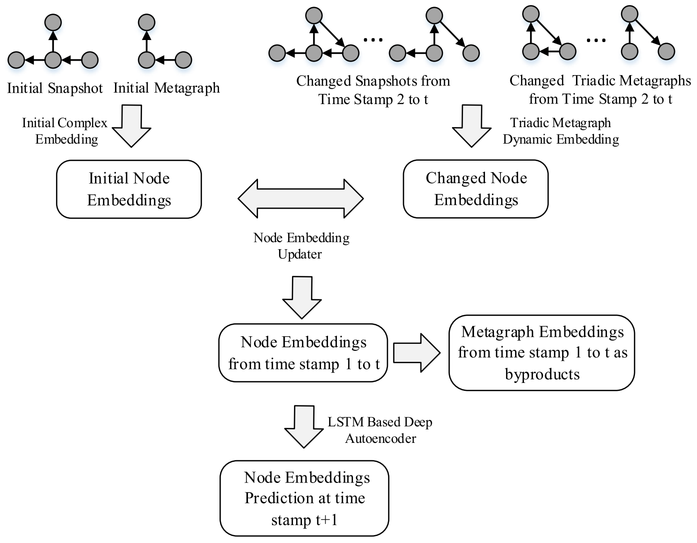

Initial Snapshot
Initial Metagraph
Changed Snapshots from Time Stamp 2 to t
Changed Triadic Metagraphs from Time Stamp 2 to t
Initial Complex Embedding
Triadic Metagraph Dynamic Embedding
Initial Node Embeddings
Changed Node Embeddings
Node Embedding Updater
Node Embeddings from time stamp 1 to t
Metagraph Embeddings from time stamp 1 to t as byproducts
LSTM Based Deep Autoencoder
Node Embeddings Prediction at time stamp t+1

Fig. 3. An overview of M-DHIN.

them. $\mathbf{u} \in \mathbb{C}^d$, $\mathbf{v} \in \mathbb{C}^d$, and $\mathbf{s} \in \mathbb{C}^d$ are the representation vectors of u, v, and s, respectively, where d is the dimension of the representation vectors.

The probability of whether a HIN triplet holds is denoted as

$$P(s|u,v) = \sigma(\mathbf{X}_{uv}), \tag{1}$$

where $X \in \mathbb{R}^{n\times n}$ is a score matrix, n is the number of training nodes, and σ is an activation function; we choose the sigmoid function.

Notice that a metagraph may be symmetric or asymmetric; that is, if a metagraph is symmetric, exchanging the first and last node will not change the semantics of the original metagraph, as shown in Figure 1(a). Correspondingly, exchanging the head and tail node will change the semantics of an asymmetric metagraph, as shown in Figure 1(b).

To address this issue, we adapt the schema of complex knowledge embeddings in [30] to fit the task of network embeddings. For a HIN triplet (u, s, v), its complex embedding is denoted as $\mathbf{u} = \mathrm{Re}(\mathbf{u}) + i\mathrm{Im}(\mathbf{u})$, $\mathbf{v} = \mathrm{Re}(\mathbf{v}) + i\mathrm{Im}(\mathbf{v})$, and $\mathbf{s} = \mathrm{Re}(\mathbf{s}) + i\mathrm{Im}(\mathbf{s})$, where $\mathrm{Re}(\mathbf{x}) \in \mathbb{R}^d$ and $\mathrm{Im}(\mathbf{x}) \in \mathbb{R}^d$ represent the real and imaginary part of the vector $\mathbf{x} \in \mathbb{C}^d$, respectively. Afterward, we introduce the Hadamard function to capture the relationship of u, v, and s in complex spaces, which is denoted as

$$\mathbf{X}_{uv} = \sum \mathbf{u} \odot \mathbf{s} \odot \bar{\mathbf{v}}, \tag{2}$$

where $\bar{\mathbf{v}}$ is the complex conjugate form of $\mathbf{v}$, and $\odot$ is the element-wise product. However, the sigmoid function in Equation (1) cannot be applied in complex spaces, and thus we only keep the real part of the objective function, which is still able to handle the symmetric and asymmetric structures well, and we will illustrate the reason in detail later. So, one element in the score matrix will eventually be

$$\mathbf{X}_{uv} = \mathrm{Re}\left(\sum \mathbf{u} \odot \mathbf{s} \odot \bar{\mathbf{v}}\right). \tag{3}$$

ALGORITHM 1: The Initial Complex Embedding Algorithm

```
Input  :
        (1) The initial HIN at the first time stamp $G^1 = (V^1, E^1)$;
        (2) Maximum number of iterations: MaxIter;
Output :
        Node embeddings $Y^1$ and metagraph embeddings $Z^1$;
1  $S^1 \leftarrow$ generate the initial metagraph set using GRAMI;
2  $(u, s, v) \leftarrow$ generate the HIN triplets based on metagraphs for training;
3  Iterations $\leftarrow 0$;
4  repeat
5      $f_\mathbf{s}(\mathbf{u}, \mathbf{v}) \leftarrow$ calculate score function to evaluate if a triplet holds using Equation (4);
6      $P(s|u, v) \leftarrow$ compute probability of a triplet being positive based on score function using
       Equation (11);
7      $\log O_{(u,s,v)} \leftarrow$ generate the objective function based on $P(s|u, v)$ using Equation (10);
8      $\mathbf{u}, \mathbf{s}, \mathbf{v} \leftarrow$ update embeddings of nodes and metagraphs using Equation (14);
9      Iterations $\leftarrow$ Iterations + 1
10 until convergence or Iterations $\geq$ MaxIterations;
```

Now that we have obtained the score matrix, the corresponding score function is defined as

$$\begin{aligned} f_\mathbf{s}(\mathbf{u}, \mathbf{v}) &= \mathrm{Re}(\langle \mathbf{u}, \mathbf{s}, \bar{\mathbf{v}} \rangle) \\ &= \langle \mathrm{Re}(\mathbf{u}), \mathrm{Re}(\mathbf{s}), \mathrm{Re}(\mathbf{v}) \rangle + \langle \mathrm{Im}(\mathbf{u}), \mathrm{Re}(\mathbf{s}), \mathrm{Im}(\mathbf{v}) \rangle \\ &\quad + \langle \mathrm{Re}(\mathbf{u}), \mathrm{Im}(\mathbf{s}), \mathrm{Im}(\mathbf{v}) \rangle - \langle \mathrm{Im}(\mathbf{u}), \mathrm{Im}(\mathbf{s}), \mathrm{Re}(\mathbf{v}) \rangle, \end{aligned} \tag{4}$$

where $\langle \cdot \rangle$ is the standard element-wise multi-linear dot product. For example, $\langle a, b, c \rangle = \sum_k a_k b_k c_k$, where a, b, c are vectors and k represents their dimension.

Equation (4) is able to handle asymmetric metagraphs thanks to the complex conjugate of one of the embeddings. In addition, if $\mathbf{s}$ is purely imaginary, i.e., its real part is zero, the score function is antisymmetric, and if real, the function is symmetric.[1] Co-authorship is a symmetric relation. Citation is an antisymmetric relation: (usually) paper A can only cite paper B but B cannot cite A; in other words, B is cited by A. By separating the imaginary and real part of the metagraph embedding s, we obtain a decomposition of the metagraph matrix $\mathbf{X_s}$ as the sum of an antisymmetric matrix $\mathrm{Im}(\mathrm{Udiag}(-\mathrm{Im}(\mathbf{s}))\bar{\mathbf{V}})$ and a symmetric matrix $\mathrm{Re}(\mathrm{Udiag}(\mathrm{Re}(\mathbf{s}))\bar{\mathbf{V}})$. From this we can see that metagraph embeddings naturally act as weights on each latent dimension, i.e., $\mathrm{Im}(\mathbf{s})$ over the antisymmetric, imaginary part of $\langle \mathbf{u}, \mathbf{v} \rangle$ and $\mathrm{Re}(\mathbf{s})$ over the symmetric, real part of $\langle \mathbf{u}, \mathbf{v} \rangle$. We have that $\langle \mathbf{u}, \mathbf{v} \rangle = \overline{\langle \mathbf{v}, \mathbf{u} \rangle}$, meaning that $\mathrm{Im}(\langle \mathbf{u}, \mathbf{v} \rangle)$ is antisymmetric, while $\mathrm{Re}(\langle \mathbf{u}, \mathbf{v} \rangle)$ is symmetric. Therefore, the mechanism introduced above enables us to accurately and effectively represent both symmetric and asymmetric (including antisymmetric) metagraphs between pairs of nodes.

In this initial step, we use an existing state-of-the-art approach GRAMI [8] to find all subgraphs that appear frequently in a database satisfying a given frequency threshold. Then we adopt these mined subgraphs to form metagraphs.

Algorithm 1 summarizes the initial complex embedding algorithm.

[1]Mathematically, if $f(x, y) = f(y, x)$, the function is symmetric; otherwise it is asymmetric, and in particular, if $f(x, y) = -f(y, x)$, the function is antisymmetric.

4.2 Dynamic Embedding Mechanism via Triadic Evolution Processes

After processing the full HIN to obtain the initial embeddings, we leverage triadic node blocks to further capture the structural evolutions of the dynamic HIN. A *triad* is a set containing three nodes. If every node is connected to each other, it is called a *closed triad,* and if there are only two edges between these three nodes, the triad is called an *open triad.* As mentioned in Section 1, the evolution of an open triad structure into a closed structure, i.e., the triadic closure process, is the fundamental change in the evolution of a dynamic HIN [37]. So in this step, we correspondingly construct the changed training datasets to include nodes that undergo a triadic closure. Meanwhile, we cannot ignore that there do exist triadic opening processes in a dynamic HIN; that is, two nodes in a triad may lose their relationship as time passes. Overall, we determine four common scenarios to describe the changes of a dynamic HIN:

(1) *Added edges form a triadic closure process.* We identify all the metagraphs having three node vectors changing from only two edges in between to a circle connected to each other. Those metagraphs will be included in the changed training datasets at time stamp t named as S^t_{change}. For a metagraph s having three nodes v_1, v_2, and v_3, we write (v_1, v_2) for the edge between v_1 and v_2. Then, S^t_{change}, obtained after the triadic closure process, is defined as

$$\begin{aligned} S^t_{change} = \Big\{ & s : \{(v_1, v_2) \notin E^t, (v_1, v_3) \in E^t, (v_2, v_3) \in E^t\} \neq \emptyset; \\ & \{(v_1, v_2) \in E^{t+1}, (v_1, v_3) \in E^{t+1}, (v_2, v_3) \in E^{t+1}\} \neq \emptyset \Big\}. \end{aligned} \tag{5}$$

(2) *Deleted edges cause a triadic opening process.* We collect all metagraphs having a triad evolving from a circle into a path with two edges; these nodes at time stamp t will be included in S^t_{change}. Similarly to the triadic closure process, S^t_{change} after the triadic opening process is defined as

$$\begin{aligned} S^t_{change} = \Big\{ & s : (v_1, v_2) \in E^t, (v_1, v_2) \notin E^{t+1}, \\ & \{(v_1, v_2) \in E^t, (v_1, v_3) \in E^t, (v_2, v_3) \in E^t\} \neq \emptyset; \\ & \{(v_1, v_2) \in E^{t+1}, (v_1, v_3) \in E^{t+1}, (v_2, v_3) \in E^{t+1}\} = \emptyset \Big\}. \end{aligned} \tag{6}$$

(3) *An added node.* Given an existing node in a metagraph denoted as v_1 and a newly added node v_2, S^t_{change} will be extended as

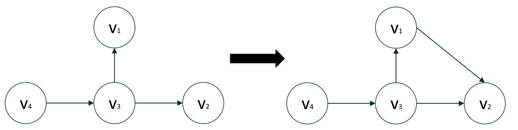
$$S^t_{change} = \{s : v_1, v_2 \in V^{t+1}, v_1 \in V^t, v_2 \notin V^t, (v_1, v_2) \notin E^t\}. \tag{7}$$

(4) *A deleted node.* Given two existing nodes in a metagraph denoted as v_1 and v_2, suppose v_2 is deleted, then S^t_{change} will become

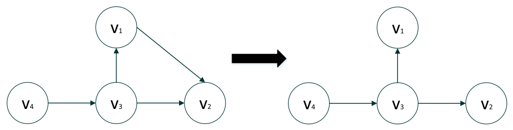
$$S^t_{change} = \{s : v_1, v_2 \in V^t, v_2 \notin V^{t+1}, (v_1, v_2) \in E^t\}. \tag{8}$$

Figure 4 illustrates the above procedure in detail.

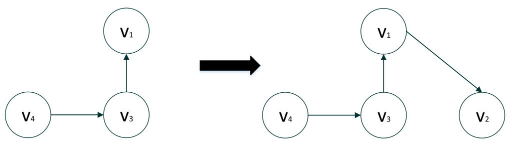
When forming the change set, the main difference between Change2vec and our model is that they only collect the nodes that have been changed and then form the meta-path in the change set, which may lose connection with the original network, and many meta-paths may be missed. For example, two newly added nodes may be connected by a meta-path through the original network but not connected in the change set. However, in our model, we build the change set based on the original metagraph; that is, when training the change set, the nodes are trained over the original metagraph after the changing process. By doing so, our model is better suited for training metapath and metagraph. In addition, this operation guarantees that the model learns the embeddings not meant for S^t_{change}, as it still has a connection with the original network. Notice that a node may be



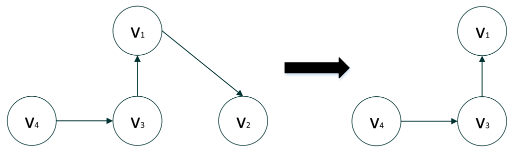
(a) Added edges form a triadic closure process. v_1 and v_2 will be included in S^t_{change}.



(b) Deleted edges cause a triadic opening process. v_1 and v_2 will be included in S^t_{change}.



(c) An added node. v_1 and v_2 will be included in S^t_{change}



(d) A deleted node. v_1 will be included in S^t_{change} and v_2 will be deleted from S^t_{change}.

Fig. 4. The formation of the change set.

involved in more than one scenario; a node can only be included once in S^t_{change}. This is to avoid duplicate calculations of a node's embedding. Once included, a node will be calculated based on the metagraph it belongs to, and the change process of the metagraph could describe all possible scenarios a node has experienced.

Once S^t_{change} has been obtained, we only train the set S^t_{change} using the metagraph-based complex mechanism to obtain the embeddings of the changed nodes instead of training the whole network all over again. Specifically, node embeddings Y^t evolve to Y^{t+1} at time stamp $t+1$ by removing the deleted node representations ($\{v : v \notin V^{t+1}, v \in V^t\}$), adding embeddings for newly added nodes ($\{v : v \notin V^t, v \in V^{t+1}\}$), and replacing the changed node representations in the triadic closure or opening processes($\{v \in V^t, v \notin V^{t+1}, Y^t_v \neq Y^{t+1}_v\}$).

Here we introduce the training objective. To obtain dynamic HIN embeddings from time stamp 1 to t with the graph changes observed, we first use a negative sampling strategy to form the training datasets. First, in a *positive triplet* (u, s, v) nodes u and v are connected via metagraph s, and in a *negative triplet* (u, s, v) nodes u and v are connected via metagraph s. For each positive HIN triplet (u, s, v), we generate the negative HIN triplets by randomly replacing u and v with other nodes, meanwhile restricting them to have the same type as the replaced one. *We also filter out replaced HIN triplets remaining positive after sampling.* Notice that the number of candidates of s is much smaller than that of u or v, so the sampled negative data are generated by only replacing u and v.

After sampling, we have training data in the form of (u, s, v, H_{uv}), where $H_{uv} \in \{1, 0\}$ is a binary value indicating whether the HIN triplet is positive or not. For a training instance (u, s, v, H_{uv}), if $H_{uv} = 1$, the objective function $O_{(u,s,v)}$ aims to maximize $P(s|u, v)$; otherwise, $P(s|u, v)$ should be minimized. Thus, we have our objective function as follows:

$$O_{(u,s,v)} = \begin{cases} P(s|u,v), & \text{if } H_{uv} = 1; \\ 1 - P(s|u,v), & \text{if } H_{uv} = 0. \end{cases} \tag{9}$$

To simplify our computations, we define $\log O_{(u,s,v)}$ as follows:

$$\log O_{(u,s,v)} = H_{uv} \log P(s|u,v) + (1 - H_{uv}) \log[1 - P(s|u,v)], \tag{10}$$

where $P(s|u, v)$ is defined as

$$P(s|u,v) = \text{sigmoid}(f_{\mathbf{s}}(\mathbf{u}, \mathbf{v})). \tag{11}$$

ALGORITHM 2: The Dynamic Embedding Algorithm

```
Input :
        (1) The dynamic HIN at time stamp $t$, $G^t = (V^t, E^t)$;
        (2) Maximum number of iterations: $MaxIter$;
Output:
        Embeddings of changed nodes $Y^t$ and embeddings of changed metagraphs $Z^t$;
1  $S^t_{change}$ ← generate the changed dataset using the four scenarios mentioned in Equations (5), (6),
   (7), and (8);
2  $(u, s, v)$ ← generate the HIN triplets based on the changed dataset $S^t_{change}$ for training;
3  $Iterations \leftarrow 0$;
4  repeat
5      $f_\mathbf{s}(\mathbf{u}, \mathbf{v})$ ← calculate score function to evaluate if a triplet holds using Equation (4);
6      $P(s|u, v)$ ← compute probability of a triplet being positive based on score function using
       Equation (11);
7      $\log O_{(u,s,v)}$ ← generate the objective function based on $P(s|u, v)$ using Equation (10);
8      $\mathbf{u}, \mathbf{s}, \mathbf{v}$ ← update embeddings of nodes and metagraphs using Equation (14);
9      $Iterations \leftarrow Iterations + 1$
10 until $convergence$ or $Iterations \geq MaxIterations$;
```

Specifically, we aim to maximize the objective function $\log O_{(u,s,v)}$. If the triplet (u, s, v) exists, $H_{uv} = 1$, then the objective function will be

$$\log O_{(u,s,v)} = \log P(s|u, v). \tag{12}$$

Maximizing the objective function will maximize the probability $P(s|u, v)$. In turn, we are able to obtain the embeddings of u, v, and s, which will maximize the probability that (u, s, v) holds. Similarly, for the negative sample that the triplet (u, s, v) does not exactly exist with $H_{uv} = 0$, then the objective function will be

$$\log O_{(u,s,v)} = \log[1 - P(s|u, v)]. \tag{13}$$

Maximizing the objective function will minimize the probability $P(s|u, v)$; correspondingly, the embeddings of u, v, and s that minimize the probability that (u, s, v) holds will be obtained.

We apply a SGD algorithm [24] to maximize the above objective function with **Adaptive Moment Estimation (Adam)** [17]. Specifically, for each training entry (u, s, v, H_{uv}), it goes backward to adjusts the embeddings $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{s}$ based on the the gradients of $\log O_{(u,s,v)}$ differentiated by $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{s}$, respectively.

$$\mathbf{u} := \mathbf{u} + \frac{d\log O_{(u,s,v)}}{d\mathbf{u}} \tag{14a}$$

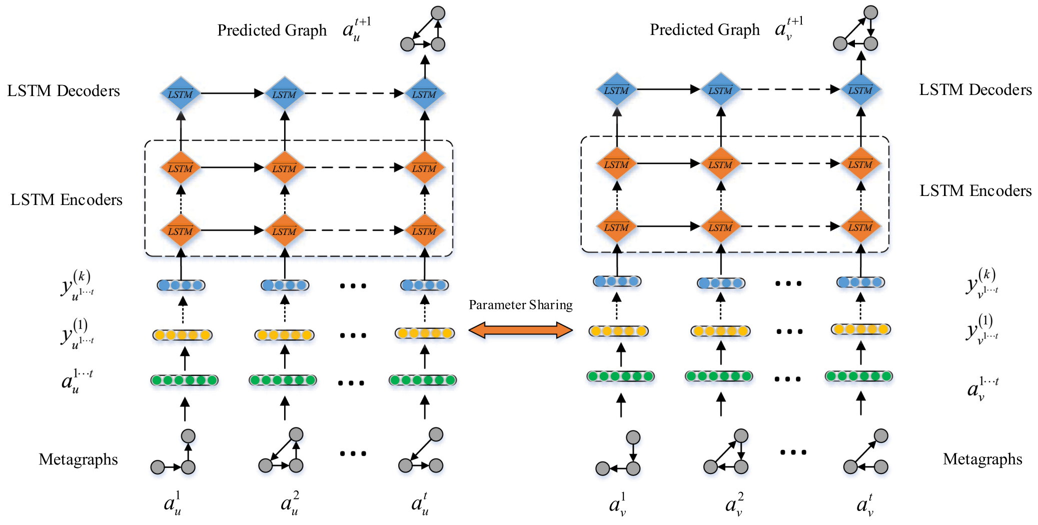
$$\mathbf{v} := \mathbf{v} + \frac{d\log O_{(u,s,v)}}{d\mathbf{v}} \tag{14b}$$

$$\mathbf{s} := \mathbf{s} + \frac{d\log O_{(u,s,v)}}{d\mathbf{s}}. \tag{14c}$$

Algorithm 2 summarizes the dynamic embedding algorithm.

4.3 Graph Prediction via LSTM-based Deep Autoencoder

After completing the two steps above, our model is able to generate the HIN representations at every time stamp; however, it is incapable to predict future structures in a dynamic HIN. In other



Predicted Graph a_u^{t+1}
Predicted Graph a_v^{t+1}
LSTM Decoders
LSTM Encoders
LSTM
$y_{u^{1\cdots t}}^{(k)}$
$y_{u^{1\cdots t}}^{(1)}$
$a_u^{1\cdots t}$
$y_{v^{1\cdots t}}^{(k)}$
$y_{v^{1\cdots t}}^{(1)}$
$a_v^{1\cdots t}$
Parameter Sharing
Metagraphs
a_u^1 a_u^2 a_u^t
a_v^1 a_v^2 a_v^t

Fig. 5. An LSTM-based deep autoencoder.

words, it can only produce the node embeddings based on the observed network evolutions, but it cannot describe unseen changes that may come in the future. To address this issue, we propose a deep autoencoder model based on an LSTM that can leverage the previous sequential structural evolutions to generate the future HIN representation. Figure 5 illustrates this LSTM-based deep autoencoder.

Note that when predicting the future network, we only train the changed metagraphs, not every metagraph. As discussed above, each node is included once by one original metagraph, which could also save training time. And for each changed metagraph included, we train it with the autoencoder. Correspondingly, each node in a change set will be calculated only once, no matter whether it is popular or not. In other words, we treat each node equally in the change set. To predict the future state of a node, it is more important to learn the dynamic process of a node instead of its popularity. So once the dynamic process of a given node is obtained, we are able to predict its future state regardless of its popularity.

Our deep autoencoder model consists of an encoder and decoder part. To construct the inputs of the encoder, for a node, we take its metagraph as neighbor nodes to form its adjacency matrix A. Then, for any pair of nodes u and v from the metagraph s, the input of the encoder consists of the time sequential adjacency vectors of u and v, denoted as $a_u = [a_u^1, \ldots, a_u^t]$ and $a_v = [a_v^1, \ldots, a_v^t]$, respectively. Specifically, a_u is a combination of two components. One is a row of adjacency matrix A that denotes the nodes that are adjacent with u, and it is further mapped into a d-dimensional vector by a fully connected layer. The other is the dynamic embeddings of node u learned in Algorithms 1 and 2. Then, the input is processed by the encoder to obtain the low-dimension representations y_u and y_v. The decoder aims to predict the neighborhood a_u^{t+1} and a_v^{t+1} via the embeddings at time stamp t, and the predicted adjacency vectors by the deep autoencoder are denoted as $\hat{a}_u^{t+1}$ and $\hat{a}_v^{t+1}$.

Concretely, for a node u with its neighborhood $a_u \in \mathbb{R}^{dt}$, in which d is the embedding dimension and t is the total time steps, the hidden representation of the first layer is represented as

$$y_u^{(1)} = f_a\left(W_A^{(1)} a_u + b_a^{(1)}\right), \tag{15}$$

in which $W_A^{(1)} \in \mathbb{R}^{d^{(1)} \times dt}$ is the parameter matrix in the first layer of the autoencoder, and $d^{(1)}$ is the representation dimension of the first layer. $b_a^{(1)} \in \mathbb{R}^{d^{(1)}}$ is the bias of the first layer of the encoder, and f_a denotes the activation function for which we choose sigmoid here. Then the output of the encoder, with k layers is computed as

$$y_u^{(k)} = f_a \left(W_A^{(k)} y_u^{(k-1)} + b_a^{(k)} \right). \tag{16}$$

To fully capture information about the past evolution of the metagraph, we further apply layers of LSTMs on the output of the encoder. For the first LSTM layer, its hidden state representation is computed as

$$\begin{aligned}
i_{u^t}^{(k+1)} &= \delta \left(W_i^{(k+1)} \left[a_u^t, y_{u^{t-1}}^{(k+1)} \right] + b_i^{(k+1)} \right) && \text{(17a)}\\
f_{u^t}^{(k+1)} &= \delta \left(W_f^{(k+1)} \left[a_u^t, y_{u^{t-1}}^{(k+1)} \right] + b_f^{(k+1)} \right) && \text{(17b)}\\
z_{u^t}^{(k+1)} &= \tanh \left(W_c^{(k+1)} \left[a_u^t, y_{u^{t-1}}^{(k+1)} \right] + b_c^{(k+1)} \right) && \text{(17c)}\\
c_{u^t}^{(k+1)} &= f_{u^t}^{(k+1)} \odot c_{u^{t-1}}^{(k+1)} + i_{u^t}^{(k+1)} \odot z_{u^t}^{(k+1)} && \text{(17d)}\\
o_{u^t}^{(k+1)} &= \delta \left(W_o^{(k+1)} \left[a_u^t, y_{u^{t-1}}^{(k+1)} \right] + b_o^{(k+1)} \right) && \text{(17e)}\\
y_{u^t}^{(k+1)} &= o_{u^t}^{(k+1)} \odot \tanh \left(c_{u^t}^{(k+1)} \right), && \text{(17f)}
\end{aligned}$$

in which i_{u^t} is the value to activate the input gate, f_{u^t} is the value to activate the forget gate, z_{u^t} is the new estimated candidate state, c_{u^t} is the cell state of the LSTM, o_{u^t} is the value to activate the output gate, δ represents the activation function for which we use sigmoid here, $W \in \mathbb{R}^{d^{(k+1)} \times (d + d^{(k+1)})}$ are parameter matrices, and $b \in \mathbb{R}^{d^{(k+1)}}$ denotes the biases. $d^{(k+1)}$ denotes the representation dimension of $k+1$ layer.

Given l layers of LSTMs, the output of the final LSTM can be represented as

$$\begin{aligned}
y_{u^t}^{(k+l)} &= o_{u^t}^{(k+l)} \odot \tanh \left(c_{u^t}^{(k+l)} \right) && \text{(18a)}\\
o_{u^t}^{(k+l)} &= \delta \left(W_{LSTM}^{(k+l)} \left[y_{u^t}^{(k+l-1)}, y_{u^{t-1}}^{(k+l)} \right] + b_o^{(k+l)} \right), && \text{(18b)}
\end{aligned}$$

in which $W_{LSTM}^{(k+l)} \in \mathbb{R}^{d^{(k+l)} \times (d^{(k+l-1)} + d^{(k+l)})}$. Finally, we aim to minimize the following loss function:

$$L_t = \left\| (\hat{a}^{t+1} - a^{t+1}) \odot B \right\|_F^2 = \left\| (f(a^1, \ldots, a^t) - a^{t+1}) \odot B \right\|_F^2. \tag{19}$$

We penalize the incorrect neighborhood reconstruction at time $t+1$ by utilizing the embeddings at time t. Therefore, our LSTM-based deep autoencoder model is able to predict node embeddings at a future time stamp. To simplify our notation, $f(\cdot)$ represents the function we adopt to generate the predicted neighborhood at time stamp $t+1$, and we use the above autoencoder framework as $f(\cdot)$. $B \in \mathbb{R}^d$ is the hyperparameter matrix to balance the weight of penalizing the observed neighborhood and $\odot$ represents the element-wise product.

To predict the HIN embeddings at future time stamp $t+1$, we optimize the objective function of the LSTM-based deep autoencoder framework. Specifically, we apply the gradient to the decoder weights on Equation (19), as follows:

$$\frac{\partial L_t}{\partial W_*^{(k+l)}} = \left[2(\hat{a}^{t+1} - a^{t+1}) \odot B \right] \left[\frac{\partial f_a (Y^{(k+l-1)} W_*^{(k+l)} + b^{(k+l)})}{\partial W_*^{(k+l)}} \right], \tag{20}$$

where $W_*^{(k+l)}$ is the parameter matrix of the $(k+l)$-th layer of the autoencoder. After calculating the derivatives, we also apply the SGD algorithm with Adam to train the model.

Algorithm 3 details the LSTM-based deep autoencoder for graph prediction. Algorithm 3 includes the evolvement of changing metagraphs in Algorithms 1 and 2 to form the adjacency matrix. It also makes use of the dynamic embeddings learned in Algorithms 1 and 2 to form a_v^t, which is the input of the autoencoder.

ALGORITHM 3: Graph Prediction via LSTM-based Deep Autoencoder Algorithm

```
Input :
    (1) The previous HIN from time stamp 1 to time stamp $t$, $\{G^1, G^2, \dots, G^t\}$;
    (2) The previous embeddings learned from time stamp 1 to time stamps $Y^1, Y^2, \dots, Y^t$;
    (3) Autoencoder parameters $\theta$;
    (4) Maximum number of iterations: $MaxIter$;
Output:
    Predicted embeddings of nodes $Y^{t+1}$ at time stamp $t+1$;
1  $a_u = [a_u^1, \dots, a_u^t]$ ← generate the sequential adjacency vectors of $u$ based on the metagraphs it belongs.;
2  $\theta$ ← RandomInit();
3  $Iterations \leftarrow 0$;
4  repeat
5      $Y_u^{(k)}$ ← Generate the output of encoder based on $a_u, Y^1, Y^2, \dots, Y^t$, and $\theta$ using Equations (15) and (16);
6      $y_{u^t}^{(k+l)}, o_{u^t}^{(k+l)}$ ← generate the final output of LSTM layers using Equation (18b);
7      $L_t$ ← compute the objective function using Equation (19);
8      $\theta$ ← update autoencoder parameters by gradient to back-propagate the whole network using Equation (20);
9      $Iterations \leftarrow Iterations + 1$;
10 until convergence or $Iterations \geq MaxIterations$;
11 Obtain node embedding $u^{t+1} \leftarrow y_{u^t}^{(k+l)}$
```

5 EXPERIMENTS

In this section, we first introduce the datasets employed to assess the effectiveness of proposed model M-DHIN. To compare with M-DHIN, we introduce other state-of-the-art models as baselines. Finally, we present the experimental setup for M-DHIN in detail.

5.1 Datasets

To assess the performance of M-DHIN, we conduct experiments using four dynamic datasets, extracted from DBLP,[2] YELP,[3] YAGO,[4] and Freebase.[5] Descriptive statistics for those datasets are presented in Table 2. For simplicity, we only provide the statistics at the initial time stamp and the last time stamp with different time spans (month and year).

- **DBLP** is a bibliographic dataset in computer science. We extract a subset from it using 15 sequential month time stamps from October 2015 to December 2016. Specifically, in October 2015, it contains 110,634 papers (P), 9,2473 authors (A), 4,274 topics (T), and 118 venues (V).

[2]http://dblp.uni-trier.de.
[3]https://www.yelp.com/dataset_challenge.
[4]https://old.datahub.io/dataset/yago.
[5]https://developers.google.com/freebase/.

Table 2. Dataset Statistics

Dataset	# Nodes	# Edges	# Node Types	# Labels	# Time Stamps
DBLP (initial)	207,499	902,362	4	4	15 (month)
DBLP (last)	256,082	1,121,273			
YELP (initial)	148,662	676,376	4	3	12 (month)
YELP (last)	183,529	802,023			
YAGO (initial)	19,024	34,023	5	4	10 (year)
YAGO (last)	24,439	39,028			
Freebase (initial)	6,641	7,213	4	3	12 (month)
Freebase (last)	8,018	8,921			

In December 2016, it contains 135,348 papers (P), 116,137 authors (A), 4,476 topics (T), and 121 venues (V). Authors were separated into four areas labeled: database, machine learning, data mining, and information retrieval.

- **YELP** is a social media dataset with reviews of restaurants. The extracted dynamic HIN has 12 monthly sequential snapshots from January 2016 to December 2016. It contains 81,240 reviews (V), 43,927 customers (C), 74 food-related keywords (K), and 23,421 restaurants (R) at the outset, in January 2016. It contains 102,367 reviews (V), 51,299 customers (C), 86 food-related keywords (K), and 29,777 restaurants (R) in December 2016. The restaurants were split into three types—American, sushi bar, and fast food.
- **YAGO** captures world knowledge, and we extract a subset with 10 yearly snapshots about movies ranging from 2007 to 2016. In 2009, it has 5,334 movies (M), 8,346 actors (A), 1,345 directors (D), 1,123 composers (C), and 2,876 producers (P). In 2018, it has 7,476 movies (M), 10,212 actors (A), 1,872 directors (D), 1,342 composers (C), and 3,537 producers (P). The movies were divided into five genres—horror, action, adventure, crime, and sci-fi.
- **Freebase** also contains world knowledge and facts, and the extracted subset is related to video games. It consists of 12 monthly snapshots from January 2016 to December 2016. It contains 3,435 games (G), 1,284 publishers (P), 1,768 developers (D), and 154 designers (S) at the outset, in January 2016. It contains 4,122 games (G), 1,673 publishers (P), 2,022 developers (D), and 201 designers (S) in December 2016. The games belong to one of three types—action, adventure, and strategy.

5.2 Tasks

For our experimental evaluation we consider a diverse set of tasks: (1) link prediction, (2) changed link prediction, (3) node classification, (4) node prediction, (5) graph reconstruction, and (6) anomaly detection. By evaluating the performance of different models on multiple benchmark tasks, we will be able to assess to which degree a model is able to describe and capture the features of a dynamic HIN. Link prediction, changed link prediction, and node prediction test a model's ability to predict the future evolution of a HIN. Node classification and graph reconstruction test whether a model is able to generate proper dynamic HIN embeddings. Anomaly detection tests a model's capacity to detect unexpected events during the evolution of a dynamic HIN.

5.3 Baselines

We include two types of baselines: one consists of static embedding methods, and the other consists of dynamic embedding methods. For the static embedding methods, we consider both homogeneous and heterogeneous methods. DeepWalk [22] and node2vec [15] were originally designed to represent the homogeneous network. metapath2vec [7] and MetaGraph2Vec [11] were devised for heterogeneous networks using metapaths and metagraphs, respectively. Notice that we did not apply methods that leverage text information, as our datasets do not contain such information, but only nodes and edges.

- DeepWalk utilizes random walks to capture the structural information of a HIN and applies homogeneous SkipGram to learn the representation. It has two main hyper-parameters, *Walk Length* (wl) for random walks, and *Window Size* (ws) for the SkipGram mechanism. To report the best performance, we use grid search to find the best configuration on different tasks using $wl \in \{20, 40, 60, 80, 100\}$ and $ws \in \{3, 5, 7\}$.
- node2vec is an extension of DeepWalk as it uses biased random walks to better explore the structure and it also uses SkipGrams to learn the network embedding. We adopt the same wl and ws as DeepWalk. For its bias parameters p and q, we use grid search on $p \in \{0.5, 1, 1.5, 2, 5\}$ and $q \in \{0.5, 1, 1.5, 2, 5\}$.
- metapath2vec adopts metapaths to capture the structural information of a HIN and uses heterogeneous SkipGrams that confine the context window to one specific type to learn the embedding. We utilize the same wl and ws as DeepWalk.
- MetaGraph2Vec constructs metagraphs by simply combining several metapaths, which is a path-oriented model in essence. Then it adopts heterogeneous SkipGrams to learn the final representation. For wl and ws, we set the same values as DeepWalk.

For a fair comparison, we also evaluate the performance of four dynamic embedding models, i.e., DynamicTriad, DynGEM, dyngraph2vec, and change2vec.

- DynamicTriad [37] describes the evolution of a network based merely on the triadic closure process and it is designed for homogeneous networks. β_0 and β_1 are two hyper-parameters denoting the weight of the triad closure process and the weight of the temporal smoothness, respectively. We leverage grid search to find the best configuration from $\beta_0 \in \{0.01, 0.1, 1, 10\}$ and $\beta_1 \in \{0.01, 0.1, 1, 10\}$.
- Change2vec [4] first learns the initial embeddings of the dynamic HIN and then samples the changed node sets to be trained using the metapath2vec model. We set its configuration to be the same as for metapath2vec.
- DynGEM [14] uses a deep autoencoder to capture the dynamics at time stamp t of a HIN only using the snapshot at time stamp $t-1$. $\alpha, \upsilon_1, \upsilon_2$ are relative weight hyperparameters chosen via grid search from $\alpha \in \{10^{-6}, 10^{-5}\}$, $\upsilon_1 \in \{10^{-4}, 10^{-6}\}$, $\upsilon_2 \in \{10^{-3}, 10^{-6}\}$
- dyngraph2vec [13] uses a deep LSTM-based autoencoder to process the previous snapshots based on the lookback window of length lb, i.e., a training snapshot of length lb. Whereas M-DHIN trains all the previous time stamp snapshots via metagraph embeddings and uses an autoencoder to predict only the final snapshot, dygraph2vec learns all snapshot graph embeddings using an autoencoder. So lb is limited due to restricted hardware resources, no greater than 10 as reported in [13]. Therefore, lb is chosen from $\{3, 4, 5, 6, 7, 8, 9, 10\}$.

As to other parameters like learning rate, embedding dimension, and so forth, we directly adopt the best setup reported in papers that originally introduced the methods listed.

We also add a variant of M-DHIN named M-DHIN-MG, which only uses the dynamic complex embeddings via metagraphs without a deep LSTM-based autoencoder mechanism, so as to measure the effectiveness of the autoencoder as part of our ablation analysis.

5.4 Experimental Setup

To assess the performance of M-DHIN, we leverage grid search to find the best experimental configuration. Concretely, the node and metagraph embedding dimensions are chosen from

$\{32, 64, 128, 256\}$, the learning rate in SGD from $\{0.01, 0, 02, 0.025, 0.05, 0.1\}$, the ratio of negative sampling from $\{3, 4, 5, 6, 7\}$, the number of autoencoder layers from $\{2, 3, 4\}$, the number of LSTM layers from $\{2, 3, 4\}$, and the training epochs from $\{5, 10, 15, 20, 25, 30, 35, 40\}$. Balancing effectiveness and efficiency, we choose the following configurations to produce the experimental results reported in the following sections. The embedding dimension is set to 128, the learning rate to 0.025, the negative sampling ratio to 5 (i.e., five negative samples for each positive sample), the number of autoencoder and LSTM layers are all set to 2, and the number of training epochs is set to 20.

All experiments are performed on a 64-bit Ubuntu 16.04.1 LTS system with Intel (R) Core (TM) i9-7900X CPU, 64 GB RAM, and a GTX-1080 GPU with 8 GB memory.

6 EXPERIMENTAL RESULTS AND ANALYSIS

One by one we report on the experimental results for the six tasks that we consider in this article. After that we analyze the parameter sensitivity to measure the stability of M-DHIN. We report on statistical significance with a paired two-tailed t-test; we mark a significant improvement of M-DHIN over the second best model in each task for $p < 0.05$ with $^\blacktriangle$.

6.1 Link Prediction

In this section, we present the outcomes of a link prediction task. For static HINs, evaluation on the link prediction task is usually conducted by first removing a fraction of the edges and then predicting the missing edges. Link prediction for dynamic HINs consists of predicting the existence of an edge at time stamp $t + 1$ based on the all previous time stamps.

We use **Mean Average Precision (MAP)** and **top-*m* recall (R@*m*)** as the evaluation metrics and set m to 100. MAP is the mean of the average precision scores for the ranking result of each node. R@m is the percentage of the ground truth ranked in the top-m returned results. Higher MAP and higher R@m represent better performance.

Table 3 provides the experimental results. It is obvious that the dataset scale has an effect on performance, with larger datasets having worse results. M-DHIN consistently and significantly outperforms all other models on every dataset. Specifically, DeepWalk and node2vec achieve the worst performance, which we attribute to the fact that they are limited to representing homogeneous static networks. MetaGraph2Vec outperforms metapath2vec on each dataset, which indicates that metagraphs are more expressive than metapaths. Notice that all the dynamic embedding methods outperform the static embedding methods as they focus on the changing parts in a dynamic HIN. DynamicTriad slightly outperforms Metagraph2vec and we attribute this to the fact that DynamicTriad was originally designed for dynamic homogeneous networks and only focuses on triadic closure processes, so that it is unable to fully capture the structure evolutions. Change2vec performs worse than DynGEM, which illustrates that a deep autoencoder is more effective than a SkipGram-based metapath embedding in describing evolving HIN. dyngraph2vec outperforms DynGEM and M-DHIN-MG due to the fact that DynGEM and M-DHIN-MG only harness the graph snapshot at time stamp t, while dyngraph2vec employs the history information with l lookback, i.e., graph snapshots from $t - l + 1$ to t via an LSTM mechanism.

Comparing M-DHIN with M-DHIN-MG, the result that M-DHIN has a better performance verifies the effectiveness of LSTM-based deep autoencoders on predicting the graph at time stamp $t+1$ by using the history information. M-DHIN even performs better than dyngraph2vec, which indicates that forming adjacency matrices according to a metagraph can offer more accurate structural information than simply using its neighbor nodes in a certain scale.

In conclusion, M-DHIN achieves the best result by using an LSTM-based deep autoencoder to realize graph prediction at time stamp $t + 1$, using the history metagraphs from time stamp 1 to t.

Table 3. Experimental Results on the Link Prediction Task

	DBLP		YELP		YAGO		Freebase	
Model	MAP	R@100	MAP	R@100	MAP	R@100	MAP	R@100
DeepWalk	0.102	0.162	0.111	0.174	0.131	0.247	0.211	0.394
node2vec	0.104	0.165	0.118	0.176	0.136	0.253	0.216	0.392
metapath2vec	0.115	0.171	0.124	0.181	0.156	0.266	0.232	0.411
MetaGraph2Vec	0.121	0.178	0.127	0.192	0.161	0.274	0.239	0.418
DynamicTriad	0.123	0.179	0.135	0.194	0.167	0.281	0.242	0.422
Change2vec	0.134	0.183	0.142	0.208	0.172	0.290	0.257	0.447
DynGEM	0.139	0.186	0.147	0.213	0.179	0.294	0.267	0.453
dyngraph2vec	0.143	0.193	0.154	0.221	0.187	0.302	0.275	0.462
M-DHIN-MG	0.140	0.192	0.151	0.219	0.183	0.298	0.271	0.459
M-DHIN	**0.150**▲	**0.197**▲	**0.159**▲	**0.225**▲	**0.194**▲	**0.309**▲	**0.281**▲	**0.467**▲

6.2 Changed Link Prediction

In real-world scenarios, only a small part of the HIN links will change over time (by being removed or added) [37], so in this section we only focus on these changed links and study the process of their evolutions. In other words, we only consider the changed subset and ignore other nodes in the training and testing processes in this task. We use MAP and R@100 as evaluation metrics.

Table 4 shows the performance on the changed link prediction task. We observe that the margins between static network embedding and dynamic network embedding become much larger, and this is because static network embedding methods consider the whole network and are not sensitive to the evolution of a dynamic HIN. The performances of DynamicTriad, DynGEM, and dyngraph2vec become worse to a certain extent, as they describe the network changing process in a global way. In other words, at each time stamp, they process the entire network instead of focusing on changing parts so as to generate dynamic embeddings. In contrast, Change2vec, M-DHIN-MG, and M-DHIN all experience an increase in both MAP and R@100 metrics, and we attribute this to the formation of S^t_{change}, which includes changing nodes only. By only training the S^t_{change} after the initial time stamp, these three models are able to capture a HIN's evolving structure more precisely.

6.3 Node Classification

In this section we report on the experimental results for the node classification task. The node labels to be classified for each dataset are introduced in Section 5.1. We calculate the micro-f1 (MIC-F1) and macro-f1 (MAC-F1) as evaluation metrics. Higher micro-f1 and macro-f1 scores represent better performance.

Table 5 provides the experimental results. Overall, M-DHIN outperforms all other models on every dataset regardless of their scale, which confirms its effectiveness. Specifically, metapath2vec and MetaGraph2Vec outperform the homogeneous static network models DeepWalk and node2vec despite the fact that they perform slightly worse on Freebase, which indicates that metapaths are a better way to explore the features of a network than random walks. As mentioned in Section 5.3, MetaGraph2Vec is basically a path-oriented model, so it is only slightly better than metapath2vec with more metapaths combined. We also observe that, unlike the results for the link prediction task, M-DHIN-MG outperforms dyngraph2vec except on MAC-F1 in YELP, and we hypothesize that this is because embedding information plays a more important role in node classification than history information. In other words, our complex embeddings based on GRAMI-generated metagraphs are

Table 4. Experimental Results on the Changed Link Prediction Task

	DBLP		YELP		YAGO		Freebase	
Model	MAP	R@100	MAP	R@100	MAP	R@100	MAP	R@100
DeepWalk	0.092	0.151	0.102	0.163	0.117	0.221	0.201	0.364
node2vec	0.096	0.152	0.106	0.168	0.121	0.229	0.204	0.361
metapath2vec	0.108	0.159	0.112	0.174	0.147	0.233	0.223	0.402
MetaGraph2Vec	0.112	0.164	0.115	0.187	0.155	0.242	0.227	0.406
DynamicTriad	0.121	0.177	0.133	0.189	0.166	0.276	0.238	0.419
Change2vec	0.139	0.187	0.148	0.211	0.176	0.295	0.263	0.452
DynGEM	0.135	0.185	0.145	0.210	0.176	0.292	0.265	0.451
dyngraph2vec	0.141	0.189	0.150	0.217	0.178	0.295	0.267	0.455
M-DHIN-MG	0.145	0.194	0.154	0.222	0.187	0.302	0.272	0.462
M-DHIN	**0.157**▲	**0.206**▲	**0.165**▲	**0.230**▲	**0.203**▲	**0.315**▲	**0.285**▲	**0.473**▲

Table 5. Experimental Results on the Node Classification Task

	DBLP		YELP		YAGO		Freebase	
Model	MIC-F1	MAC-F1	MIC-F1	MAC-F1	MIC-F1	MAC-F1	MIC-F1	MAC-F1
DeepWalk	0.164	0.161	0.123	0.109	0.289	0.264	0.518	0.431
node2vec	0.168	0.166	0.131	0.115	0.293	0.271	0.521	0.433
metapath2vec	0.174	0.172	0.237	0.244	0.301	0.284	0.503	0.411
MetaGraph2Vec	0.177	0.176	0.242	0.245	0.305	0.287	0.509	0.417
DynamicTriad	0.193	0.188	0.261	0.268	0.337	0.308	0.526	0.454
Change2vec	0.197	0.193	0.267	0.271	0.341	0.324	0.533	0.466
DynGEM	0.211	0.207	0.273	0.279	0.347	0.326	0.537	0.471
dyngraph2vec	0.219	0.217	0.278	0.288	0.358	0.333	0.545	0.477
M-DHIN-MG	0.223	0.219	0.282	0.286	0.361	0.338	0.549	0.480
M-DHIN	**0.231**▲	**0.227**▲	**0.285**▲	**0.294**▲	**0.366**▲	**0.341**▲	**0.554**▲	**0.484**▲

more expressive than embeddings generated by an LSTM-based deep autoencoder, even though M-DHIN-MG only utilizes the graph at time stamp t for node classification. Combining both complex embeddings and history information, M-DHIN performs better than M-DHIN-MG.

6.4 Node Prediction

In the node prediction task, we predict the label of a node at time stamp $t + 1$ based on the node embeddings from time stamp 1 to t. We choose micro-f1 (MIC-F1) and macro-f1 (MAC-F1) as our evaluation metrics.

Table 6 provides the experimental results for the node prediction task. We observe that the performance in node prediction is similar to that on the node classification task with M-DHIN outperforming all baselines on each dataset. Notice that, similar to the link prediction task, dyngraph2vec outperforms M-DHIN-MG, which proves that history information attributes more to prediction-related tasks than embedding information.

Table 6. Experimental Results on the Node Prediction Task

	DBLP		YELP		YAGO		Freebase	
Model	MIC-F1	MAC-F1	MIC-F1	MAC-F1	MIC-F1	MAC-F1	MIC-F1	MAC-F1
DeepWalk	0.132	0.129	0.117	0.093	0.291	0.269	0.489	0.392
node2vec	0.137	0.135	0.123	0.109	0.295	0.275	0.495	0.398
metapath2vec	0.145	0.141	0.225	0.231	0.307	0.286	0.477	0.379
MetaGraph2Vec	0.149	0.142	0.237	0.233	0.308	0.291	0.482	0.385
DynamicTriad	0.166	0.162	0.249	0.254	0.335	0.302	0.502	0.414
Change2vec	0.169	0.165	0.253	0.261	0.338	0.322	0.507	0.418
DynGEM	0.189	0.181	0.261	0.271	0.353	0.331	0.511	0.431
dyngraph2vec	0.195	0.192	0.269	0.280	0.365	0.338	0.528	0.439
M-DHIN-MG	0.192	0.188	0.267	0.277	0.364	0.334	0.524	0.433
M-DHIN	**0.201**▲	**0.199**▲	**0.273**▲	**0.286**▲	**0.369**▲	**0.345**▲	**0.533**▲	**0.447**▲

6.5 Graph Reconstruction

This task is similar to the traditional link prediction task, that is, to construct graph edges between pairs of nodes based on node embeddings. Given two nodes u and v, we aim to determine whether there exists an edge between them via the absolute difference in positions between their corresponding embeddings, i.e., $|\mathbf{u}^t - \mathbf{v}^t|$. The ratio of existing links in the top m pairs of nodes is the reconstruction precision. Then we evaluate the experimental performance using the metrics MAP and R@m.

Table 7 presents the experimental results on the graph reconstruction task. M-DHIN has the best performance on all datasets, which further proves the advantage of combining metagraph-based complex embeddings with history information learned by an LSTM-based deep autoencoder. Notice that Change2vec performs relatively well on this task, outperforming dyngraph2vec on each dataset except for being slightly worse on R@100 in YELP. We attribute this to the fact that graph reconstruction relies on relations between nodes. Change2vec leverages metapaths to better express the relations, while dyngraph2vec's autoencoder model employs the adjacency matrix that focuses on a node's neighborhood instead of the edges in between. M-DHIN-MG outperforms Change2vec, and it verifies that our complex embeddings via GRAMI-generated metagraphs are more expressive in describing relations than traditional metapath-based embeddings.

6.6 Anomaly Detection

The anomaly detection task is to detect newly arriving nodes or edges that do not naturally belong to the existing clusters. We use a k-means clustering algorithm to generate clusters based on the dynamic HIN and then use anomaly injection [2] to create the anomalous nodes and edges. Specifically, we inject 1% and 5% anomalies to build the testing datasets. We adopt the **area under the curve (AUC)** score to evaluate the performance of all models. Higher AUC scores represent better performance.

Table 8 shows the experimental results of all models. M-DHIN outperforms other baselines, detecting 1% and 5% anomalies in every dataset, consistently and significantly. metapath2vec and MetaGraph2Vec outperform the homogeneous static network models DeepWalk and node2vec, which indicates that in identifying anomalies, a metapath is more powerful than a random walk since it defines certain relations, being more sensitive to outliers, while a random walk regards every node equally. As mentioned above, MetaGraph2Vec is actually a metapath-based model as it

Table 7. Experimental Results on the Graph Reconstruction Task

	DBLP		YELP		YAGO		Freebase	
Model	MAP	R@100	MAP	R@100	MAP	R@100	MAP	R@100
DeepWalk	0.111	0.171	0.119	0.188	0.147	0.255	0.225	0.412
node2vec	0.114	0.175	0.124	0.190	0.154	0.259	0.229	0.418
metapath2vec	0.122	0.182	0.129	0.197	0.169	0.278	0.241	0.423
MetaGraph2Vec	0.125	0.185	0.133	0.203	0.174	0.283	0.245	0.425
DynamicTriad	0.131	0.191	0.141	0.209	0.176	0.297	0.258	0.429
Change2vec	0.146	0.203	0.164	0.226	0.204	0.318	0.287	0.471
DynGEM	0.135	0.194	0.152	0.217	0.189	0.303	0.265	0.451
dyngraph2vec	0.139	0.199	0.159	0.228	0.197	0.310	0.275	0.464
M-DHIN-MG	0.149	0.207	0.167	0.233	0.208	0.320	0.292	0.477
M-DHIN	**0.154**▲	**0.211**▲	**0.172**▲	**0.239**▲	**0.218**▲	**0.327**▲	**0.301**▲	**0.483**▲

Table 8. Experimental Results on the Anomaly Detection Task

	DBLP		YELP		YAGO		Freebase	
Model	1%	5%	1%	5%	1%	5%	1%	5%
DeepWalk	0.743	0.724	0.712	0.689	0.694	0.676	0.738	0.719
node2vec	0.749	0.727	0.707	0.688	0.692	0.679	0.735	0.722
metapath2vec	0.752	0.735	0.718	0.699	0.701	0.685	0.744	0.727
MetaGraph2Vec	0.756	0.739	0.722	0.704	0.706	0.691	0.748	0.729
DynamicTriad	0.757	0.742	0.724	0.707	0.715	0.693	0.751	0.738
Change2vec	0.784	0.773	0.746	0.728	0.732	0.705	0.776	0.762
DynGEM	0.772	0.748	0.728	0.717	0.721	0.697	0.759	0.743
dyngraph2vec	0.776	0.752	0.733	0.723	0.725	0.701	0.761	0.749
M-DHIN-MG	0.789	0.778	0.752	0.734	0.737	0.706	0.784	0.768
M-DHIN	**0.795**▲	**0.783**▲	**0.755**▲	**0.736**▲	**0.745**▲	**0.712**▲	**0.792**▲	**0.773**▲

simply combines metapaths. Change2vec performs better than DynGEM and dyngraph2vec, which is due to the fact that a metapath is more expressive than a node's neighborhood adjacency matrix. An adjacency matrix is unable to describe the specific relationship between nodes; it only describes the existence of nodes and edges and regards them equally, and thus is not effective in anomaly detection. Note that M-DHIN-MG outperforms Change2vec, which verifies that our GRAMI-generated metagraphs are more sensitive than metapaths to anomalies, since metagraphs can express more sophisticated relation information than metapaths. M-DHIN performs even better than M-DHIN-MG, and we attribute this to that employing history information is helpful to identify whether a newly arriving node or edge is anomalous or not.

6.7 Computational Costs

In this section, we present the computation cost of our model in detail. Table 9 illustrates the results. The time reported is wall-clock time, averaged over 10 runs. Considering the scale of the dataset and the hardware we use, the cost of time is acceptable on each task (at most of around 1 hour). It further proves that M-DHIN is scalable to large dynamic HINs.

Table 9. Running Times on Each Dataset of M-DHIN

	LP	CLP	NC	NP	GR	AD
Datasets	Time (s)	Time (s)	Time (s)	Time (s)	Time (s)	Time (s)
DBLP (Train)	3,484	2,282	2,468	3,023	2,643	4,273
DBLP (Test)	1,638	1,033	1,172	1,436	1,233	2,023
YELP (Train)	2,422	2,133	1,836	2,237	1,927	2,863
YELP (Test)	1,162	1,023	832	1,023	923	1,245
YAGO (Train)	458	327	389	412	399	523
YAGO (Test)	221	153	184	203	191	247
Freebase (Train)	183	132	146	173	166	201
Freebase (Test)	88	58	71	82	75	96

Abbreviations used: LP: link prediction, CLP: changed link prediction, NC: node classification, NP: node prediction, GR: graph reconstruction, AD: anomaly detection.

Table 10. Ablation Analysis of the Initial Embedding on the DBLP Dataset

	LP		CLP		NC		NP		GR		AD	
Model	MAP	R@100	MAP	R@100	MIC-F1	MAC-F1	MIC-F1	MAC-F1	MAP	R@100	1%	5%
M-DHIN	**0.150**	**0.197**	**0.157**	**0.206**	**0.231**	**0.227**	**0.201**	**0.199**	**0.154**	**0.211**	**0.795**	**0.783**
M-DHIN\initial	0.124	0.182	0.137	0.178	0.213	0.212	0.183	0.178	0.135	0.196	0.782	0.768

Same abbreviations used as in Table 9.

6.8 Ablation Analysis

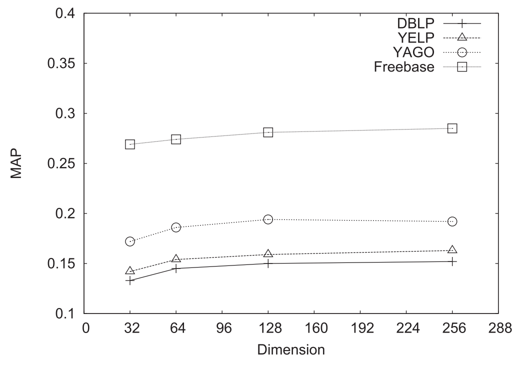
To evaluate the effect of the initial embedding, here we conduct the ablation analysis. The variant of M-DHIN that we consider is denoted as M-DHIN\initial, as it is trained without initial embedding. We only report the experimental outcomes on the DBLP dataset as the findings on the other datasets are qualitatively similar. Table 10 shows the experimental results of the ablation analysis over initial embedding. We observe that M-DHIN outperforms M-DHIN\initial consistently, which verifies the effectiveness of the initial embedding.

6.9 Parameter Sensitivity

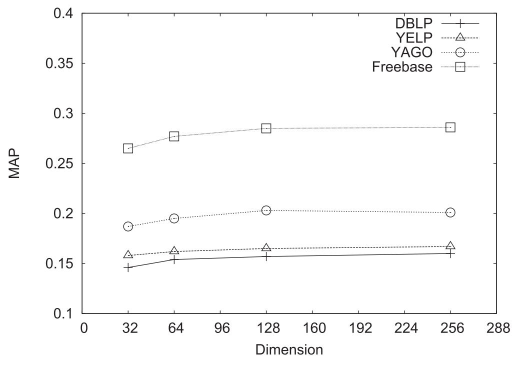
In this final subsection, we conduct a parameter sensitivity study on each task. Specifically, we choose node dimension and the ratio of negative sampling for analysis. For simplicity, we only choose one evaluation metric for each dataset in each task.

For our analysis of the sensitivity of M-DHIN to the choice of dimension, we choose MAP as our metric for the link prediction, changed link prediction, and graph reconstruction tasks; for node classification and node prediction, we choose MIC-F1 as the evaluation metric; for anomaly detection, we choose AUC on 1% anomalies for evaluation. The other parameters are set the same as those mentioned in Section 5.4. Figure 6 provides the experimental results. Basically, M-DHIN is not very sensitive to dimension, and in many cases, dimension 128 results in the best performance. Although in some cases dimension 256 outperforms dimension 128, as higher dimension provides a higher representation ability, the gap is not very large. To balance effectiveness and efficiency, we prefer a dimension of 128 for our experiments.

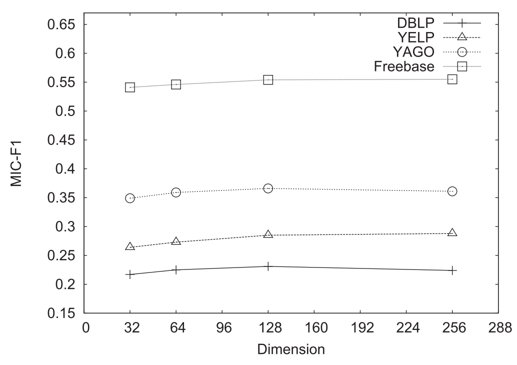
As to the analysis of M-DHIN's sensitivity to the negative ratio, for link prediction, changed link prediction, and graph reconstruction tasks, we adopt R@100 for analysis; for node classification and node prediction, we choose MAC-F1 for parameter analysis; for anomaly detection, we choose



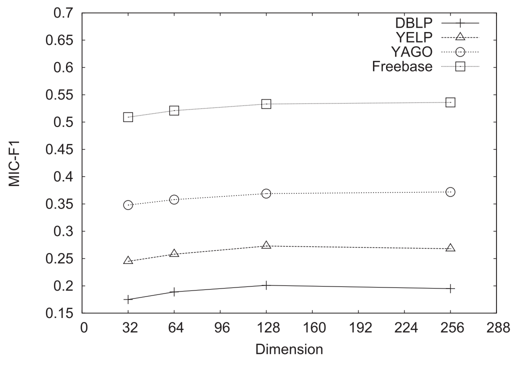
(a) Link prediction



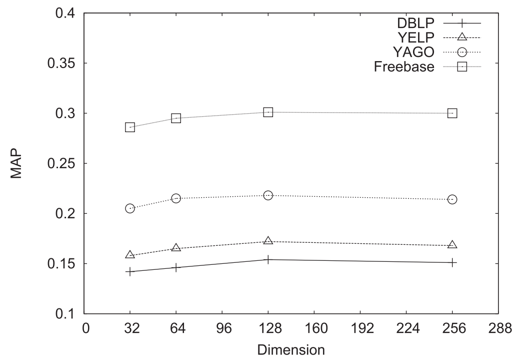
(b) Changed link prediction



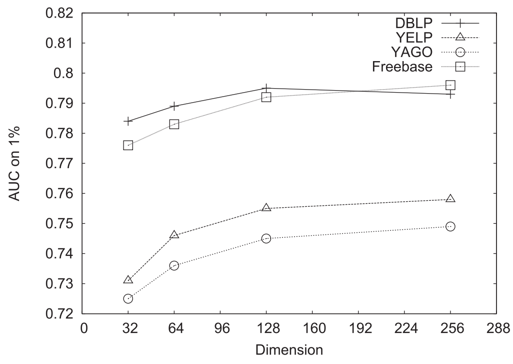
(c) Node classification



(d) Node prediction



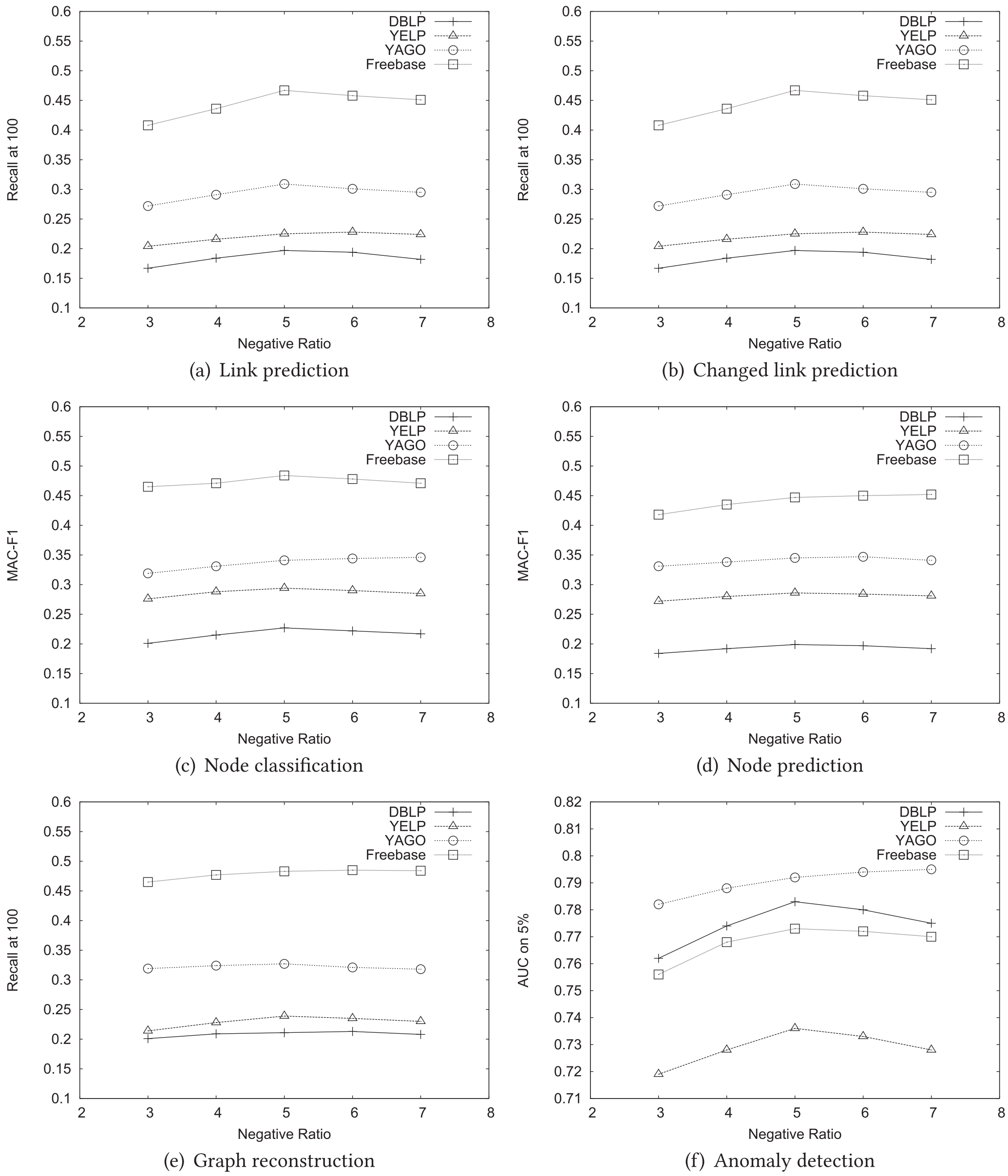
(e) Graph reconstruction



(f) Anomaly detection

Fig. 6. Dimension sensitivity analysis.

AUC on 5% anomalies as the analysis metric. We set the other parameters to be the same as those in Section 5.4. Figure 7 presents the results on different negative ratios. Overall, M-DHIN is not very sensitive to the choice of negative ratio. We observe that in most cases ratio 5 obtains the best results. This is due to that with a lower ratio, it tends to cause overfitting issues with too many ground-truth samples in the training datasets, while with a higher ratio, it tends to have an underfitting problem with too many negative samples in training datasets. Therefore, we choose a negative ratio of 5 for our experiments.



(a) Link prediction (b) Changed link prediction

(c) Node classification (d) Node prediction

(e) Graph reconstruction (f) Anomaly detection

Fig. 7. Negative ratio sensitivity analysis.

7 CONCLUSIONS

In this article, we have proposed a model, M-DHIN, to learn representations of dynamic heterogeneous information networks (HINs). First, the initial embedding of a dynamic HIN is obtained via a metagraph-based complex embedding mechanism. After this, a change dataset is introduced that includes the nodes that have experienced one of four scenarios of evolution, that is, triadic closure, triadic open, newly arrived, or newly deleted. After that we only train on the change dataset, which makes M-DHIN scalable. Additionally, we enable M-DHIN to predict the future network structure through an LSTM-based deep autoencoder. It processes historical information via an LSTM encoder and generates the predicted graph.

We have compared M-DHIN with state-of-the-art baselines on four real-world datasets. M-DHIN significantly and consistently outperforms these baselines on six tasks, i.e., link prediction, changed link prediction, node classification, node prediction, graph reconstruction, and anomaly detection. Our experimental results demonstrate that M-DHIN is able to represent dynamic HINs in an effective and comprehensive manner, for predicting the future evolution and detecting abnormal changes. In addition, we have also shown that M-DHIN is robust w.r.t. the choice of hyperparameters.

M-DHIN can be applied to many real-world applications. For example, it can be used for recommendation applications. In a shopping context, given a user's historical click records, M-DHIN can help predict items of interest, just like the link prediction task. For social networks, M-DHIN can help to recommend users that might be of interest to a given user, just like the node prediction task. In addition, M-DHIN can help to detect anomalies in dynamic networks; real-world networks are always evolving and M-DHIN can help to guarantee that changes of the network are intended.

M-DHIN can be improved in a number of ways. First, M-DHIN lacks in terms of the interpretability of the network dynamics. Hence, in future work we intend to provide more insights into the evolution of a graph, so as to better understand its temporal dynamics. Second, we aim to add automatic hyperparameter optimization and reduce the number of parameters to further improve M-DHIN's accuracy and efficiency. Third, future work also includes ways of automatically learning useful metagraphs in diverse types of dynamic HINs. Fourth, it is of interest to see whether recent graph neural network models are applicable to dynamic HINs and how they perform in comparison with M-DHIN. Fifth, we intend to study alternative ways of setting the time resolution so as to replace the analysis of snapshots at discrete time points and thereby further improve the efficiency. And finally, our decision to only update the embeddings in the change set may not be able to express the influence of the changing structure at the macro level, especially for influential nodes. As a potential future improvement, we could first identify influential nodes and then, given an influential node, we could update the embeddings of all its neighboring nodes.

ACKNOWLEDGMENTS

We are grateful to our anonymous reviewers for their constructive feedback.

REFERENCES

[1] Amr Ahmed, Nino Shervashidze, Shravan M. Narayanamurthy, Vanja Josifovski, and Alexander J. Smola. 2013. Distributed large-scale natural graph factorization. In *22nd International World Wide Web Conference (WWW'13)*. 37–48. https://doi.org/10.1145/2488388.2488393

[2] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: A survey. *Data Mining and Knowledge Discovery* 29, 3 (2015), 626–688. https://doi.org/10.1007/s10618-014-0365-y

[3] Mikhail Belkin and Partha Niyogi. 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic] (NIPS'01)*. 585–591. http://papers.nips.cc/paper/1961-laplacian-eigenmaps-and-spectral-techniques-for-embedding-and-clustering.

[4] Ranran Bian, Yun Sing Koh, Gillian Dobbie, and Anna Divoli. 2019. Network embedding and change modeling in dynamic heterogeneous networks. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'19)*. 861–864. https://doi.org/10.1145/3331184.3331273

[5] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. GraRep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management (CIKM'15)*. 891–900. https://doi.org/10.1145/2806416.2806512

[6] Michael A. A. Cox and Trevor F. Cox. 2008. *Multidimensional Scaling*. Springer Berlin. 875–878.

[7] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 135–144.

[8] Mohammed Elseidy, Ehab Abdelhamid, Spiros Skiadopoulos, and Panos Kalnis. 2014. GRAMI: Frequent subgraph and pattern mining in a single large graph. *PVLDB* 7, 7 (2014), 517–528. https://doi.org/10.14778/2732286.2732289

[9] Yujie Fan, Yanfang Ye, Qian Peng, Jianfei Zhang, Yiming Zhang, Xusheng Xiao, Chuan Shi, Qi Xiong, Fudong Shao, and Liang Zhao. 2020. Metagraph aggregated heterogeneous graph neural network for illicit traded product identification in underground market. In *20th IEEE International Conference on Data Mining (ICDM'20)*, Claudia Plant, Haixun Wang, Alfredo Cuzzocrea, Carlo Zaniolo, and Xindong Wu (Eds.). IEEE, 132–141. https://doi.org/10.1109/ICDM50108.2020.00022

[10] Yang Fang, Xiang Zhao, Peixin Huang, Weidong Xiao, and Maarten de Rijke. 2019. M-HIN: Complex embeddings for heterogeneous information networks via metagraphs. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'19)*. 913–916. https://doi.org/10.1145/3331184.3331281

[11] Tao-Yang Fu, Wang-Chien Lee, and Zhen Lei. 2017. HIN2Vec: Explore meta-paths in heterogeneous information networks for representation learning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM'17)*. 1797–1806.

[12] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'18)*. 1416–1424. https://doi.org/10.1145/3219819.3219947

[13] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. 2020. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge Based Systems* 187 (2020), 104816–104824. https://doi.org/10.1016/j.knosys.2019.06.024

[14] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. DynGEM: Deep embedding method for dynamic graphs. *CoRR* abs/1805.11273 (2018). arXiv:1805.11273 http://arxiv.org/abs/1805.11273.

[15] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 855–864.

[16] Zhipeng Huang and Nikos Mamoulis. 2017. Heterogeneous information network embedding for meta path based proximity. *CoRR* abs/1701.05291 (2017).

[17] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR'15), Conference Track Proceedings*. http://arxiv.org/abs/1412.6980.

[18] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations (ICLR'17), Conference Track Proceedings*. https://openreview.net/forum?id=SJU4ayYgl.

[19] Zhihui Li, Feiping Nie, Xiaojun Chang, Yi Yang, Chengqi Zhang, and Nicu Sebe. 2018. Dynamic affinity graph construction for spectral clustering using multiple features. *IEEE Transactions on Neural Networks and Learning Systems* 29, 12 (2018), 6323–6332. https://doi.org/10.1109/TNNLS.2018.2829867

[20] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. 2018. Heterogeneous graph neural networks for malicious account detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM'18)*, Alfredo Cuzzocrea, James Allan, Norman W. Paton, Divesh Srivastava, Rakesh Agrawal, Andrei Z. Broder, Mohammed J. Zaki, K. Selçuk Candan, Alexandros Labrinidis, Assaf Schuster, and Haixun Wang (Eds.). ACM, 2077–2085. https://doi.org/10.1145/3269206.3272010

[21] Minnan Luo, Feiping Nie, Xiaojun Chang, Yi Yang, Alexander G. Hauptmann, and Qinghua Zheng. 2018. Adaptive unsupervised feature selection with structure regularization. *IEEE Transactions on Neural Networks and Learning Systems* 29, 4 (2018), 944–956. https://doi.org/10.1109/TNNLS.2017.2650978

[22] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online learning of social representations. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'14)*. 701–710.

[23] Mahmudur Rahman, Tanay Kumar Saha, Mohammad Al Hasan, Kevin S. Xu, and Chandan K. Reddy. 2018. DyLink2Vec: Effective feature representation for link prediction in dynamic networks. *CoRR* abs/1804.05755 (2018). arXiv:1804.05755 http://arxiv.org/abs/1804.05755.

[24] Terry Rooker. 1989. Review of neurocomputing: Foundations of research. *AI Magazine* 10, 4 (1989), 64–66. https://doi.org/10.1609/aimag.v10i4.972

[25] Sam T. Roweis and Lawrence K. Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* 290, 5500 (2000), 2323–2326.

[26] Aravind Sankar, Xinyang Zhang, and Kevin Chen-Chuan Chang. 2019. Meta-GNN: Metagraph neural network for semi-supervised learning in attributed heterogeneous information networks. In *International Conference on Advances in Social Networks Analysis and Mining (ASONAM'19)*, Francesca Spezzano, Wei Chen, and Xiaokui Xiao (Eds.). ACM, 137–144. https://doi.org/10.1145/3341161.3342859

[27] Jingbo Shang, Meng Qu, Jialu Liu, Lance M. Kaplan, Jiawei Han, and Jian Peng. 2016. Meta-path guided embedding for similarity search in large-scale heterogeneous information networks. *CoRR* abs/1610.09769 (2016).

[28] Jian Tang, Meng Qu, and Qiaozhu Mei. 2015. PTE: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1165–1174. https://doi.org/10.1145/2783258.2783307

[29] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web (WWW'15)*. 1067–1077.

[30] Théo Trouillon, Christopher R. Dance, Éric Gaussier, Johannes Welbl, Sebastian Riedel, and Guillaume Bouchard. 2017. Knowledge graph completion via complex tensor factorization. *Journal of Machine Learning Research* 18 (2017), 130:1–130:38.

[31] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1225–1234. https://doi.org/10.1145/2939672.2939753

[32] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S. Yu. 2019. Heterogeneous graph attention network. In *The World Wide Web Conference (WWW'19)*, Ling Liu, Ryen W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia (Eds.). ACM, 2022–2032. https://doi.org/10.1145/3308558.3313562

[33] Duncan J. Watts and Steven H. Strogatz. 1998. Collective dynamics of 'small-world' networks. *Nature* 393 (1998), 440–442.

[34] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla. 2019. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'19)*. 793–803. https://doi.org/10.1145/3292500.3330961

[35] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2018. MetaGraph2Vec: Complex semantic path augmented heterogeneous network embedding. In *Advances in Knowledge Discovery and Data Mining - 22nd Pacific-Asia Conference (PAKDD'18), Proceedings, Part II*. 196–208. https://doi.org/10.1007/978-3-319-93037-4_16

[36] Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. 2018. TIMERS: Error-bounded SVD restart on dynamic networks. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI'18), the 30th Innovative Applications of Artificial Intelligence (IAAI'18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI'18)*. 224–231. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16674.

[37] Le-kui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic network embedding by modeling triadic closure process. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI'18), the 30th Innovative Applications of Artificial Intelligence (IAAI'18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI'18)*. 571–578. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16572.

[38] Linhong Zhu, Dong Guo, Junming Yin, Greg Ver Steeg, and Aram Galstyan. 2016. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Transactions on Knowledge and Data Engineering* 28, 10 (2016), 2765–2777.

Received November 2020; revised June 2021; accepted September 2021