# MACHINE LEARNING TASKS AND REPRESENTATIONS FOR HETEROGENEOUS INFORMATION NETWORKS

YANG FANG

# Machine Learning Tasks and Representations for Heterogeneous Information Networks

Yang Fang

# Machine Learning Tasks and Representations for Heterogeneous Information Networks

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. P.P.C.C. Verbeek
ten overstaan van een door het College voor Promoties
ingestelde commissie,
in het openbaar te verdedigen
op maandag 11 december 2023, te 16:00 uur

door

Yang Fang

geboren te Ningbo

The research was carried out at the Information Retrieval Lab of the University of Amsterdam, supported by the NSFC under grants Nos. 61872446, 62272469, U19B2024.

ISBN: 978-94-6483-588-5

## Acknowledgements

In October 2018, Maarten de Rijke agreed to have me join the Information Retrieval Lab (IRLab) at the University of Amsterdam. There was a lot of excitement for me to have the opportunity to experience a different life. However, after a long preparation and even with the flight booked, COVID-19 struck and changed everything.

I am sincerely grateful that Maarten still offered me the chance to join the group online. Nevertheless, it is a big regret for me that I was not able to receive his guidance offline in the Netherlands. Although there are many inconveniences meeting online, he has always maintained the greatest patience and encouragement for me. I am also impressed by his gentleness and erudition. Every time I communicated with him, I could always be inspired. I sincerely hope to meet Maarten in person in the Netherlands in the near future.

I want to thank my co-promotor Dr. Andrew Yates who helped me to revise the thesis. I also would like to thank my committee members, prof. dr. Evangelos Kanoulas, dr. Mohammad Alian Nejadi, prof. dr. Paul Groth, dr. Qing Liao, dr. Xin Wang, prof. dr. Christof Monz and prof. dr. Zhaochun Ren. Thanks a lot for your time and effort to review my thesis.

I thank my colleagues at the IRLab and I hope to meet them face-to-face in the future. Special thanks to Yangjun Zhang, Jin Huang, Jiahuan Pei, Jie Zhou, Yifan Chen, Wanyu Chen, and Weijia Zhang, who solved many problems for me. I also thank Petra for helping me organizing the online meetings and many other things because of my absence.

I want to thank my parents for their love and support. Last but not least, I want to thank my girlfriend Cuiwei Wu, your support and encouragement to me is irreplaceable. You fit all the good words. I look forward to building a small family with you. You are everything to me.

Yang Fang
Changsha, February 2023

# Contents

# 1 Introduction

Complex information often involves multiple types of objects and relations. Such information can be represented via *heterogeneous information networks* (HINs) [83]. In a HIN different types of nodes (objects) are connected by edges (relations) [93]. Typical HINs involve bibliography networks, social networks, knowledge graphs etc. Previous research typically chooses to build homogeneous information networks to model relations in real-life networks, which only feature a single type of node. Such homogeneous information networks only describe part of the information in a real-life system, and ignore the heterogeneity of different nodes and relations, which will cause a significant loss of information loss that cannot easily be recovered.

In recent years, HINs have drawn more attention from the research community. HINs provide a more natural and complete abstraction of the real world than homogeneous information networks. HINs provide a richer modeling tool than homogeneous information networks. They naturally integrate different types of objects and interactions, while containing more fruitful structural and semantic features. Based on the above advantages, and compared to homogeneous information networks, HINs help to provide more effective solutions for many machine learning tasks, such as search, classification, and prediction tasks [8].

To be able to process a network in a machine learning context, *network representation learning* (NRL), also known as network embedding learning, has been investigated extensively. NRL is aimed at obtaining an embedding of a network in a low-dimensional space, Classical network embedding models like DeepWalk [77], LINE [95], and node2vec [33] have been devised for homogeneous networks, using random walks to capture the structure of networks. However, these methods lack the ability to capture a *heterogeneous* information network with multiple types of objects and relations. Hence, models designed specifically for HINs have been proposed [18, 28, 50]. A central concept here is that of a *metapath*, which is a sequence of node types with edge types in between. To leverage the relationship between nodes and metapaths, different mechanisms have been proposed, such as the heterogeneous SkipGram [18], proximity distance [50], and the Hardmard function [28]. However, because of the limited ability of metapaths to capture the neighborhood structure of a node, the performance of these network representation learning methods is limited.

Recently, graph neural networks (GNNs) have shown promising results on the task of modeling the structure of a network [55, 101, 115]. GNNs usually involve encoders that are able to explore and capture the neighborhood structure around a node, thus improving the performance on representing an HIN. GNNs propagate and aggregate the node features along the network topology, and are realized in an end-to-end semi-supervised manner.

In this thesis, we aim to explore different learning mechanisms for network representation learning to fit into different scenarios and different machine learning tasks of HINs. Specifically, we first study representation learning of dynamic HINs, as real-life networks are always evolving. Then, we pre-train a HIN without using labeled information, and the pre-trained GNN model can easily be adapted to different datasets and different downstream tasks. We also investigate few-shot learning of HINs, where only a handful of labels are given. Last, we make use of the auxiliary textual information to further facilitate the learning of HINs.

## 1.1 Research outline and questions

Throughout this thesis, we intend to answer the following research question, *how to comprehensively mine the information and features of HINs under different scenarios*? Different representation learning mechanisms are proposed in this thesis, so that the corresponding downstream machine learning tasks can be addressed. Specifically, four research aspects of HINs are considered in this thesis, and we formulate each of them as a research question.

**RQ1** How to learn the representation of dynamic HINs?

Real-life information networks are always evolving, with new nodes and edges coming and old nodes and edges deleted. Most traditional NRL models focus on static networks. 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 answer **RQ1**, 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 so-called *metagraphs*. We also propose an LSTM-based deep autoencoder mechanism to enable M-DHIN to predict the future network via history structure evolutions.

**RQ2** How to pre-train HINs?

Traditional NRL models such as GNNs need to be trained in an end-to-end manner with supervised information for a task, and the model learned on one dataset cannot easily be transferred to other, out-of-domain datasets. For different datasets and tasks, the methods listed above need to be re-trained all over. Additionally, in many real-life datasets, the amount of available labeled data is rarely sufficient for effective training. The shortcomings listed above could be addressed by pre-training techniques that have been widely used in natural language processing and computer vision. Then, how can we pre-train graph-like data?

To answer **RQ2**, we first design two pre-training tasks that are applied on large datasets mining self-supervision information. Then, for a specific downstream task on a specific dataset, we use fine-tuning techniques with few task-specific parameters, so that the pre-trained model could be fast adapted to new tasks and new datasets.

**RQ3** How to learn the representation of HINs in a few-shot setting?

During traditional representation learning processes, it is taken for granted that the majority of labels in the network is available, and the GNNs are trained in a supervised manner. In practice, however, it is common that only a handful of labels are given, which poses serious challenges to keeping up the performance. For few-shot learning, meta-learning approaches have been studied extensively in computer vision, so is it possible to apply meta-learning techniques on graph data?

To answer **RQ3**, we propose a unified meta-learning framework that takes subgraphs as training samples to form meta-training and meta-testing datasets. A heterogeous GNN module is used as a base model to fully capture heterogeneous information. We also adopt a GAN-based contrastive module to leverage unsupervised information, and a structure module to employ graph structural information. With the help of the meta-training framework, it can be applied across different tasks and graphs.

**RQ4** How to make use of the textual information when learning the representation of HINs?

Most HINs come with textual information, e.g., a title and an abstract of a paper node in an academic network, which can provide fruitful additional information for downstream tasks. Most current work on HINs ignores such textual information and maps the node of a graph into low-dimensional representations based only on structural information. Existing textual network embedding models are all designed for a supervised scenario, which requires abundant labeled data for training. In other words, they are not applicable to the few-shot learning setting. However, in real-life applications, it is common that only a handful of labels are available, which poses serious challenges to keeping up the performance. Second, these methods are all originally designed for homogeneous networks, with no prior work yet trying to solve few-shot learning issues for textual HINs.

To answer **RQ4**, we propose a learnable continuous prompt learning framework to solve the few-shot problem for textual HINs. Specifically, our proposal contains a text encoder to leverage textual information, a graph encoder that encodes structural and heterogeneous features, along with self-supervised information. We then introduce a contrastive learning mechanism to align the text and graph embeddings.

## 1.2 Main contributions

In this section, we summarize the main contributions of this thesis.

### 1.2.1 Algorithmic contributions

We propose several novel network embedding algorithms to fit into four different HIN representation scenarios.

(1) Representation learning for Dynamic HINs

   (a) A novel dynamic network embedding model, M-DHIN, that learns representations of every snapshot of a dynamic HIN via metagraph-based complex embeddings on dynamic datasets.

   (b) An LSTM-based deep autoencoder mechanism to enable M-DHIN to predict the future network via history structure evolutions.

(2) Pre-training on HINs

   (a) A pre-training and fine-tuning framework PF-HIN to mine information contained in a HIN; PF-HIN is transferable to different downstream tasks and to datasets of different domains.

   (b) A deep bi-directional transformer encoder to capture the structural features of a HIN; the architecture of PF-HIN is a variant of a GNN.

   (c) Two pre-training mechanisms, i.e., type-based masked node modeling and adjacent node prediction; both help PF-HIN to capture heterogeneous node features and relationships between nodes.

(3) Representation learning on few-shot HINs

   (a) A meta-learning model META-HIN to deal with few-shot leaning problems in HINs.

   (b) A sampling strategy which extracts subgraphs to be trained so that META-HIN is applicable to three tasks and transferable across different HINs.

   (c) A structural module, a heterogeneous module, and a GAN-based contrastive module to capture the structural information, heterogeneous features, and unlabeled information of a subgraph, respectively.

(4) Representation learning on textual HINs

   (a) A prompt learning framework P-HIN to leverage textual information in textual HINs, and to deal with few-shot leaning problems simultaneously.

   (b) A graph encoder that can capture structural and heterogeneous features of HINs, while preserving the node-level and edge-level self-supervised information.

### 1.2.2 Empirical contributions

To verify the effectiveness of our algorithms, we conduct extensive experiments to evaluate the proposed algorithms for the considered tasks.

(1) Representation learning on Dynamic HINs

   (a) An empirical evaluation of M-DHIN on six downstream tasks, and a comparison against state-of-the-art models.

  (b) An analysis of the impact of different modules of M-DHIN.
  (c) An analysis of the impact of parameters, i.e., the choice of dimension and the negative ratio when sampling.

(2) Pre-training on HINs

  (a) An empirical evaluation of PF-HIN on four downstream tasks and four datasets, and a comparison against state-of-the-art models.
  (b) An analysis of the impact of different modules of PF-HIN.
  (c) An analysis of the impact of parameters, i.e., the maximum length of the input sequence and the dimension of the node embedding.

(3) Representation learning on few-shot HINs

  (a) An empirical evaluation of META-HIN on three downstream tasks, and a comparison against state-of-the-art models.
  (b) An analysis of the impact of different modules of META-HIN.
  (c) An analysis of the impact of parameters, i.e., the maximum number of nodes of a subgraph, and the number of shots used for meta-learning.

(4) Representation learning on textual HINs

  (a) An empirical evaluation of P-HIN on three downstream tasks, and a comparison against state-of-the-art models.
  (b) An analysis of the impact of different modules of P-HIN.
  (c) An analysis of the impact of parameters, i.e., the number of the text tokens, and the number of shots used for training.

## 1.3 Thesis overview

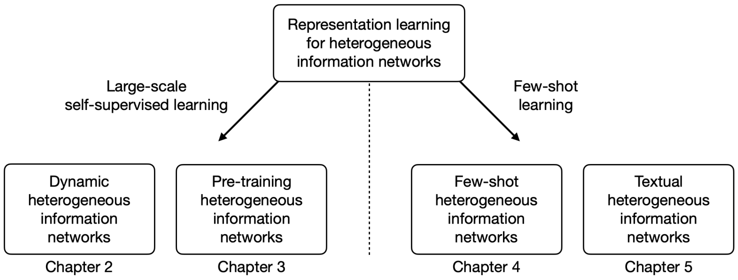
In this section, we present an overview of the remaining chapters. In Chapter 2 to 5, we introduce four algorithms to handle four different heterogeneous information network scenarios with different downstream tasks. Then we empirically analyze the performance of the proposed algorithms against state-of-the-art alternatives. Specifically, the thesis is organized as follows.

In Chapter 2, 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.

In Chapter 3, we propose a self-supervised pre-training and fine-tuning framework, PF-HIN, to capture the features of a heterogeneous information network. Unlike traditional network representation learning models that have to train the entire model all over again for every downstream task and dataset, PF-HIN only needs to fine-tune the model and a small number of extra task-specific parameters, thus improving model efficiency and effectiveness. During pre-training, we first transform the neighborhood of a given node into a sequence. PF-HIN is pre-trained based on two self-supervised tasks, masked node modeling and adjacent node prediction. We adopt deep bi-directional transformer encoders to train the model, and leverage factorized embedding parameterization and cross-layer parameter sharing to reduce the parameters. In the fine-tuning stage, we choose four benchmark downstream tasks, i.e., link prediction, similarity search, node classification, and node clustering. PF-HIN outperforms state-of-the-art alternatives on each of these tasks, on four datasets.

In Chapter 4, we design a meta-learning framework, called META-HIN, for few-shot learning problems on HINs. To the best of our knowledge, we are the first to design a specific algorithm to address the few-shot problem on HINs. Unlike most previous models, which focus on a single task on a single graph, META-HIN is able to deal with three tasks (i.e., node classification, link prediction, and anomaly detection) across multiple graphs. Subgraphs are sampled to build the support and query set. Before being processed by the meta-learning module, subgraphs are modeled via a structure module to capture structural features. Then, a heterogeneous GNN module is used as the base model to express features of subgraphs. We also design a GAN-based contrastive learning module that is able to exploit unsupervised information of the subgraphs. In our experiments, we fuse several datasets from multiple domains to verify META-HIN's broad applicability in a multiple-graph scenario. META-HIN consistently and significantly outperforms state-of-the-art alternatives on every task and across all datasets that we consider.

In Chapter 5, we propose a prompt-learning framework P-HIN that provides a new angle to leverage textual information and that fits few-shot problems. To the best of our knowledge, we are among the first to introduce and exploit the idea of prompt learning in the context of textual HINs. The proposed framework P-HIN is composed of a text encoder and a graph encoder, and utilizes contrastive learning to align the graph-text pair. This pre-training operation naturally fits our few-shot learning setting. For the graph encoder, we introduce two graph pre-training tasks, masked node modeling and edge reconstruction, to exploit self-supervised information. During optimization, instead of handcrafted prompts, we use a learnable continuous text that enables more efficient


Representation learning for heterogeneous information networks
Large-scale self-supervised learning
Few-shot learning
Dynamic heterogeneous information networks
Chapter 2
Pre-training heterogeneous information networks
Chapter 3
Few-shot heterogeneous information networks
Chapter 4
Textual heterogeneous information networks
Chapter 5
Figure 1.1: A visual map of the thesis.

and task-relevant transfer to downstream datasets. We consider a residual connection to use the context from the graph to prompt the text encoder. In experiments, P-HIN consistently and significantly outperforms state-of-the-art alternatives on all real-life datasets.

In Chapter 6, we summarize the thesis and provide some future research directions that build on the results in the thesis.

Figure 1.1 provides a graphical overview of the thesis. The chapters can be read independently of each other.

## 1.4 Origins

The main research chapters in the thesis are based on the following papers:

**Chapter 2** is based on the following paper:

- Y. Fang, X. Zhao, P. Huang, W. Xiao, and M. de Rijke. Scalable representation learning for dynamic heterogeneous information networks via metagraphs. *ACM Trans. Inf. Syst.*, 40(4):64:1–64:27, 2022.

YF designed the model, implemented it, ran experiments and did most of the writing. All authors contributed to design and discussion of the model. MdR and XZ contributed to the writing.

**Chapter 3** is based on the following paper:

- Y. Fang, X. Zhao, Y. Chen, W. Xiao, and M. de Rijke. PF-HIN: Pre-training for heterogeneous information networks. *IEEE Trans. Knowl. Data Eng.*, 35(8):8372–8385.

YF designed the model, implemented it, ran experiments and did most of the writing. All authors contributed to design and discussion of the model. MdR and XZ contributed to the writing.

**Chapter 4** is based on the following paper:

- Y. Fang, X. Zhao, W. Xiao, and M. de Rijke. Few-shot learning for heterogeneous information networks. *ACM Trans. Inf. Syst.*, Under review.

YF designed the model, implemented it, ran experiments and did most of the writing. All authors contributed to design and discussion of the model. MdR and XZ contributed to the writing.

**Chapter 5** is based on the following paper:

- Y. Fang, X. Zhao, W. Xiao, and M. de Rijke. Prompt learning for textual heterogeneous information networks. *IEEE Trans. Knowl. Data Eng.*, Under review.

YF designed the model, implemented it, ran experiments and did most of the writing. All authors contributed to design and discussion of the model. MdR and XZ contributed to the writing.

The writing of the thesis also benefited from work on the following publications:

- P. Huang, X. Zhao, M. Hu, Y. Fang, X. Li, and W. Xiao. Extract-select: A span selection framework for nested named entity recognition with generative adversarial training. In *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 85–96. Association for Computational Linguistics, 2022.
- Q. Guo, X. Zhao, Y. Fang, S. Yang, X. Lin, and D. Ouyang. Learning hypersphere for few-shot anomaly detection on attributed networks. In *CIKM 2022: 31st ACM International Conference on Information and Knowledge Management*, pages 635–645. ACM, 2022.

In the following chapter, we will introduce how to learn the representation of dynamic HINs.

# 2 Dynamic Representation Learning

This chapter is set up to address **RQ1**:

**RQ1** How to learn the representation of dynamic heterogeneous information networks (HINs)?

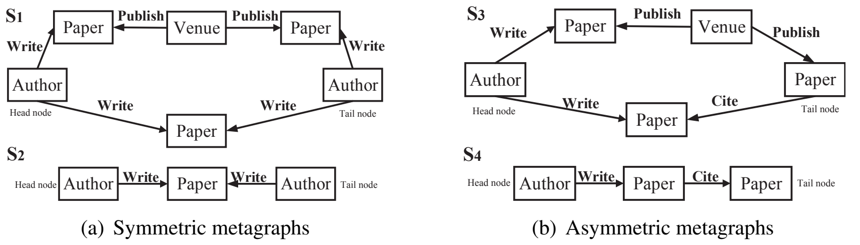
Particularly, we focus on how to represent a dynamic heterogeneous information networks. Most research on representation learning for HINs focuses on static HINs. In practice, however, networks are dynamic and subject to constant change. In this chapter, we propose a novel and scalable representation learning model, M-DHIN, to capture the evolution of a dynamic HIN. A dynamic HIN is regarded as a series of snapshots with different time stamps. We first use a static embedding method based on metagraphs to learn the initial embeddings of a dynamic HIN. Afterwards, we build a so-called *change dataset* that only records the evolution of the nodes. We also introduce an LSTM based deep autoencoder model that enables M-DHIN to predict future 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.

## 2.1 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., 33, 77, 95] 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

This chapter was published as Y. Fang, X. Zhao, P. Huang, W. Xiao, and M. de Rijke. Scalable representation learning for dynamic heterogeneous information networks via metagraphs. *ACM Trans. Inf. Syst.*, 40(4):64:1–64:27, 2022.



(a) Symmetric metagraphs (b) Asymmetric metagraphs

Figure 2.1: Examples of metagraphs.

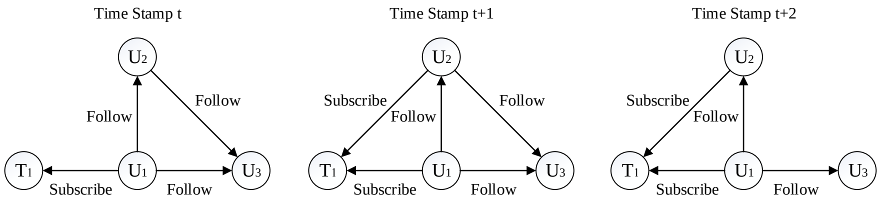


Figure 2.2: An example of a dynamic HIN model of a social network.

between HIN nodes [18, 28, 50]. Unlike static network embeddings, the techniques for dynamic HINs need to be incremental and scalable so as to be able 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 [23]. 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 2.1. When training the model, we observe that the structure of metagraphs can be either symmetric or asymmetric, as illustrated in Figures 2.1(a) and 2.1(b), respectively. To better represent HIN nodes, we incorporate a complex space oriented embedding scheme [98] 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 [107] and we describe their evolution as *triadic closure* and *opening processes* [125]. Figure 2.2 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 $\mathrm{U}_3$, 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 2.2 from time $t$ to $t + 1$, only $\mathrm{U}_1$, $\mathrm{U}_2$ and $\mathrm{T}_1$ will be included in the change dataset, and they will become a metagraph instance to be trained. From time $t + 1$ to $t + 2$, only $\mathrm{U}_1$, $\mathrm{U}_2$ and $\mathrm{U}_3$ 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 contains 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.

The main contributions of the chapter 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 chapter is structured as follows. We introduce related work in Section 2.2, and present the preliminaries in Section 2.3. Then we present M-DHIN, together with a theoretical analysis, in Section 2.4. Next, we introduce our experimental setup in Section 2.5 and present our experimental results in Section 2.6. We conclude in Section 2.7.

## 2.2 Related work

### 2.2.1 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, 11, 85, 103], 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 [95].

To preserve the local and global structure of a network, some research proposes to utilize random walks or paths of a network. DeepWalk [77] first extracts random walks from a network and then applies the SkipGram model on random walks to generate network embeddings. LINE [95] employs both first-order and second-order proximities so as to capture both local and neighboring structure. node2vec [33] harnesses a biased random walk sampling strategy to represent the network structure, which is more expressive than DeepWalk. GrapRep [9] 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 [103] 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 [94] 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 [18] uses a heterogeneous SkipGram by setting the context window limited to one type. HINE [50] 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 [87] proposes a metapath guided sampling strategy to improve the model efficiency while its objective function is partially defined. HIN2Vec [28] 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 [116] has been 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 [29, 55, 115]. HetGNN [115] groups nodes according to their type and applies Bi-LSTM on each group. GEM [66] learns representations from heterogeneous account-device graphs to detect the malicious account in static setting. HAN [105] proposes to represent a heterogeneous network based on the hierarchical attention, including node-level and semantic-level attentions. Meta-GNN [86] makes use of metagraphs to extract diverse semantics, however, it does not take symmetric and asymmetric metagraphs into consideration. PIdentifier [21] 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.

### 2.2.2 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 [120, 126]. Li et al. [63] 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. [68] realize unsupervised feature selection by building adaptive reconstruction graph meanwhile considering its multiconnected-components (multi-cluster) structure. DynGEM [30] proposes an autoencoder mechanism to capture the dynamics, but it only utilizes one previous time step snapshot to generate the current graph embeddings. TIMERS [120] leverages incremental Singular Value Decomposition (SVD) to update the network embeddings, and returns SVD values when the error is beyond the threshold. DynamicTraid [125] only uses the triadic closure process to describe the evolution of a dynamic network and it is designed for homogeneous networks. DYLINK2VEC [81] 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 it is unable to predict the future network. Dyngraph2vec [31] 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, 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 to predict the future structure of the HIN via an LSTM-based deep autoencoder.

## 2.3 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 2.1 lists the main terms and notation used.

#### Dynamic heterogeneous information network

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 is 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

Table 2.1: 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 $t$ |
| $a_u = [a_u^1, \dots, a_u^t]$ | 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 |

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 $\{G^1, \dots, G^{Time}\}$. 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 $T_E$ remain unchanged during the entire evolution of the network. Figure 2.2 illustrates an example of a dynamic HIN.

### 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 2.1, metagraphs can be classified as symmetric and asymmetric ones; we will deal with both conditions in the proposed model.

### Dynamic Heterogeneous information network representation learning

Given a series of dynamic networks $\{G^1, \dots, G^{Time}\}$, *dynamic HIN representation learning* is to learn the dynamic low-dimensional vectors of nodes $Y^t \in \mathbb{C}^{|V^t| \times d}$, $d$ 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.

## 2.4 Proposed model

In this section, we detail our proposed model M-DHIN in detail. An overview of M-DHIN is provided in Figure 2.3. 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 $t+1$.

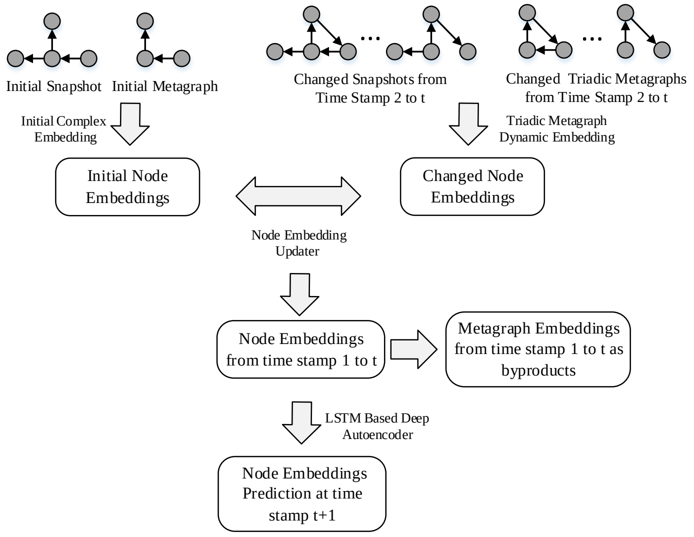

Initial Snapshot
Initial Metagraph
Changed Snapshots from Time Stamp 2 to t
Changed Triadic Metagraphs from Time Stamp 2 to t
Initial Complex Embedding
Triadic Metagraph Dynamic Embedding
Initial Node Embeddings
Changed Node Embeddings
Node Embedding Updater
Node Embeddings from time stamp 1 to t
Metagraph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

Figure 2.3: An overview of M-DHIN.

## 2.4.1 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 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

$$P(s|u, v) = \sigma(\mathbf{X}_{uv}), \tag{2.1}$$

where $X \in \mathbb{R}^{n\times n}$ is a score matrix, $n$ is the number of training nodes, and $\sigma$ 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 2.1(a). Correspondingly, exchanging the head and tail node will change the semantics of an asymmetric metagraph, as shown in Figure 2.1(b).

To address this issue, we adapt the schema of complex knowledge embeddings in [98] to fit the task of network embeddings. For a HIN triplet $(u, s, v)$, its complex embedding is denoted as $\mathbf{u} = \text{Re}(\mathbf{u}) + \imath\text{Im}(\mathbf{u})$, $\mathbf{v} = \text{Re}(\mathbf{v}) + \imath\text{Im}(\mathbf{v})$ and $\mathbf{s} = \text{Re}(\mathbf{s}) + \imath\text{Im}(\mathbf{s})$, where $\text{Re}(\mathbf{x}) \in \mathbb{R}^d$ and $\text{Im}(\mathbf{x}) \in \mathbb{R}^d$ represent the real and imaginary part of the vector $\mathbf{x} \in \mathbb{C}^d$, respectively. Afterwards, we introduce the Hadamard function to capture the relationship of $u$, $v$ and $s$ in complex spaces, which is denoted as

$$\mathbf{X}_{uv} = \sum \mathbf{u} \odot \mathbf{s} \odot \bar{\mathbf{v}}, \tag{2.2}$$

where $\bar{\mathbf{v}}$ is the complex conjugate form of $\mathbf{v}$, and $\odot$ is the element-wise product. However, the sigmoid function in Eq. 2.1 cannot be applied in complex spaces, 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

$$\mathbf{X}_{uv} = \text{Re}\left(\sum \mathbf{u} \odot \mathbf{s} \odot \bar{\mathbf{v}}\right). \tag{2.3}$$

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

$$\begin{aligned} f_{\mathbf{s}}(\mathbf{u},\mathbf{v}) &= \text{Re}(\langle \mathbf{u}, \mathbf{s}, \bar{\mathbf{v}} \rangle) \\ &= \langle \text{Re}(\mathbf{u}), \text{Re}(\mathbf{s}), \text{Re}(\mathbf{v}) \rangle + \langle \text{Im}(\mathbf{u}), \text{Re}(\mathbf{s}), \text{Im}(\mathbf{v}) \rangle \\ &\quad + \langle \text{Re}(\mathbf{u}), \text{Im}(\mathbf{s}), \text{Im}(\mathbf{v}) \rangle - \langle \text{Im}(\mathbf{u}), \text{Im}(\mathbf{s}), \text{Re}(\mathbf{v}) \rangle, \end{aligned} \tag{2.4}$$

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.

Eq. 2.4 is able to handle asymmetric metagraphs thanks to the complex conjugate of one of the embeddings. In addition, if 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 $\mathbf{X}_{\mathbf{s}}$ as the sum of an antisymmetric matrix $\text{Im}(\mathbf{U}\text{diag}(-\text{Im}(\mathbf{s}))\bar{\mathbf{V}})$ and a symmetric matrix $\text{Re}(\mathbf{U}\text{diag}(\text{Re}(\mathbf{s}))\bar{\mathbf{V}})$. From this we can see that metagraph embeddings naturally act as weights on each latent dimension, i.e., $\text{Im}(\mathbf{s})$ over the antisymmetric, imaginary part of $\overline{\langle \mathbf{u}, \mathbf{v} \rangle}$ and $\text{Re}(\mathbf{s})$ 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 $\text{Im}(\langle \mathbf{u}, \mathbf{v} \rangle)$ is antisymmetric, while $\text{Re}(\langle \mathbf{u}, \mathbf{v} \rangle)$ 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 [20] 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.

**Algorithm 1:** The initial complex embedding algorithm.

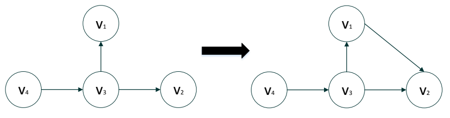
```
Input :
        (1) The initial HIN at the first time stamp $G^1 = (V^1, E^1)$;
        (2) Maximum number of iterations: $MaxIter$;
Output :
        Node embeddings $Y^1$ and metagraph embeddings $Z^1$;
1  $S^1 \leftarrow$ generate the initial metagraph set using GRAMI;
2  $(u, s, v) \leftarrow$ 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 funtion to evaluate if a triplet holds using
         Eq. 2.4;
6      $P(s|u, v) \leftarrow$ compute probability of a triplet being positive based on
         score function using Eq. 2.11;
7      $\log O_{(u,s,v)} \leftarrow$ generate the objective function based on $P(s|u, v)$ using
         Eq. 2.10;
8      $\mathbf{u}, \mathbf{s}, \mathbf{v} \leftarrow$ update embeddings of nodes and metagraphs using Eq. 2.14;
9      $Iterations \leftarrow Iterations + 1$
10 until $convergence\ or\ Iterations \geq MaxIterations$;
```

### 2.4.2 Dynamic embedding mechanism via triadic evolution processes

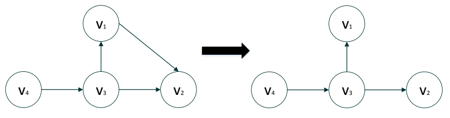
After processing the full HIN to obtain the initial embeddings, we leverage triadic node blocks to further capture the structural evolutions of the dynamic HIN. A *triad* is a set containing three nodes. If every node is connected to each other, it is called a *closed triad* and if there are only two edges between these three nodes, the triad is called an *open triad.* As mentioned in Section 2.1, 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 [125]. 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

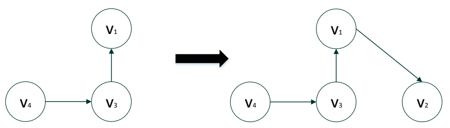
$$\begin{aligned} S^t_{change} = \ & \big\{s : \{(v_1, v_2) \notin E^t, (v_1, v_3) \in E^t, (v_2, v_3) \in E^t\} \neq \emptyset; \\ & \{(v_1, v_2) \in E^{t+1}, (v_1, v_3) \in E^{t+1}, (v_2, v_3) \in E^{t+1}\} \neq \emptyset\big\}. \end{aligned} \quad (2.5)$$



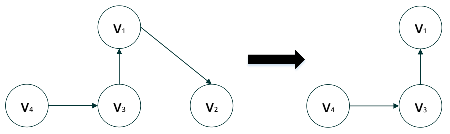
(a) Added edges form a triadic closure process. $v_1$ and $v_2$ will be included in $S^t_{change}$.



(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 $S^t_{change}$.

Figure 2.4: The formation of the change set.

(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

$$\begin{aligned} S^t_{change} = \, & \{ s : (v_1, v_2) \in E^t, (v_1, v_2) \notin E^{t+1}, \\ & \{(v_1, v_2) \in E^t, (v_1, v_3) \in E^t, (v_2, v_3) \in E^t\} \neq \emptyset; \\ & \{(v_1, v_2) \in E^{t+1}, (v_1, v_3) \in E^{t+1}, (v_2, v_3) \in E^{t+1}\} = \emptyset \} . \end{aligned} \tag{2.6}$$

(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

$$S^t_{change} = \{s : v_1, v_2 \in V^{t+1}, v_1 \in V^t, v_2 \notin V^t, (v_1, v_2) \notin E^t\}. \tag{2.7}$$

(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

$$S^t_{change} = \{s : v_1, v_2 \in V^t, v_2 \notin V^{t+1}, (v_1, v_2) \in E^t\}. \tag{2.8}$$

Figure 2.4 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 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

$$O_{(u,s,v)} = \begin{cases} P(s|u,v), & \text{if } H_{uv} = 1; \\ 1 - P(s|u,v), & \text{if } H_{uv} = 0. \end{cases} \tag{2.9}$$

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

$$\log O_{(u,s,v)} = H_{uv} \log P(s|u,v) + (1 - H_{uv}) \log[1 - P(s|u,v)], \tag{2.10}$$

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

$$P(s|u,v) = \text{sigmoid}(f_{\mathbf{s}}(\mathbf{u}, \mathbf{v})). \tag{2.11}$$

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

$$\log O_{(u,s,v)} = \log P(s|u,v). \tag{2.12}$$

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

$$\log O_{(u,s,v)} = \log[1 - P(s|u, v)]. \tag{2.13}$$

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 stochastic gradient descent (SGD) algorithm [84] to maximize the above objective function with Adaptive Moment Estimation (Adam) [54]. Specifically, for each training entry $(u, s, v, H_{uv})$, it goes backwards 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.

$$\mathbf{u} := \mathbf{u} + \frac{d\log O_{(u,s,v)}}{d\mathbf{u}} \tag{2.14a}$$

$$\mathbf{v} := \mathbf{v} + \frac{d\log O_{(u,s,v)}}{d\mathbf{v}} \tag{2.14b}$$

$$\mathbf{s} := \mathbf{s} + \frac{d\log O_{(u,s,v)}}{d\mathbf{s}}. \tag{2.14c}$$

Algorithm 2 summarizes the dynamic embedding algorithm.

### 2.4.3 Graph prediction via LSTM-based deep autoencoders

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 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 2.5 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

**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 Eq. 2.5, 2.6, 2.7 and 2.8;
2  $(u, s, v) \leftarrow$ generate the HIN triplets based on the changed dataset $S^t_{change}$
     for training;
3  $Iterations \leftarrow 0$;
4  repeat
5      $f_{\mathbf{s}}(\mathbf{u}, \mathbf{v}) \leftarrow$ calculate score function to evaluate if a triplet holds using
         Eq. 2.4 ;
6      $P(s|u, v) \leftarrow$ compute probability of a triplet being positive based on
         score function using Eq. 2.11;
7      $\log O_{(u,s,v)} \leftarrow$ generate the objective function based on $P(s|u, v)$ using
         Eq. 2.10;
8      $\mathbf{u}, \mathbf{s}, \mathbf{v} \leftarrow$ update embeddings of nodes and metagraphs using Eq. 2.14;
9      $Iterations \leftarrow Iterations + 1$
10 until *convergence or Iterations* $\geq$ *MaxIterations*;
```

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 Algorithm 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

$$y_u^{(1)} = f_a(W_A^{(1)} a_u + b_a^{(1)}), \tag{2.15}$$

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

$$y_u^{(k)} = f_a(W_A^{(k)} y_u^{(k-1)} + b_a^{(k)}). \tag{2.16}$$

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

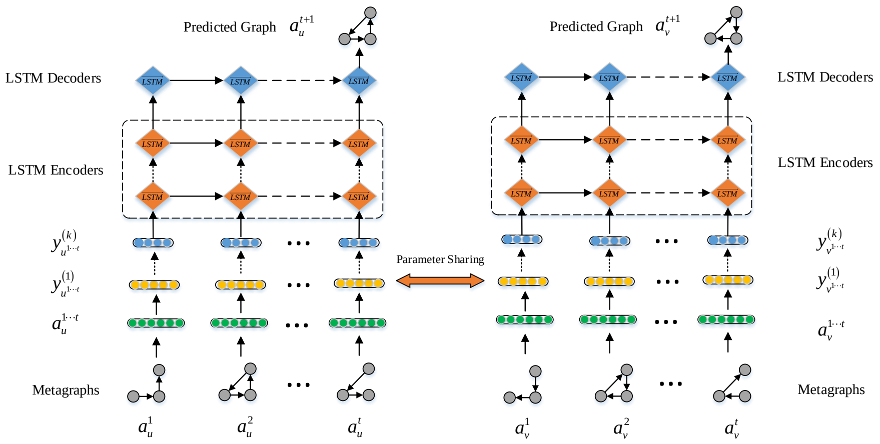

Predicted Graph $a_u^{t+1}$
Predicted Graph $a_v^{t+1}$
LSTM Decoders
LSTM Encoders
LSTM
$y_{u^{1\cdots t}}^{(k)}$
$y_{u^{1\cdots t}}^{(1)}$
$a_u^{1\cdots t}$
$y_{v^{1\cdots t}}^{(k)}$
$y_{v^{1\cdots t}}^{(1)}$
$a_v^{1\cdots t}$
Parameter Sharing
Metagraphs
$a_u^1$ $a_u^2$ $a_u^t$
$a_v^1$ $a_v^2$ $a_v^t$

Figure 2.5: An LSTM-based deep autoencoder.

representation is computed as

$$i_{u^t}^{(k+1)} = \delta(W_i^{(k+1)}[a_u^t, y_{u^{t-1}}^{(k+1)}] + b_i^{(k+1)}) \tag{2.17a}$$
$$f_{u^t}^{(k+1)} = \delta(W_f^{(k+1)}[a_u^t, y_{u^{t-1}}^{(k+1)}] + b_f^{(k+1)}) \tag{2.17b}$$
$$z_{u^t}^{(k+1)} = \tanh(W_c^{(k+1)}[a_u^t, y_{u^{t-1}}^{(k+1)}] + b_c^{(k+1)}) \tag{2.17c}$$
$$c_{u^t}^{(k+1)} = f_{u^t}^{(k+1)} \odot c_{u^{t-1}}^{(k+1)} + i_{u^t}^{(k+1)} \odot z_{u^t}^{(k+1)} \tag{2.17d}$$
$$o_{u^t}^{(k+1)} = \delta(W_o^{(k+1)}[a_u^t, y_{u^{t-1}}^{(k+1)}] + b_o^{(k+1)}) \tag{2.17e}$$
$$y_{u^t}^{(k+1)} = o_{u^t}^{(k+1)} \odot \tanh(c_{u^t}^{(k+1)}), \tag{2.17f}$$

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 output gate, $\delta$ 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 representation dimension of $k+1$ layer.

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

$$y_{u^t}^{(k+l)} = o_{u^t}^{(k+l)} \odot \tanh(c_{u^t}^{(k+l)}) \tag{2.18a}$$
$$o_{u^t}^{(k+l)} = \delta(W_{LSTM}^{(k+l)}[y_{u^t}^{(k+l-1)}, y_{u^{t-1}}^{(k+l)}] + b_o^{(k+l)}), \tag{2.18b}$$

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:

$$L_t = |(\hat{a}^{t+1} - a^{t+1}) \odot B|_F^2 = |(f(a^1, \ldots, a^t) - a^{t+1}) \odot B|_F^2. \tag{2.19}$$

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 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 2.19, as follows:

$$\frac{\partial L_t}{\partial W_*^{(k+l)}} = \left[2(\hat{a}^{t+1} - a^{t+1}) \odot B\right] \left[\frac{\partial f_a(Y^{(k+l-1)} W_*^{(k+l)} + b^{(k+l)})}{\partial W_*^{(k+l)}}\right], \quad (2.20)$$

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 evolution of changing metagraphs in Algorithm 1 and 2 to form the adjacency matrix. It also makes use of the dynamic embeddings learned in Algorithm 1 and 2 to form $a_v^t$, which is the input of the autoencoder.

## 2.5 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.

### 2.5.1 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 2.2. 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). 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.

[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/

**Algorithm 3:** Graph Prediction via LSTM-based Deep Autoencoder Algorithm.

```
Input :
        (1) The previous HIN from time stamp 1 to time stamp $t$, $\{G^1, G^2, \ldots, G^t\}$;
        (2) The previous embeddings learned from time stamp 1 to time stamps $Y^1, Y^2, \ldots, Y^t$;
        (3) Autoencoder parameters $\theta$;
        (4) Maximum number of iterations: $MaxIter$;
Output :
        Predicted embeddings of nodes $Y^{t+1}$ at time stamp $t+1$ ;
1  $a_u = [a_u^1, \ldots, a_u^t] \leftarrow$ generate the sequential adjacency vectors of $u$ based
   on the metagraphs it belongs.;
2  $\theta \leftarrow$ RandomInit() ;
3  $Iterations \leftarrow 0$;
4  repeat
5      $Y_u^{(k)} \leftarrow$ Generate the output of encoder based on $a_u$, $Y^1, Y^2, \ldots, Y^t$
       and $\theta$ using Eq. 2.15 and 2.16;
6      $y_{u^t}^{(k+l)}, o_{u^t}^{(k+l)} \leftarrow$ generate the final output of LSTM layers using
       Eq. 2.18b;
7      $L_t \leftarrow$ compute the objective function using Eq. 2.19;
8      $\theta \leftarrow$ update autoencoder parameters by gradient to back-propagate the
       whole network using Eq. 2.20;
9      $Iterations \leftarrow Iterations + 1$ ;
10 until $convergence$ or $Iterations \geq MaxIterations$;
11 Obtain node embedding $u^{t+1} \leftarrow y_{u^t}^{(k+l)}$
```

- **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

Table 2.2: 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 | | | |

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.

### 2.5.2 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.

### 2.5.3 Baselines

We include two types of baselines, one consists of static embedding methods, the other consists of dynamic embedding methods. For the static embeddings methods, we consider both homogeneous and heterogeneous methods. DeepWalk [77] and node2vec [33] were originally designed to represent the homogeneous network. metapath2vec [18] and MetaGraph2Vec [28] 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 Skip-Grams 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 [125] describes the evolution of a network based merely on the triadic closure process and it is designed for homogeneous networks. $\beta_0$ and $\beta_1$ 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 [30] uses a deep autoencoder to capture the dynamics at time stamp $t$ of a HIN only using the snapshot at time stamp $t-1$. $\alpha, v_1, v_2$ are relative weight hyperparameters chosen via grid search from $\alpha \in \{10^{-6}, 10^{-5}\}$, $v_1 \in \{10^{-4}, 10^{-6}\}$ $v_2 \in \{10^{-3}, 10^{-6}\}$
- dyngraph2vec [31] 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 [31]. 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.

### 2.5.4 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 $\{3, 4, 5, 6, 7\}$, number of autoencoder layers from $\{2, 3, 4\}$, the number of LSTM layers from $\{2, 3, 4\}$, 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., 5 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.

## 2.6 Experimental results and analysis

One by one we report on the experimental results for the six tasks that we consider in this chapter. 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 $^{\blacktriangle}$.

### 2.6.1 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 2.3 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,

Table 2.3: 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**$^\blacktriangle$ | **0.197**$^\blacktriangle$ | **0.159**$^\blacktriangle$ | **0.225**$^\blacktriangle$ | **0.194**$^\blacktriangle$ | **0.309**$^\blacktriangle$ | **0.281**$^\blacktriangle$ | **0.467**$^\blacktriangle$ |

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 on 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 $t$.

### 2.6.2 Changed link prediction

In real-world scenarios, only a small part of the HIN links will change over time (by being removed or added) [125], 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 2.4: 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**$^\blacktriangle$ | **0.206**$^\blacktriangle$ | **0.165**$^\blacktriangle$ | **0.230**$^\blacktriangle$ | **0.203**$^\blacktriangle$ | **0.315**$^\blacktriangle$ | **0.285**$^\blacktriangle$ | **0.473**$^\blacktriangle$ |

Table 2.4 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 that 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 becomes 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.

### 2.6.3 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 2.5.1. 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 2.5 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 2.5.3, 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 more expressive than embeddings gener-

Table 2.5: 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**▲ |

Table 2.6: 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**▲ |

ated 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.

### 2.6.4 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 2.6 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 2.7: 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**$^\blacktriangle$ | **0.211**$^\blacktriangle$ | **0.172**$^\blacktriangle$ | **0.239**$^\blacktriangle$ | **0.218**$^\blacktriangle$ | **0.327**$^\blacktriangle$ | **0.301**$^\blacktriangle$ | **0.483**$^\blacktriangle$ |

### 2.6.5 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 2.7 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 on describing relations than traditional metapath based embeddings.

### 2.6.6 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 area under the curve (AUC) score to evaluate the performance of all models. Higher AUC scores represent better performance.

Table 2.8 shows the experimental results of all models. M-DHIN outperforms other

Table 2.8: 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**$^{\blacktriangle}$ | **0.783**$^{\blacktriangle}$ | **0.755**$^{\blacktriangle}$ | **0.736**$^{\blacktriangle}$ | **0.745**$^{\blacktriangle}$ | **0.712**$^{\blacktriangle}$ | **0.792**$^{\blacktriangle}$ | **0.773**$^{\blacktriangle}$ |

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 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.

## 2.6.7 Computational costs

In this section, we present the computation cost of our model in detail. Table 2.9 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.

## 2.6.8 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

Table 2.9: Running times on each dataset of M-DHIN. Abbreviations used: LP: link prediction, CLP: changed link prediction, NC: node classification, NP: node prediction, and GR: graph reconstruction, AD: Anomaly Detection.

| | LP | CLP | NC | NP | GR | AD |
|---|---|---|---|---|---|---|
| Datasets | Time (s) | Time (s) | Time (s) | Time (s) | Time (s) | Time (s) |
| DBLP (Train) | 3484 | 2282 | 2468 | 3023 | 2643 | 4273 |
| DBLP (Test) | 1638 | 1033 | 1172 | 1436 | 1233 | 2023 |
| YELP (Train) | 2422 | 2133 | 1836 | 2237 | 1927 | 2863 |
| YELP (Test) | 1162 | 1023 | 832 | 1023 | 923 | 1245 |
| 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 |

Table 2.10: Ablation analysis of the initial embedding on the DBLP dataset. Same abbreviations used as in Table 2.9.

| | 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 |

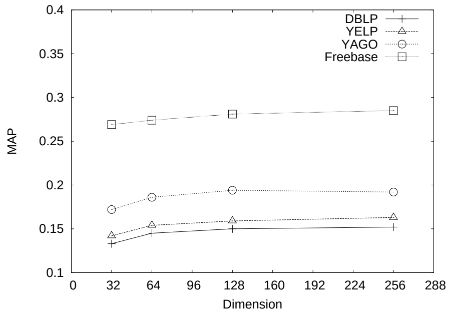
dataset as the findings on the other datasets are qualitatively similar. Table 2.10 shows the experimental results of the ablation analysis over initial embedding. We observe that M-DHIN outperforms M-DHIN\initial consistently, which verifies the effectiveness of the initial embedding.

### 2.6.9 Parameter sensitivity

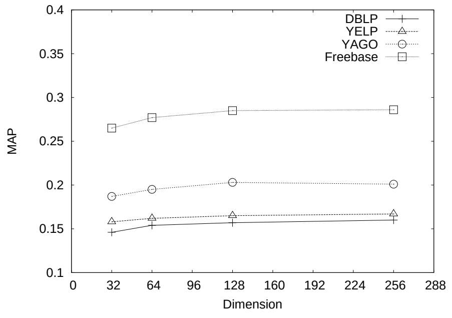
In this final subsection, we conduct a parameter sensitivity study on each task. Specifically, we choose node dimension and the ratio of negative sampling for analysis. For simplicity, we only choose one evaluation metric for each dataset in each task.

For our analysis of the sensitivity of M-DHIN to the chocie 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 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 2.5.4. Figure 2.6 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.

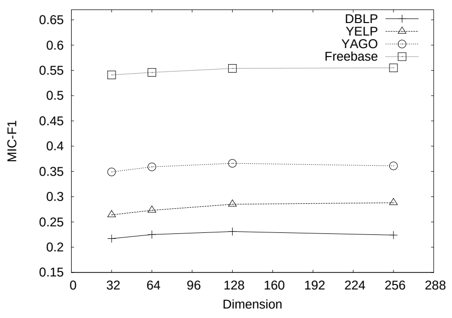
As to the analysis of M-DHIN's sensitivity to the negative ratio, for link prediction, changed link prediction and graph reconstruction tasks, we adopt R@100 for analysis; for node classification and node prediction, we choose MAC-F1 for parameter analysis;



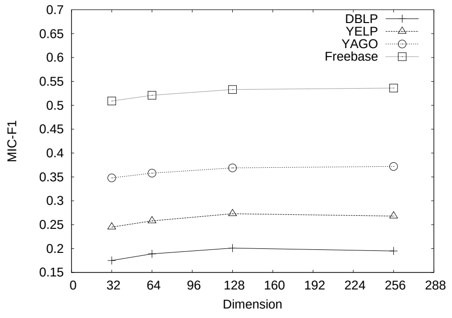
(a) Link prediction



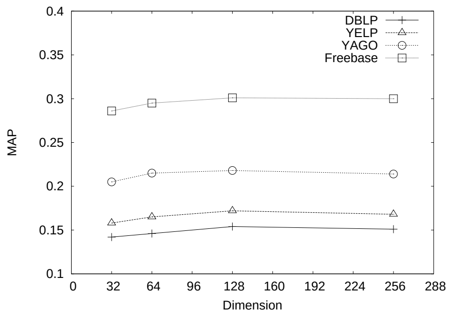
(b) Changed link prediction



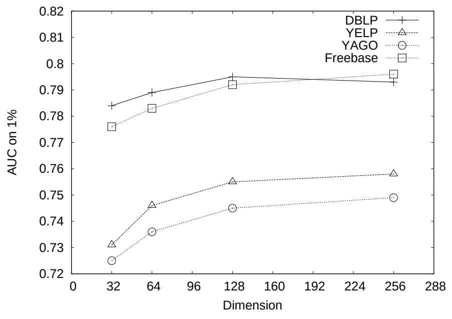
(c) Node classification



(d) Node prediction

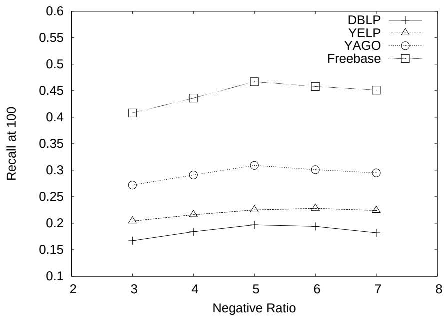


(e) Graph reconstruction

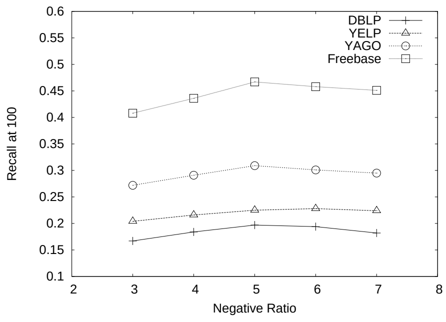


(f) Anomaly detection

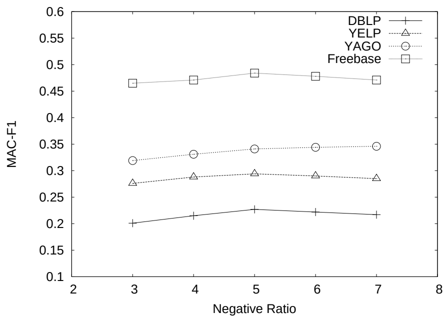
Figure 2.6: Dimension sensitivity analysis.



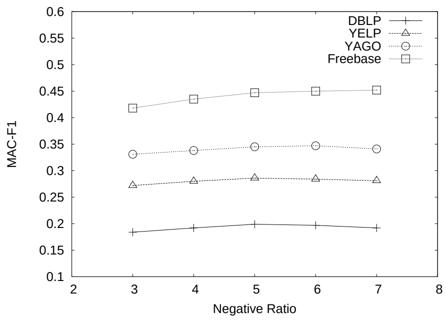
(a) Link prediction



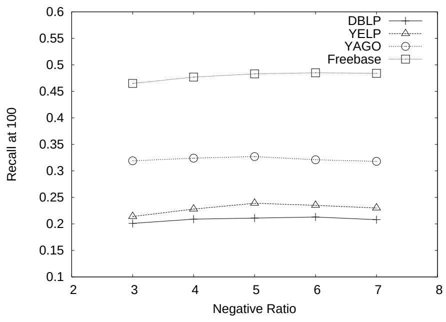
(b) Changed link prediction



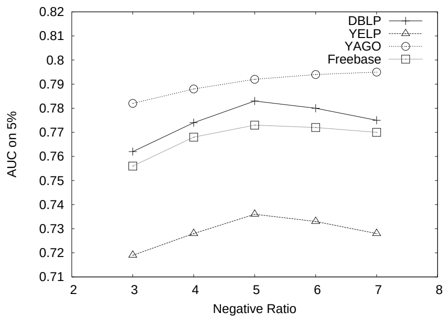
(c) Node classification



(d) Node prediction



(e) Graph reconstruction



(f) Anomaly detection

Figure 2.7: Negative ratio sensitivity analysis.

for anomaly detection we choose AUC on 5% anomalies as analysis metric. We set the other parameters to be the same as those in Section 2.5.4. Figure 2.7 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 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.

## 2.7 Conclusions

In this chapter, we have proposed a model, M-DHIN, to address **RQ1**, that is, 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 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, 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.

In this chapter, we have focused on answering **RQ1**. We have proposed a dynamic embedding model M-DHIN. It efficiently learns the representation of every time stamp of a dynamic HINs and adopts an LSTM-based deep autoencoder to predict the future network. In the next chapter, we will turn to another HINs scenario, namely how to pre-train HINs.

# 3
# Pre-training Heterogeneous Information Networks

In this chapter, we intend to answer the following research question:

**RQ2** How to pre-train heterogeneous information networks (HINs)?

Particularly, we focus on the pre-training of HINs. Previous network representation learning methods typically require sufficient task-specific labeled data to address domain-specific problems. The trained model usually cannot be transferred to out-of-domain datasets. We propose a self-supervised pre-training and fine-tuning framework, PF-HIN, to capture the features of a heterogeneous information network. Unlike traditional network representation learning models that have to train the entire model all over again for every downstream task and dataset, PF-HIN only needs to fine-tune the model and a small number of extra task-specific parameters, thus improving model efficiency and effectiveness. We design two self-supervised pre-training tasks, masked node modeling and adjacent node prediction. PF-HIN outperforms state-of-the-art alternatives on each of these tasks, on four datasets.

## 3.1 Introduction

Traditional network representation learning methods such as graph neural networks (GNNs) need to be trained in an end-to-end manner with supervised information for a task, and the model learned on one dataset cannot easily be transferred to other, out-of-domain datasets. For different datasets and tasks, traditional methods need to be re-trained all over. Additionally, in many real-life datasets, the amount of available labeled data is rarely sufficient for effective training.

Inspired by advances in pre-training frameworks in language technology [13, 43, 78], there is a trend to investigate pre-trained models for network representation learning (NRL). In particular, graph contrastive coding (GCC) [79] and GPT-GNN [46] are the most advanced solutions in this stream.[1] Nevertheless, they are mainly proposed for

This chapter was submitted as Y. Fang, X. Zhao, Y. Chen, W. Xiao, and M. de Rijke. PF-HIN: Pre-training for heterogeneous information networks. *IEEE Trans. Knowl. Data Eng.*, 35(8):8372–8385.

[1]The experiments on downstream tasks using GPT-GNN [46] were conducted on the same dataset that had been employed for pre-training.

generic NRL, meaning that they overlook the heterogeneous features of HINs; while they are generally applicable to HINs, they tend to fall short when handling HINs (as demonstrated empirically in Section 3.5 below).

We aim to overcome the shortcomings listed above, and propose

(1) to pre-train a model on large datasets using self-supervision tasks, and

(2) for a specific downstream task on a specific dataset, to use fine-tuning techniques with few task-specific parameters, so as to improve the model efficiency and effectiveness.

We refer to this two-stage (**P**re-training and **F**ine-tuning) framework for exploring the features of a **HIN** as PF-HIN.

Given a node in a HIN, we first explore the node's neighborhood by transforming it into a sequence to better capture the features of the neighboring structure. Then, a ranking of all the nodes is established based on their betweenness centrality, eigencentrality and closeness centrality. We use rank-guided heterogeneous walks to generate the sequence and group different types of nodes into so-called mini-sequences, that is, sequences of nodes of the same type [115]. Such a sampling operation can be conducted universally across different datasets, so that structural patterns and heterogeneous features can be transferred. For type information, our model is pre-trained to treat different types of nodes differently, so that in downstream tasks, different types of nodes will also be processed differently. This is the main commonality between pre-training and downstream tasks.

We design two tasks for pre-training PF-HIN. One is the *masked node modeling* (MNM) task, in which a certain percentage of nodes in the mini-sequences are masked and we need to predict those masked nodes. This operation is meant to help PF-HIN learn type-specific node features. The other task is the *adjacent node prediction* (ANP) task, which is meant to capture the relationship between nodes. Given a node $u_i$ having sequence $X_i$, our aim is to predict whether the node $u_j$ with sequence $X_j$ is an adjacent node. Other pre-training tasks like attribute masking and node type masking focus on mining features from auxiliary information of the nodes. Our proposed MNM and ANP tasks are directly applied to a graph. The MNM and ANP tasks provide more informative self-supervision for pre-training. These two tasks need to be realized by a transformer encoder, which requires the data to be sequence-like. That is the main reason why we transform the sampled nodes into a sequence. We adopt two strategies to reduce the parameters to further improve the efficiency of PF-HIN, i.e., factorized embedding parameterization and cross-layer parameter sharing. The large-scale dataset we use for pre-training is the open academic graph (OAG), containing 179 million nodes and 2 billion edges.

During fine-tuning, we choose four benchmark downstream tasks: (i) link prediction, (ii) similarity search, (iii) node classification, and (iv) node clustering. Different tasks have different fine-tuning settings [15]. We detail how to fine-tune the pre-trained model on different tasks. In link prediction and similarity search, we use node sequence pairs as input, and identify whether there is a link between two nodes and measure the similarity between two nodes, respectively. In the node classification and node clustering tasks, we use a single node sequence as input, employing a softmax layer for classification and a $k$-means algorithm for clustering, respectively.

In our experiments, which are meant to demonstrate that PF-HIN is transferable across datasets, besides a subset of OAG denoted as OAG-mini, we include three other datasets for downstream tasks: DBLP, YELP and YAGO. PF-HIN outperforms the state-of-the-art on these downstream tasks.

Our main contributions can be summarized as follows:

- We propose a pre-training and fine-tuning framework PF-HIN to mine information contained in a HIN; PF-HIN is transferable to different downstream tasks and to datasets of different domains.
- We adopt deep bi-directional transformer encoders to capture the structural features of a HIN; the architecture of PF-HIN is a variant of a GNN.
- We use type-based masked node modeling and adjacent node prediction tasks to pre-train PF-HIN; both help PF-HIN to capture heterogeneous node features and relationships between nodes.
- We show that PF-HIN outperforms the state of the art on four benchmark downstream tasks across datasets.

## 3.2 Related work

### 3.2.1 Network representation learning

Research on NRL traces back to dimensionality reduction techniques [3, 11, 85, 103], which utilize feature vectors of nodes to construct an affinity graph and then calculate eigenvectors. Graph factorization models [1] represent a graph as an adjacency matrix, and generate a low-dimensional representation via matrix factorization. Such models suffer from high computational costs and data sparsity, and cannot capture the global network structure [95].

Random walks or paths in a network are being used to help preserve the local and global structure of a network. DeepWalk [77] leverages random walks and applies the SkipGram word2vec model to learn network embeddings. node2vec [33] extends DeepWalk; it adopts a biased random walk strategy to explore the network structure. LINE [95] harnesses first- and second-order proximities to encode local and neighborhood structure information.

The aforementioned approaches are designed for homogeneous networks; other methods have been introduced for heterogeneous networks. PTE [94] defines the conditional probability of nodes of one type generated by nodes of another, and forces the conditional distribution to be close to its empirical distribution. Metapath2vec [18] has a heterogeneous SkipGram with its context window restricted to one specific type. HINE [50] uses metapath-based proximity and minimizes the distance between nodes' joint probability and empirical probabilities. HIN2Vec [28] uses Hadamard multiplication of nodes and metapaths to capture features. More introduction of heterogeneous representation learning could be found in [19, 112].

Some models employ a self-supervision technique to realize the heterogeneous representation [106], but they do not learn transferable knowledge on downstream datasets and tasks. Hence, these models cannot be directly applied in pre-training setting.

What we contribute to NRL on top of the work listed above is an efficient and effective method for representation learning for HINs based on graph neural networks (GNNs).

### 3.2.2 Graph neural networks

GNN models have shown promising results for representing networks. Efforts have been devoted to generalizing convolutional operations from visual data to graph data. Bruna et al. [7] propose a spectral graph theory-based graph convolution operation. Graph convolutional networks (GCNs) [55] adopt localized first-order approximations of spectral graph convolutions to improve scalability. There is a line of research to improve spectral GNN models [14, 41, 59, 60], but it processes the whole graph simultaneously, leading to efficiency bottlenecks. To address the problem, spatial GNN models have been proposed [29, 38, 71, 73]. GraphSAGE [38] leverages a sampling strategy to iteratively sample neighboring nodes instead of the whole graph. Gao et al. [29] utilize a sub-graph training method to reduce memory and computational cost.

GNNs fuse neighboring nodes or walks in graphs so as to learn a new node representation [57, 101, 117]. The main difference with convolution-based models is that graph attention networks introduce attention mechanisms to assign higher weights to more important nodes or walks. GAT [101] harnesses masked self-attention layers to apply different weights to different nodes in a neighborhood, to improve efficiency on graph-structured data. GIN [109] models injective multiset functions for neighbor aggregation by parameterizing universal multiset functions with neural networks.

The above GNN models have been devised for homogeneous networks as they aggregate neighboring nodes or walks regardless of their types. Targeting HINs, Het-GNN [115] first samples a fixed number of neighboring nodes of a given node and then groups these based on their types. Then, it uses a neural network architecture with two modules to aggregate the feature information of the neighboring nodes. One module is used to encode features of each type of node, the other to aggregate features of different types. HGT [47] uses node- and edge-type dependent parameters to describe heterogeneous attention over each edge; it also uses a heterogeneous mini-batch graph sampling algorithm for training.

What we contribute to GNNs is that the traditional NRL and GNN models listed above need to be re-trained all over again for different datasets and tasks, while our proposal PF-HIN only needs fine-tuning using a small number of task-specific parameters for a specific task and dataset, after pre-training via self-supervision tasks.

### 3.2.3 Graph pre-training

There exist relatively few approaches for pre-training a GNN model for downstream tasks. InfoGraph [90] maximizes the mutual information between graph-level embeddings and sub-structure embeddings. Hu et al. [44] pre-train a GNN at the level of nodes and graphs to learn local and global features, showing performance improvements on

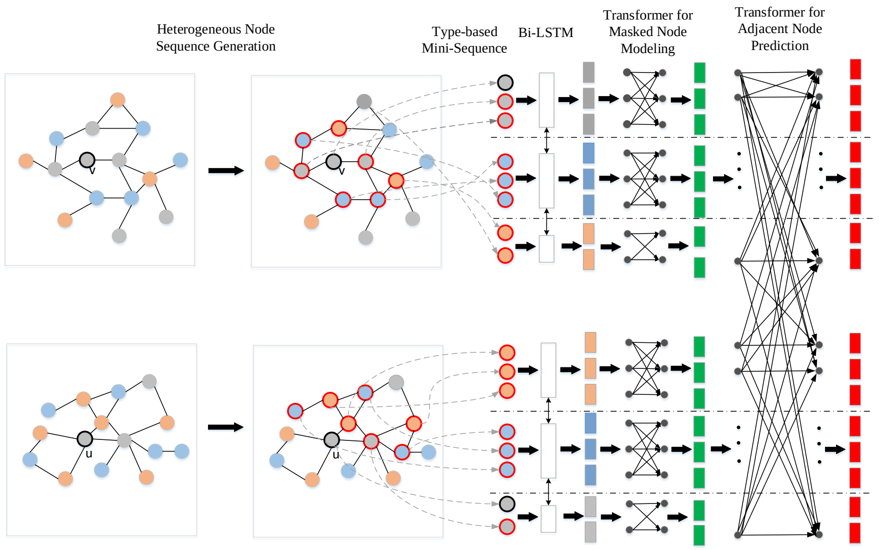

Heterogeneous Node Sequence Generation
Type-based Mini-Sequence
Bi-LSTM
Transformer for Masked Node Modeling
Transformer for Adjacent Node Prediction
v
u

Figure 3.1: The pre-training procedure that makes up PF-HIN. The generation of node sequences is detailed in Section 3.3.1. Bi-LSTM based learning of input embeddings is described in Section 3.3.2. Masked node modeling (MNM) is discussed in Section 3.3.3 and followed by adjacent node prediction (ANP).

various graph classification tasks. Our proposed model, PF-HIN, differs as we focus on node-level transfer learning and pre-train our model on a single (large-scale) graph.

Hu et al. [45] design three pre-training tasks: denoising link reconstruction, centrality score ranking, and cluster preserving. GPT-GNN [46] adopts HGT [47] as its base GNN and uses attribute generation and edge generation as pre-training tasks. Hu et al. [46] only conduct their downstream tasks on the same dataset that was used for pre-training. GCC [79] designs subgraph instance discrimination as a pre-training task and uses contrastive learning to train GNNs, with its base GNNs as GIN; then it transfers its pre-trained model to different datasets. However, it is designed for homogeneous networks, not apt to exploit heterogeneous networks. PT-HGNN [51] adopts network schemas to contrastively preserve the heterogeneous properties as a form of prior knowledge to be transferred to downstream tasks. However, network schemas are domain-specific; they may not transfer the learned knowledge to datasets in different domains.

What we contribute to graph pre-training on top of the work listed above is that we are able to not only fine-tune the proposed model across different tasks and different datasets, but can also deal with heterogeneous networks.

## 3.3 The proposed model PF-HIN

A visual presentation of the proposed model, *pre-training and fin-tuning heterogeneous information network* (PF-HIN), is given in Figure 3.1. Below, we describe the node sequence generation procedure, the input representation, followed by the pre-training and fine-tuning stages of PF-HIN.

### 3.3.1 Heterogeneous node sequence generation

We first transform the structure of a node's neighborhood to a sequence of length $k$. To measure the importance of nodes based on the structural roles in the graph, node centrality is proposed in [5]. This framework makes use of three centrality metrics,[2] i.e., (i) betweenness centrality, (ii) eigencentrality, and (iii) closeness centrality. Betweenness centrality is calculated as the fraction of shortest paths that pass through a given node. Eigencentrality measures the influence of a node on its neighbors. Closeness centrality computes the total length of the shortest paths between the given node and others. We assign learnable weights to these metrics.

To capture heterogeneous features of a node's neighbor, we adopt a so-called rank-guided heterogeneous walk to form the sequence mentioned above. The walk is with restart; it will iteratively travel from a node to its neighbors. It starts from node $v$, and it first reaches out to a node with a higher rank, which is what makes the walk rank-guided. This walk will not stop until it collects a pre-determined number of nodes. In order to assign the model with a sense of heterogeneity, we constrain the number of different types to be collected in the sequence so that every type of node can be included. We group nodes into mini-sequences, where a mini-sequence is a sequence of nodes having the same type [115]. In each mini-sequence, the nodes are sorted based on each node's rank, which serve as a kind of position information.

Importantly, unlike traditional sampling strategies like random walks, breadth first search or depth first search, our sampling strategy is able to extract important and influential neighboring nodes for each node by selecting nodes with a higher rank; this allows us to capture more representative structural information of a neighborhood. The centrality of nodes follows a power-law distribution, which means that nodes with a high degree of centrality are limited. Our sampling strategy makes sure that these more representative nodes are selected while other low-ranked nodes can also be covered. In traditional sampling strategies, the embedding of the 'hub' node could be impaired by weakly correlated neighbors. Moreover, our sampling strategy collects all types of node for each node while traditional strategies ignore the nodes' types. Nodes of the same type are grouped in mini-sequences so that further type-based analysis can be conducted to capture the heterogeneous features of a HIN. Additionally, metapaths, metagraphs and network schemas are all domain-specific; they are usually pre-defined by domain experts and may not be transferable to datasets of different domains. Our sampling strategy captures universal graph structural patterns. More empirical results with analysis are provided in Section 3.5.3.

[2]This framework is extensible in the sense that additional metric that is of particular interest to the user can be explicitly supplemented after the three metrics, as long as the combination exhibits better performance in downstream tasks. In our implementation, we stick to the basic version consisted of three metrics to demonstrate the effectiveness of the framework.

### 3.3.2 Input embeddings learned via Bi-LSTMs

After generating the sequences, we learn input embeddings of each node in the sequence using a Bi-LSTM layer. A Bi-LSTM is able to process sequence-like data and can learn deep feature interactions and obtain larger expressive capability for node representation. Given the input sequence $\{x_1, x_2, \ldots, x_n\}$, in which $x_i \in \mathbb{R}^{d\times 1}$, a Bi-LSTM is used to capture the interaction relationships between nodes. The Bi-LSTM is composed of a forward and a backward LSTM layer. The LSTM layer is defined as follows:

$$\mathbf{j}_i = \delta(\mathbf{W}_{xj}x_i + \mathbf{W}_{hj}h_{i-1} + \mathbf{W}_{cj}c_{i-1} + \mathbf{b}_j), \tag{3.1}$$

$$\mathbf{f}_i = \delta(\mathbf{W}_{xf}x_i + \mathbf{W}_{hi}h_{i-1} + \mathbf{W}_{cf}c_{i-1} + \mathbf{b}_f), \tag{3.2}$$

$$\mathbf{z}_i = \tanh(\mathbf{W}_{xc}x_i + \mathbf{W}_{hc}h_{i-1} + \mathbf{b}_c), \tag{3.3}$$

$$c_i = \mathbf{f}_i \odot c_{i-1} + \mathbf{j}_i \odot \mathbf{z}_i, \tag{3.4}$$

$$\mathbf{o}_i = \delta(\mathbf{W}_{xo}x_i + \mathbf{W}_{ho}h_{i-1} + \mathbf{W}_{co}c_i + \mathbf{b}_o), \tag{3.5}$$

$$h_i = \mathbf{o}_i\tanh(c_i), \tag{3.6}$$

where $h_i \in \mathbb{R}^{d/2} \times 1$ is the output hidden state of node $i$, $\odot$ represents the element-wise product, $\mathbf{W} \in \mathbb{R}^{(d/2)\times(d/2)}$ and $b \in \mathbb{R}^{d/2\times 1}$ are learnable parameters, which denote weight and bias, respectively; $\mathbf{j}_i$, $\mathbf{f}_i$, $\mathbf{o}_i$ are the input gate vector, forget gate vector and output vector, respectively. We concatenate the hidden state of the forward and backward LSTM layers to form the final hidden state of the Bi-LSTM layer. For each type of node, we adopt different Bi-LSTMs so as to extract type-specific features.

### 3.3.3 Masked node modeling

After generating the input embeddings via a Bi-LSTM, we adopt masked node modeling (MNM) as our pre-training task. We randomly mask a percentage of the input nodes and then predict those masked nodes. We conduct this task on the type-based mini-sequences generated by the aforementioned rank guided heterogeneous walk. For each group of nodes with the same type, we randomly mask nodes in the mini-sequence. Given the mini-sequence of type $t$, denoted as $\{x_1^t, x_2^t, \ldots, x_n^t\}$, we randomly choose $15\%$ of the nodes to be replaced. And for a chosen node $x_i^t$, we replace its token with the actual [MASK] token with 80% probability, another random node token with $10\%$ probability and the unchanged $x_i^t$ with 10% probability. The masked sequence is fed into the bi-directional transformer encoders. The embeddings generated via the Bi-LSTM are used as token embeddings, while the rank information is transferred as position embeddings. After the transformer module, the final hidden state $h_i^{t_L}$ corresponding to the [MASK] token is fed to a feedforward layer. The output is used to predict the target node via a softmax classification layer:

$$z_i^t = \text{Feedforward}(h_i^{t_L}), \tag{3.7}$$

$$\mathbf{p}_i^t = \text{softmax}(\mathbf{W}^{\text{MNM}} z_i^t), \tag{3.8}$$

where $z_i^t$ is the output of the feedforward layer, $\mathbf{W}^{\text{MNM}} \in V^t \times d$ is the classification weight shared with the input node embedding matrix, $V^t$ is the number of nodes in the $t$-type mini-sequence, $d$ is the dimension of the hidden state size, $\mathbf{p}_i^t$ is the predicted distribution of $x_i^t$ over all nodes.

For training, we use the cross-entropy between the one-hot label $\mathbf{y}_i^t$ and the prediction $\mathbf{p}_i^t$:

$$\mathcal{L}_{\mathrm{MNM}}^t = -\sum_m y_m^t \log p_m^t, \tag{3.9}$$

where $y_m^t$ and $p_m^t$ are the $m$-th components of $\mathbf{y}_i^t$ and $\mathbf{p}_i^t$, respectively. We adopt a smoothing strategy by setting $y_m^t = \epsilon$ for the target node and $y_m^t = \frac{1-\epsilon}{V^t-1}$ for each of the other nodes. By doing so, we loosen the restriction that a one-hot label corresponds to only one answer.

### 3.3.4 Adjacent node prediction

Aside from a masked node modeling module, we design another pre-training task, i.e., adjacent node prediction (ANP), to capture the relationship between nodes. Note that the ANP and MNM tasks are conducted simultaneously in practice. Unlike the MNM task, which operates on type-based mini-sequences, we perform the ANP task on full sequences, and we compare two full sequences to see whether their starting nodes are adjacent or not. The reason that we do not perform the ANP task on type-based mini-sequences is that given $k$ types of nodes, there will be $k(k-1)/2$ mini-sequence pairs to be analyzed, which is very time-consuming.

In our setting, for node $v$ with sequence $X_v$ and node $u$ with sequence $X_u$, 50% of the time we choose $u$ to be the actual adjacent node of $v$ (labeled as IsAdjacent), and 50% of the time we randomly choose $u$ from the corpus (labeled as NotAdjacent) to save training time. More fake nodes could also be included. Given the classification layer weights $\mathrm{W}^{\mathrm{ANP}}$, the scoring function $s_\tau$ of whether the node pair is adjacent is shown as follows:

$$s_\tau = \mathrm{sigmoid}(C\mathrm{W}^{\mathrm{ANP}^T}), \tag{3.10}$$

where $s_\tau \in \mathbb{R}^2$ is a binary vector with $s_{\tau 0}, s_{\tau 1} \in [0,1]$ and $s_{\tau 0} + s_{\tau 1} = 1$, $C \in \mathbb{R}^H$ denotes the hidden vector of classification label used in a transformer architecture [15]. Considering the positive adjacent node pair $\mathbb{S}^+$ and a negative adjacent node pair $\mathbb{S}^-$, we calculate a cross-entropy loss as follows:

$$\mathcal{L}_{\mathrm{ANP}} = -\sum_{\tau \in \mathbb{S}^+ \cup \mathbb{S}^-} (y_\tau \mathrm{log}(s_{\tau 0})) + (1 - y_\tau)\mathrm{log}(s_{\tau 1})), \tag{3.11}$$

where $y_\tau$ is the label (positive or negative) of that node pair.

During the whole pre-training pipeline, we minimize the following loss:

$$\mathcal{L} = \mathcal{L}_{\mathrm{MNM}} + \mathcal{L}_{\mathrm{ANP}}. \tag{3.12}$$

Through the MNM task, the model is able to predict a missing node by considering the neighborhood and context of the missing node, thus exploring the node-wise network structure. Through ANP task, the model can predict whether two nodes are connected by considering the relationships of them and their context, thus exploring the edge-wise network structure. In other words, PF-HIN adopt structure-level pre-training tasks. However, previous pre-training tasks like attribute masking and node type masking only make use of the auxiliary information of a node. Therefore, PF-HIN provides more informative self-supervision for pre-training.

**Algorithm 4:** The pre-training procedure of PF-HIN.

```
Input :
Input HIN $G$;
Output:
Optimized model parameters $\Theta$ (for downstream tasks);
1  Generates the node sequences via rank guided heterogeneous walk;
2  for each pair of sampled sequences do
3      Apply Bi-LSTM on type-based mini-sequences to learn the input
         embeddings of each node in the sequence;
4      for each sequence do
5          Mask nodes in the type-based mini-sequences;
6          Feed the mini-sequences into transformer layers;
7          Calculate the masked node modeling loss by Eq. (3.9);
8      end
9      Feed the two sequenes into transformer layers;
10     Calculate the ajacent node prediction loss by Eq. (3.11);
11     Update the parameters $\Theta$ by Adam.
12 end
13 return Optimized pre-trained model parameters $\Theta^*$.
```

## 3.3.5 Transformer architecture

Our two pre-training tasks share the same transformer architecture. To increase the training speed of our model, we adopt two parameter reduction techniques to lower memory requirements, inspired by the ALBERT architecture [56]. Instead of setting the node embedding size $Q$ to be equal to the hidden layer size $H$ like BERT [15] does, we make more efficient use of the total number of model parameters, dictating that $H \gg Q$. We adopt factorized embedding parameterization, which decomposes the parameters into two smaller matrices. Rather than mapping the one-hot vectors directly to a hidden space with size $H$, we first map them to a low-dimensional embedding space with size $Q$, and then map it to the hidden space. Additionally, we adopt cross-layer parameter sharing to further boost efficiency. Traditional sharing mechanisms either only share the feed forward network parameters across layers or only the attention parameters. We share *all* parameters across layers.

We denote the number of transformer layers as $L$, and the number of self-attention heads as $A$. For our parameter settings we follow the configuration of ALBERT [56], where $L$ is set to 12, $H$ to 768, $A$ to 12, $Q$ to 128, and the total number of parameters is equal to 12M. For the procedure of our pre-training task, see Algorithm 4.

## 3.3.6 Fine-tuning PF-HIN

The self-attention mechanism in the transformer allows PF-HIN to model many downstream tasks. Fine-tuning can be realized by simply swapping out the proper inputs and outputs, regardless of the single node sequence or sequence pairs used. For each downstream task, the task-specific inputs and outputs are simply plugged into PF-HIN and all parameters are fine-tuned end-to-end. Here, we introduce four tasks:

(1) link prediction,

(2) similarity search,

(3) node classification, and

(4) node clustering as downstream tasks.

Specifically, in link prediction, we predict whether there is a link between two nodes, and the inputs are the node sequence pairs. To generate output, we feed the classification label into the sigmoid layer, so as to predict the existence of a link between two nodes. The only new parameters are the classification layer weights $W \in \mathbb{R}^{2\times H}$, where $H$ is the size of hidden state.

In similarity search, in order to measure the similarity between two nodes, we use the node sequence pairs as input. We leverage the token-level output representations to compute the similarity score of two nodes.

In node classification, we only use a single node sequence as input and generate the classification label via a softmax layer. To calculate the classification loss, we only need to add classification layer weights $W \in \mathbb{R}^{K\times H}$ as new parameters, where $K$ is the number of classification labels and $H$ is the size of hidden state.

In node clustering, we also use a one node sequence as input and then put the token-level output embeddings to a clustering model, so as to cluster the data.

Experimental details for these downstream tasks are introduced in Section 3.4 below.

## 3.4 Experimental setup

We detail our datasets, baseline models, and parameter settings.

### 3.4.1 Datasets

We adopt the open academic graph OAG[3] as our pre-training dataset, which is a heterogeneous academic dataset. It contains over 178 million nodes and 2.223 billion edges with five types of node: (i) author, (ii) paper, (iii) venue, (iv) field, and (v) institute.

For downstream tasks, we transfer our pre-trained model to four datasets: (i) OAG-mini, (ii) DBLP, (iii) YELP, and (iv) YAGO. OAG-mini is a subset extracted from OAG; the authors are split into four areas: machine learning, data mining, database, and information retrieval. DBLP[4] is also an academic dataset with four types of node: (i) author, (ii) paper, (iii) venue, and (iv) topic; the authors are split into the same areas as those in OAG-mini. YELP[5] is a social media dataset, with restaurants reviews and four types of node: (i) review, (ii) customer, (iii) restaurant, and (iv) food-related keywords. The

[3]https://www.openacademic.ai/oag/

[4]http://dblp.uni-trier.de

[5]https://www.yelp.com/dataset_challenge

Table 3.1: Dataset statistics.

| Dataset | #nodes | #edges | #node types |
|---|---:|---:|:---:|
| OAG | 178,663,927 | 2,236,196,802 | 5 |
| OAG-mini | 473,324 | 2,343,578 | 5 |
| DBLP | 301,273 | 1,382,587 | 4 |
| YELP | 201,374 | 872,432 | 4 |
| YAGO | 52,384 | 143,173 | 5 |

restaurants are separated into (i) Chinese food, (ii) fast food, and (iii) sushi bar. YAGO[6] is a knowledge base and we extracted a subset of it containing movie information, having five types of node: (i) movie, (ii) actor, (iii) director, (iv) composer, and (v) producer. The movies are split into five types: (i) action, (ii) adventure, (iii) sci-fi, (iv) crime, and (v) horror.

The dataset statistics are shown in Table 3.1. In this chapter, we aim to address the issue that the data is usually scarce with only a few labels given, which means that the fine-tuned data is limited. So in practice, 10% of the training data are fine-tuned with label.

### 3.4.2 Algorithms used for comparison

We first choose network embedding methods to directly train the downstream datasets for specific tasks as baselines: DeepWalk [77], LINE [95] and node2vec [33]; they were originally applied to homogeneous information networks. DeepWalk and node2vec leverage random walks, while node2vec uses a biased walk strategy to capture the network structure. LINE uses the local and neighborhood structural information via first-order and second-order proximities.

We include three state-of-the-art algorithms devised for HINs: metapath2vec [18], HINE [50], HIN2Vec [28]. They are all based on metapaths, but differ in the way they use metapath features: metapath2vec adopts heterogeneous SkipGrams, HINE proposes a metapath-based notion of proximity, and HIN2Vec utilizes the Hadamard multiplication of nodes and metapaths.

We also include other GNN models, i.e., GCN [55], GAT [101], GraphSAGE [38] and GIN [109], which were originally devised for homogeneous information networks. GCN and GraphSAGE are based on convolutional operations, while GCN requires the Laplacian of the full graph, and GraphSAGE only needs a node's local neighborhood. GAT employs an attention mechanism to capture the correlation between central node and neighboring nodes. GIN uses parameterizing universal multiset functions with neural networks to model injective multiset functions for neighbor aggregation.

We also select HetGNN [115], HGT [47] as models for comparison; both have been devised for HIN embeddings. HetGNN samples heterogeneous neighbors, grouping them based on their node types, and then aggregates feature information of the sampled neighboring nodes. HGT has node- and edge-type dependent parameters to characterize heterogeneous attention over each edge.

[6]`https://old.datahub.io/dataset/yago`

The above network embedding methods are all directly applied on the downstream datatasks.

Aside from those network embedding methods, for a fair comparison, we also choose GPT-GNN [46], GCC [79] and PT-HGNN [51] to run the entire pre-training and fine-tuning pipeline. GPT-GNN utilizes attribute generation and edge generation tasks to pre-train GNN, with HGT as its base GNN. GCC adopts subgraph instance discrimination as a pre-training task, taking GIN as its base GNN. PT-HGNN proposes to adopt node-level and schema-level contrastive learning as the pre-training task.

### 3.4.3 Parameters

For pre-training, we set the generated sequence length $k$ to 20. The dimension of the node embedding is set to 128 and the size of hidden state is set to 768. On transformer layers, we use $0.1$ as the dropout probability. The Adam learning rate is initiated as $0.001$ with a linear decay. We use 256 sequences to form a batch and the training epoch is set to 20. The training loss is the sum of the mean masked node modeling likelihood and the mean adjacent node prediction likelihood.

In fine-tuning, most parameters remain the same as in pre-training, except the learning rate, batch size and number of epochs. We use grid search to determine the best configuration. The learning rate is chosen from $\{0.01, 0.02, 0.025, 0.05\}$. The training epoch is chosen from $\{2, 3, 4, 5\}$. The batch size is chosen from $\{16, 32, 64\}$. The optimal parameters are task-specific. For the other models, we adopt the best configurations reported in the source publications.

We report on statistical significance with a paired two-tailed t-test and we mark a significant improvement of PF-HIN over GPT-GNN for $p < 0.05$ with $^{\blacktriangle}$.

## 3.5 Results and analysis

We present the results of fine-tuning PF-HIN on four downstream tasks:

(1) link prediction,

(2) similarity search,

(3) node classification, and

(4) node clustering.

We analyze the computational costs, conduct an ablation analysis, and study the parameter sensitivity.

### 3.5.1 Downstream tasks

#### Link prediction

This task is to predict which links will occur in the future. Unlike previous work [33] that randomly samples a certain percentage of links as the training dataset and uses the remaining links as the evaluation dataset, we adopt a sequential split of training and test

Table 3.2: Results for the link prediction task.

| | OAG-mini | | DBLP | | YELP | | YAGO | |
|---|---|---|---|---|---|---|---|---|
| Model | AUC | F1 | AUC | F1 | AUC | F1 | AUC | F1 |
| DeepWalk | 0.378 | 0.266 | 0.583 | 0.351 | 0.602 | 0.467 | 0.735 | 0.525 |
| LINE | 0.382 | 0.257 | 0.274 | 0.357 | 0.605 | 0.463 | 0.739 | 0.531 |
| node2vec | 0.392 | 0.273 | 0.584 | 0.355 | 0.609 | 0.471 | 0.742 | 0.534 |
| metapath2vec | 0.412 | 0.282 | 0.604 | 0.367 | 0.618 | 0.473 | 0.744 | 0.541 |
| HINE | 0.423 | 0.288 | 0.607 | 0.369 | 0.621 | 0.482 | 0.763 | 0.548 |
| HIN2Vec | 0.426 | 0.291 | 0.611 | 0.376 | 0.625 | 0.493 | 0.768 | 0.578 |
| GCN | 0.437 | 0.294 | 0.623 | 0.392 | 0.638 | 0.516 | 0.779 | 0.583 |
| GraphSage | 0.445 | 0.293 | 0.627 | 0.395 | 0.641 | 0.525 | 0.783 | 0.592 |
| GAT | 0.451 | 0.294 | 0.631 | 0.392 | 0.644 | 0.537 | 0.781 | 0.596 |
| GIN | 0.456 | 0.299 | 0.636 | 0.394 | 0.647 | 0.539 | 0.785 | 0.598 |
| HetGNN | 0.467 | 0.317 | 0.642 | 0.402 | 0.663 | 0.544 | 0.793 | 0.602 |
| HGT | 0.473 | 0.321 | 0.648 | 0.407 | 0.672 | 0.549 | 0.799 | 0.605 |
| GPT-GNN | 0.513 | 0.371 | 0.678 | 0.423 | 0.679 | 0.558 | 0.811 | 0.617 |
| GCC | 0.507 | 0.352 | 0.669 | 0.417 | 0.668 | 0.552 | 0.805 | 0.609 |
| PT-HGNN | **0.523** | **0.388** | **0.697** | 0.436 | 0.677 | 0.551 | 0.802 | 0.603 |
| PF-HIN | 0.519 | 0.383 | 0.692 | **0.442**▲ | **0.691**▲ | **0.565**▲ | **0.822**▲ | **0.624**▲ |

data. We first train a binary logistic classifier on the graph of training data, and then use the test dataset with the same number of random negative (non-existent) links to evaluate the trained classifier. We only consider the new links in the training dataset and remove duplicate links from the evaluation. We adopt AUC and F1 score as evaluation metrics.

We present the link prediction results in Table 3.2, with the highest results set in bold. Scores increase as the dataset size decreases. Traditional homogeneous models (DeepWalk, LINE, node2vec) perform worse than traditional heterogeneous metapath based models (metapath2vec, HINE, HIN2Vec); metapaths capture the network structure better than random walks. However, homogeneous GNN models (GCN, GraphSAGE, GAT, GIN) achieve even better results than traditional heterogeneous methods. Deep neural networks explore the entire network more effectively, generating better representations for link prediction. HetGNN and HGT outperform the homogeneous GNN models, since they take the node types into consideration. GPT-GNN, GCC and PT-HGNN outperform all of the above methods including their base GNN (HGT and GIN). Adopting pre-training tasks can boost the downstream task performance.

PF-HIN outperforms GCC. This is because our pre-training on (type-based) mini-sequences helps to explore the HIN, while GIN is designed for homogeneous information. PF-HIN outperforms GPT-GNN due to our choice of pre-training task, as the ANP task is more effective on predicting links between nodes than the edge generation task used in GPT-GNN. By deciding whether two nodes are adjacent, ANP can tell if a link connects them. PT-HGNN performs slightly better than PF-HIN on OAG-mini and DBLP datsets. This is because that these two datasets share the same domain of

Table 3.3: Results for the similarity search task.

| | OAG-mini | DBLP | YELP | YAGO |
|---|---|---|---|---|
| Model | AUC | AUC | AUC | AUC |
| DeepWalk | 0.478 | 0.511 | 0.553 | 0.656 |
| LINE | 0.482 | 0.506 | 0.558 | 0.661 |
| node2vec | 0.483 | 0.513 | 0.559 | 0.653 |
| metapath2vec | 0.494 | 0.545 | 0.578 | 0.673 |
| HINE | 0.506 | 0.554 | 0.583 | 0.683 |
| HIN2Vec | 0.512 | 0.556 | 0.587 | 0.687 |
| GCN | 0.509 | 0.553 | 0.581 | 0.682 |
| GraphSage | 0.513 | 0.557 | 0.586 | 0.689 |
| GAT | 0.510 | 0.555 | 0.584 | 0.691 |
| GIN | 0.514 | 0.559 | 0.587 | 0.690 |
| HetGNN | 0.527 | 0.563 | 0.592 | 0.694 |
| HGT | 0.532 | 0.568 | 0.591 | 0.697 |
| GPT-GNN | 0.563 | 0.596 | 0.603 | 0.708 |
| GCC | 0.554 | 0.587 | 0.597 | 0.702 |
| PT-HGNN | **0.578** | 0.609 | 0.592 | 0.699 |
| PF-HIN | 0.574 | **0.612**$^{\blacktriangle}$ | **0.613**$^{\blacktriangle}$ | **0.719**$^{\blacktriangle}$ |

pre-training datasets OAG, and PT-HGNN takes advantage of network-schema which can capture higher-order structure of HIN. However, it performs much worse than PF-HIN on YELP and YAGO, which proves that network schema is not transferable on datasets of different domains.

### Similarity search

In this task, we aim to find nodes that are similar to a given node. To evaluate the similarity between two nodes, we calculate the cosine similarity based on the node representations. It is hard to rank all pairs of nodes explicitly, so we provide an estimation based on the grouping label $g(\cdot)$, in which similar nodes are gathered in one group. Given a node $u$, if we rank other nodes based on the similarity score, intuitively, nodes from the same group (similar ones) should be at the top of the ranked list while dissimilar ones should be ranked at the bottom.

Table 3.3 displays the results for the similarity search task. The highest scores are set in bold. The traditional heterogeneous models and homogeneous GNN models achieve comparable results; metapath based mechanisms and deep neural networks can generate expressive node embeddings for similarity search. HetGNN and HGT outperform the above methods, which shows the power of combining GNN and type features. PF-HIN performs well in all cases especially on YELP and YAGO, illustrating the effectiveness of our pre-training model on learning heterogeneous node representations for similarity search.

Table 3.4: Results for the multi-label node classification task.

| | OAG-mini | | DBLP | | YELP | | YAGO | |
|---|---|---|---|---|---|---|---|---|
| Model | MIC | MAC | MIC | MAC | MIC | MAC | MIC | MAC |
| DeepWalk | 0.175 | 0.173 | 0.193 | 0.191 | 0.163 | 0.145 | 0.328 | 0.265 |
| LINE | 0.177 | 0.172 | 0.184 | 0.179 | 0.274 | 0.276 | 0.366 | 0.320 |
| node2vec | 0.180 | 0.178 | 0.201 | 0.198 | 0.194 | 0.151 | 0.332 | 0.280 |
| metapath2vec | 0.195 | 0.194 | 0.209 | 0.207 | 0.264 | 0.269 | 0.370 | 0.332 |
| HINE | 0.198 | 0.192 | 0.234 | 0.230 | 0.276 | 0.284 | 0.401 | 0.363 |
| HIN2Vec | 0.204 | 0.201 | 0.246 | 0.241 | 0.291 | 0.306 | 0.428 | 0.394 |
| GCN | 0.209 | 0.205 | 0.257 | 0.256 | 0.302 | 0.311 | 0.459 | 0.447 |
| GraphSage | 0.211 | 0.207 | 0.267 | 0.269 | 0.305 | 0.318 | 0.464 | 0.456 |
| GAT | 0.213 | 0.211 | 0.271 | 0.273 | 0.303 | 0.315 | 0.469 | 0.462 |
| GIN | 0.218 | 0.223 | 0.274 | 0.277 | 0.308 | 0.321 | 0.474 | 0.466 |
| HetGNN | 0.234 | 0.237 | 0.285 | 0.282 | 0.309 | 0.324 | 0.478 | 0.471 |
| HGT | 0.239 | 0.241 | 0.283 | 0.278 | 0.313 | 0.327 | 0.484 | 0.483 |
| GPT-GNN | 0.276 | 0.274 | 0.304 | 0.299 | 0.322 | 0.339 | 0.496 | 0.493 |
| GCC | 0.266 | 0.264 | 0.295 | 0.291 | 0.315 | 0.332 | 0.489 | 0.486 |
| PT-HGNN | **0.297** | 0.287 | 0.316 | **0.315** | 0.312 | 0.330 | 0.486 | 0.484 |
| PF-HIN | 0.294 | **0.292**$^\blacktriangle$ | **0.318**$^\blacktriangle$ | 0.311 | **0.330**$^\blacktriangle$ | **0.354**$^\blacktriangle$ | **0.516**$^\blacktriangle$ | **0.509**$^\blacktriangle$ |

## Node classification

Next, we report on the results for the multi-label node classification task. The size (ratio) of the training data is set to 30% and the remaining nodes are used for testing. We adopt micro-F1 (MIC) and macro-F1 (MAC) as our evaluation metrics.

Table 3.4 provides the results on the node classification task; the highest scores are set in bold. GNN based models perform relatively well, showing the benefits of using deep neural networks for exploring features of the network data for classification. PF-HIN achieves high scores thanks to our fine-tuning framework, which aggregates the full sequence information for node classification.

## Node clustering

Finally, we report on the outcomes for the node clustering task. We feed the generated node embeddings of each model into a clustering model. Here, we choose a k-means algorithm to cluster the data. The size (ratio) of the training data is set to 30% and the remaining nodes are used for testing. We use normalized mutual information (NMI) and adjusted rand index (ARI) as evaluation metrics.

Table 3.5 shows the performance for the node clustering task, with the highest scores set in bold. Despite the strong ability of homogeneous GNN models to capture structural information of a network, they perform slightly worse than traditional heterogeneous models. PF-HIN performs steadily well on four datasets, proving that PF-HIN is able to generate effective node embeddings for node clustering.

Table 3.5: Results for the node clustering task.

| | OAG-mini | | DBLP | | YELP | | YAGO | |
|---|---|---|---|---|---|---|---|---|
| Model | NMI | ARI | NMI | ARI | NMI | ARI | NMI | ARI |
| DeepWalk | 0.601 | 0.613 | 0.672 | 0.686 | 0.713 | 0.744 | 0.856 | 0.886 |
| LINE | 0.598 | 0.609 | 0.678 | 0.693 | 0.705 | 0.739 | 0.861 | 0.894 |
| node2vec | 0.604 | 0.616 | 0.673 | 0.689 | 0.719 | 0.748 | 0.867 | 0.899 |
| metapath2vec | 0.628 | 0.632 | 0.711 | 0.738 | 0.748 | 0.785 | 0.896 | 0.917 |
| HINE | 0.634 | 0.638 | 0.718 | 0.741 | 0.753 | 0.786 | 0.899 | 0.921 |
| HIN2Vec | 0.637 | 0.640 | 0.721 | 0.744 | 0.749 | 0.789 | 0.902 | 0.923 |
| GCN | 0.616 | 0.641 | 0.701 | 0.719 | 0.744 | 0.775 | 0.881 | 0.907 |
| GraphSage | 0.619 | 0.643 | 0.705 | 0.722 | 0.746 | 0.778 | 0.885 | 0.911 |
| GAT | 0.622 | 0.645 | 0.709 | 0.728 | 0.748 | 0.782 | 0.893 | 0.915 |
| GIN | 0.625 | 0.650 | 0.711 | 0.735 | 0.752 | 0.785 | 0.898 | 0.919 |
| HetGNN | 0.644 | 0.678 | 0.729 | 0.748 | 0.759 | 0.792 | 0.904 | 0.926 |
| HGT | 0.648 | 0.682 | 0.734 | 0.753 | 0.763 | 0.794 | 0.909 | 0.931 |
| GPT-GNN | 0.673 | 0.712 | 0.756 | 0.761 | 0.769 | 0.808 | 0.919 | 0.937 |
| GCC | 0.652 | 0.707 | 0.741 | 0.756 | 0.765 | 0.799 | 0.913 | 0.933 |
| PT-HGNN | **0.694** | **0.731** | **0.773** | **0.776** | 0.766 | 0.802 | 0.915 | 0.935 |
| PF-HIN | 0.691 | 0.728 | 0.770 | 0.773 | **0.781**$^{\blacktriangle}$ | **0.816**$^{\blacktriangle}$ | **0.927**$^{\blacktriangle}$ | **0.946**$^{\blacktriangle}$ |

## 3.5.2 Computational costs

To evaluate the efficiency of our fine-tuning framework compared to other models, we conduct an analysis of the computational costs. Specifically, we analyze the running time of each model on each task, using the early stopping mechanism. Due to space limitations we only report the results on the DBLP dataset; the results for the remaining datasets are qualitatively similar. See Table 3.6. We use standard hardware (Intel (R) Core (TM) i7-10700K CPU + GTX-2080 GPU); the time reported is wall-clock time, averaged over 10 runs.

PF-HIN's running time is longer than of the three traditional homogeneous models DeepWalk, LINE and node2vec, which are based on random walks; it is not as high as any of the other models. GNN based models like GCN, GraphSAGE, GAT, GIN and HetGNN have a higher running time than all other models, since the complexity of traditional deep neural networks is much higher than other algorithms.

GPT-GNN, GCC, PT-HGNN and PF-HIN have relatively short running times; this is because pre-trained parameters help the loss function converge much faster.

## 3.5.3 Ablation analysis

We analyze the effect of the pre-training tasks, the bi-directional transformer encoders, the components of the input representation, the rank-guided heterogeneous walk sampling strategy, the centrality metric and the fine-tuning setting.

Table 3.6: Running times on the DBLP dataset. Abbreviations used: LP: link prediction, SS: similarity search, MC: multi-label node classification, and NC: node clustering.

| | LP | SS | MC | NC |
|---|---|---|---|---|
| Model | Time (h) | Time (h) | Time (h) | Time (h) |
| DeepWalk | 1.09 | 1.15 | 0.74 | 0.53 |
| LINE | 1.19 | 1.31 | 0.76 | 0.62 |
| node2vec | 1.46 | 1.54 | 0.97 | 0.74 |
| metapath2vec | 1.68 | 1.90 | 1.35 | 1.05 |
| HINE | 1.84 | 2.11 | 1.38 | 1.08 |
| HIN2Vec | 2.02 | 2.53 | 1.56 | 1.26 |
| GCN | 3.45 | 4.14 | 2.66 | 2.08 |
| GraphSage | 2.75 | 3.25 | 2.17 | 1.46 |
| GAT | 2.55 | 2.94 | 1.73 | 1.55 |
| GIN | 3.59 | 4.54 | 3.21 | 2.33 |
| HGT | 3.11 | 3.80 | 2.53 | 2.19 |
| HetGNN | 3.15 | 3.68 | 2.51 | 1.80 |
| GPT-GNN | 1.64 | 1.92 | 1.35 | 1.03 |
| GCC | 1.55 | 2.28 | 1.56 | 1.13 |
| PT-HGNN | 1.53 | 1.78 | 1.22 | 0.93 |
| PF-HIN | 1.52 | 1.71 | 1.18 | 0.82 |

### Effect of pre-training task

To evaluate the effect of the pre-training tasks, we introduce two variants of PF-HIN, i.e., PF-HIN\MNM and PF-HIN\ANP. PF-HIN\MNM is like PF-HIN but excludes pre-training on the masked node modeling task, PF-HIN\ANP is like PF-HIN but excludes pre-training on the adjacent node prediction task. Due to space limitations, we only report the experimental outcomes on the DBLP dataset; the findings on the other datasets are qualitatively similar. Table 3.7 shows the experimental results of the ablation analysis over pre-training tasks. For the link prediction task, ANP is more important than MNM since predicting if two nodes are adjacent could also tell if they are connected by a link. In similarity search, those two tasks have a comparable effect. For the node classification and node clustering tasks, MNM plays a more important role, and this is because MNM directly models node features and hence PF-HIN is better able to explore them.

### Effect of the bi-directional transformer encoder

Our bi-directional transformer encoder is a variant of GNN applied to HINs, aggregating neighborhood information. For our analysis, we replace the transformer encoders with a CNN, a bi-directional LSTM, and an attention mechanism. Specifically, the model using the CNN encoder is denoted as PF-HIN(CNN), the model using bi-directional LSTM as PF-HIN(LSTM), and the model using an attention mechanism as PF-HIN(attention). Again, we report on experiments on the DBLP dataset only. Table 3.8 presents the

Table 3.7: Ablation analysis of the pre-training tasks on the DBLP dataset. Same abbreviations used as in Table 3.6.

| | LP | | SS | MC | | NC | |
|---|---|---|---|---|---|---|---|
| Model | AUC | F1 | AUC | MIC | MAC | NMI | ARI |
| PF-HIN | **0.692** | **0.442** | **0.612** | **0.318** | **0.311** | **0.770** | **0.773** |
| PF-HIN\ANP | 0.421 | 0.221 | 0.356 | 0.217 | 0.257 | 0.548 | 0.629 |
| PF-HIN\MNM | 0.498 | 0.283 | 0.332 | 0.172 | 0.174 | 0.422 | 0.438 |

Table 3.8: Ablation analysis of the encoder on the DBLP dataset. Same abbreviations used as in Table 3.6.

| | LP | | SS | MC | | NC | |
|---|---|---|---|---|---|---|---|
| Model | AUC | F1 | AUC | MIC | MAC | NMI | ARI |
| PF-HIN | **0.692** | **0.442** | **0.612** | **0.318** | **0.311** | **0.770** | **0.773** |
| PF-HIN(CNN) | 0.643 | 0.418 | 0.578 | 0.296 | 0.287 | 0.743 | 0.747 |
| PF-HIN(LSTM) | 0.657 | 0.421 | 0.567 | 0.293 | 0.279 | 0.754 | 0.753 |
| PF-HIN(attention) | 0.629 | 0.411 | 0.572 | 0.299 | 0.291 | 0.748 | 0.757 |

experimental results of different encoders. The CNN, LSTM and attention mechanism based models achieve comparable results on the four tasks. PF-HIN consistently outperforms all three models, which shows the importance of our bi-directional transformer encoder for mining the information contained in a HIN.

### Effect of pre-training strategy

In PF-HIN, the MNM task is pre-trained on (type-based) mini-sequences, while the ANP task is pre-trained on two full sequences generated via two starting nodes. Here we analyze the effect of this strategy. We consider three variants of PF-HIN. (i) The first is to pre-train the two tasks on two full sequences without considering the heterogeneous features of the network, denoted as PF-HIN(full-full). (ii) In the second, we try to assign the ANP task with a sense of heterogeneous features. As explained in Section 3.3.4, it is too time-consuming to pre-train all mini-sequence pairs for the ANP task, so in the second model we choose the two longest mini-sequences to train the ANP task, while the MNM task is trained on full sequences, denoted as PF-HIN(full-mini). (iii) The third is that the ANP and MNM tasks are both trained on (type-based) mini-sequences, denoted as PF-HIN(mini-mini). Table 3.9 shows the results of the ablation analysis. PF-HIN(full-full) outperforms PF-HIN(full-mini), which illustrates that despite taking heterogeneous features into consideration, only choosing two mini-sequences for the ANP task may harm the performance as information is missed. However, PF-HIN(mini-mini) outperforms PF-HIN(full-full), showing that considering heterogeneous features on the MNM task may help boost the model performance. The strategy selected for PF-HIN achieves the best results.

Table 3.9: Ablation analysis of the pre-training strategy on the DBLP dataset. Same abbreviations used as in Table 3.6.

| | LP | | SS | MC | | NC | |
|---|---|---|---|---|---|---|---|
| Model | AUC | F1 | AUC | MIC | MAC | NMI | ARI |
| PF-HIN | **0.692** | **0.442** | **0.612** | **0.318** | **0.311** | **0.770** | **0.773** |
| PF-HIN(full-full) | 0.671 | 0.419 | 0.589 | 0.295 | 0.294 | 0.747 | 0.757 |
| PF-HIN(full-mini) | 0.659 | 0.407 | 0.577 | 0.287 | 0.283 | 0.733 | 0.751 |
| PF-HIN(mini-mini) | 0.679 | 0.429 | 0.596 | 0.307 | 0.301 | 0.759 | 0.766 |

Table 3.10: Ablation analysis of the rank-guided heterogeneous walk sampling strategy on the DBLP dataset. Same abbreviations used as in Table 3.6.

| | LP | | SS | MC | | NC | |
|---|---|---|---|---|---|---|---|
| Model | AUC | F1 | AUC | MIC | MAC | NMI | ARI |
| PF-HIN | **0.692** | **0.442** | **0.612** | **0.318** | **0.311** | **0.770** | **0.773** |
| PF-HIN(BFS) | 0.659 | 0.417 | 0.577 | 0.279 | 0.292 | 0.743 | 0.758 |
| PF-HIN(DFS) | 0.643 | 0.406 | 0.565 | 0.269 | 0.283 | 0.741 | 0.752 |
| PF-HIN(random) | 0.668 | 0.423 | 0.589 | 0.292 | 0.297 | 0.657 | 0.764 |

### Effect of rank-guided heterogeneous walk sampling

In this chapter, we have adopted a rank-guided heterogeneous walk sampling strategy to sample nodes to form input sequences. Here we consider three variants. The first is to only use a breadth first search (BFS) sampling strategy, denoted as PF-HIN(BFS); the second is to only use a depth first search (DFS) sampling strategy, denoted as PF-HIN(DFS); and the last is to randomly choose neighboring nodes to form the node sequence, denoted as PF-HIN(random).

Table 3.10 shows the experimental results. PF-HIN(BFS) outperforms PF-HIN(DFS) and PF-HIN(random); aggregating a node's closest neighborhood's information is more informative than choosing far-away nodes or randomly chosen neighboring nodes. PF-HIN outperforms PF-HIN(BFS); choosing nodes with a higher importance leads to better performing feature representations of a HIN.

Notice that sampling random walk treats high rank nodes and less ranked nodes equally. PF-HIN outperforms PF-HIN(random), which further proves that sampling influential and representative nodes will improve the overall model performance.

### Effect of the centrality metrics

In this chapter, we use three centrality metrics to measure the importance of a node. Here we introduce three variants. The first is to remove the betweenness centrality denoted as PF-HIN\betweenness; the second is to remove the eigencentrality denoted as PF-HIN\eigen; the third is to remove the closeness centrality denoted as PF-HIN\closeness. Each variant assigns equal weights for the left two metrics.

Table 3.11 presents the experimental results. Removing a metric influences the experimental results, which illustrates that each metric is necessary for ranking the

Table 3.11: Ablation analysis of the centrality metric on the DBLP dataset. Same abbreviations used as in Table 3.6.

| | LP | | SS | MC | | NC | |
|---|---|---|---|---|---|---|---|
| Model | AUC | F1 | AUC | MIC | MAC | NMI | ARI |
| PF-HIN | **0.692** | **0.442** | **0.612** | **0.318** | **0.311** | **0.770** | **0.773** |
| PF-HIN\betweenness | 0.674 | 0.423 | 0.582 | 0.304 | 0.298 | 0.757 | 0.759 |
| PF-HIN\eigen | 0.668 | 0.416 | 0.586 | 0.297 | 0.292 | 0.753 | 0.751 |
| PF-HIN\closeness | 0.671 | 0.420 | 0.577 | 0.301 | 0.302 | 0.761 | 0.754 |

Table 3.12: Ablation analysis of the fine-tuning setting on the DBLP dataset. Same abbreviations used as in Table 3.6.

| | LP | | SS | MC | | NC | |
|---|---|---|---|---|---|---|---|
| Model | AUC | F1 | AUC | MIC | MAC | NMI | ARI |
| PF-HIN | **0.692** | **0.442** | **0.612** | **0.318** | **0.311** | **0.770** | **0.773** |
| PF-HIN(Freeze) | 0.673 | 0.427 | 0.587 | 0.295 | 0.291 | 0.742 | 0.747 |

nodes. In addition, each metric has a different influence on different tasks, so it is reasonable for us to adopt the learnable weights for them.

#### Effect of the fine-tuning setting

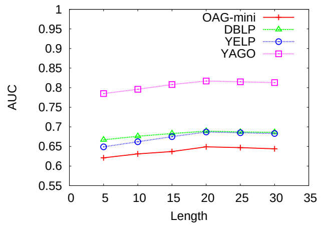
There are two kinds of fine-tuning setting, freezing and full fine-tuning. Freezing fine-tuning is to freeze the parameters of the pre-trained model when fine-tuning, denoted as PF-HIN(Freeze). Full fine-tuning is to train the model with the downstream classifier in an end-to-end manner. PF-HIN leverages the full fine-tuning setting.

Table 3.12 shows the experimental results. PF-HIN consistently outperforms PF-HIN(Freeze), which shows that full fine-tuning is helpful to boost the model performance.

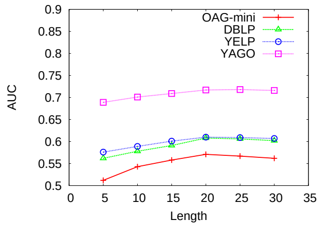
### 3.5.4 Parameter sensitivity

Finally, we conduct a sensitivity analysis of the hyper-parameters of PF-HIN. We choose two parameters for analysis: the maximum length of the input sequence, and the dimension of the node embedding. For each downstream task, we only choose one metric for evaluation: AUC for link prediction, AUC for similarity search, micro-F1 value for node classification, and NMI value for node clustering. Figures 3.2 and 3.3 show the results of our parameter analysis.

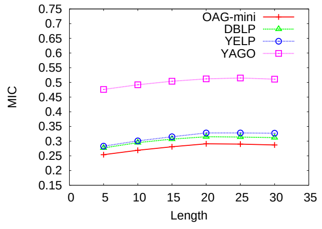
Figure 3.2 shows the results for the maximum length of the input sequence. The performance improves rapidly when the length increases from 0 to 20. A short node sequence is not able to fully express neighborhood information. When the length reaches 20 or longer, the performance stabilizes, and longer sequence lengths may even hurt the performance. Given a node, its neighboring information can be well represented by its direct neighborhood, however, including far-away nodes may introduce noise.



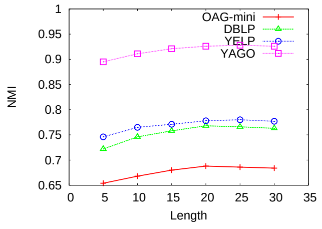
(a) Link prediction



(b) Similarity search



(c) Node classification



(d) Node clustering

Figure 3.2: Sensitivity w.r.t. maximum length of sequences.

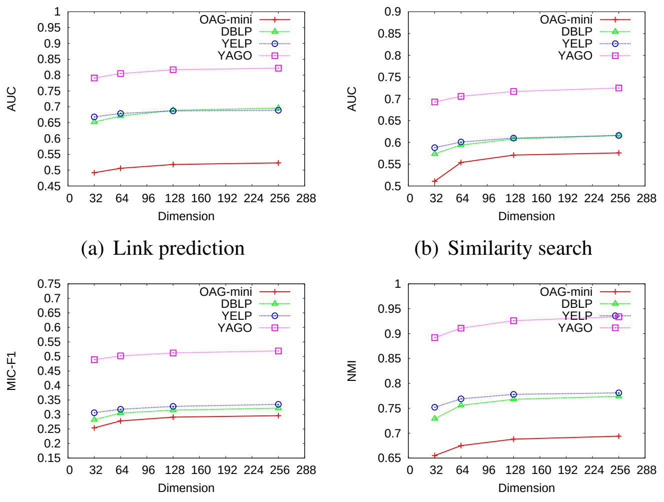
According to this analysis, we choose the length of the input sequence to be 20 so as to balance effectiveness and efficiency.

As to the dimension of node embeddings, Figure 3.3 shows that the performance improves as the dimension increases, for all tasks and datasets. The higher dimensions are able to capture more features. PF-HIN is not very sensitive to the dimension we choose, especially once it is at least 128. The performance gap is not very large between dimension 128 and 256. Thus, we choose 128 as our setting for the dimension of node embeddings for efficiency considerations.

We also conduct a parameter analysis of the percentage of fine-tuning data. GPT-GNN, GCC and PF-HGNN are chosen as methods for comparison as they all follow a pre-training and fine-tuning setup. We choose node classification as an example and choose MIC as the evaluation metric. The experimental results are presented in Figure 3.4. PF-HIN consistently performs best among the pre-training and fine-tuning models, which shows that PF-HIN generalizes well with different percentages of fine-tuning data.

## 3.6 Conclusions

We have considered the problem of network representation learning for heterogeneous information networks (HINs). To answer **RQ2**, we propose a novel model, PF-HIN, to mine the information captured by a HIN. PF-HIN is a self-supervised pre-training and fine-tuning framework. We first use rank-guided heterogeneous walks to generate input sequences and group them into (type-based) mini-sequences. The pre-training tasks we utilize are masked node modeling (MNM) and adjacent node prediction (ANP). Then we leverage bi-directional transformer layers to pre-train the model. We adopt factorized



(a) Link prediction

(b) Similarity search

(c) Node classification

(d) Node clustering

Figure 3.3: Sensitivity w.r.t. the dimension of node embeddings.

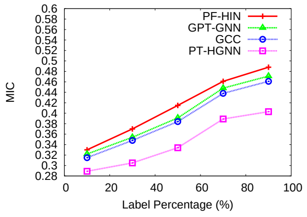


Figure 3.4: Parameter analysis of the percentage of fine-tuning data

embedding parameterization and cross-layer parameter sharing strategies to reduce the number of parameters. We fine-tune PF-HIN on four tasks: link prediction, similarity search, node classification, and node clustering. PF-HIN outperforms state-of-the-art models on the above tasks on four real-life datasets.

In future work, we plan to conduct further graph learning tasks in the context of a diverse range of information retrieval tasks, including, but not limited to, academic search, financial search, product search, and social media search. It is also of interest to see how to model a dynamic HIN that is constantly evolving, using a pre-training and fine-tuning framework.

In this chapter, we have focused on answering **RQ2**, and as part of this, we have proposed a pre-training and fine-tuning framework PF-HIN. It comes with two self-supervised pre-training tasks, i.e., masked node modeling and adjacent node prediction, to mine unsupervised features of HINs. The two tasks are realized by a bi-directional transformer encoder. Next, we turn to another HINs scenario, namely, how to learn representations of HINs in a few-shot setting.

# 4 Few-shot Representation Learning

In this chapter, we intend to answer the following research question:

**RQ3** How to learn the representation of heterogeneous information networks (HINs) in a few-shot setting?

Particularly, we focus on the representation learning of HINs under few-shot scenario. Current approaches to mining graph data often rely on abundant supervised information. However, supervised signals for graph learning tend to be scarce for a new task and only a handful of labeled nodes may be available. Meta-learning mechanisms are able to harness prior knowledge that can be adapted to new tasks. In this chapter, we design a meta-learning framework, called META-HIN, for few-shot learning problems on HINs. Unlike most previous models, which focus on a single task on a single graph, META-HIN is able to deal with three tasks (i.e., node classification, link prediction, and anomaly detection) across multiple graphs. we sample subgraphs to build the support and query set. In our experiments, we fuse several datasets from multiple domains to verify META-HIN's broad applicability in a multiple-graph scenario. META-HIN consistently and significantly outperforms state-of-the-art alternatives on every task and across all datasets that we consider.

## 4.1 Introduction

Representation learning is essential for mining a HIN [61, 64, 119]. Recent efforts resort to graph neural networks (GNNs) to achieve promising results [55, 113]. During the representation learning process, it is taken for granted that the majority of labels in the network is available, and the GNNs are trained in a supervised manner. In practice, however, it is common that only a handful of labels are given, which poses serious challenges to keeping up the performance. In order to effectively mine HINs with scarce labels, we investigate *few-shot learning* problems on HINs in this chapter.

Inspired by meta-learning approaches that have been studied extensively in computer vision, there is an emerging line of research that applies meta-learning to few-shot learning of graph data [6, 17, 35, 48, 122]. In this line of work, well-trained initial

This chapter was submitted as Y. Fang, X. Zhao, W. Xiao, and M. de Rijke. Few-shot learning for heterogeneous information networks. *ACM Trans. Inf. Syst.*, Under review.

parameters of a base GNN are learned and then the learned base-learner is adapted to new tasks, following a standard *model-agnostic meta-learning* (MAML) framework [27].

## Current limitations

Various limitations hinder the application of meta-learning approaches to HINs. First, they are typically designed for *homogeneous* networks, with no prior work yet trying to solve few-shot learning problems on HINs. Second, most of them can only deal with one task on a single graph. For example, Meta-GNN [122] and Meta-MGNN [35] are only designed for a node classification task and cannot be transferred across different graphs, especially those having different distributions; Meta-Graph [6] is only designed for a link prediction task on a single graph; and Meta-GDN [17] is devised for anomaly detection on a single graph. Third, most previous methods overlook the unlabeled information in graphs; making full use of abundant unsupervised information could help improve the performance. Finally, the question of how to leverage the structural information of HINs has not been studied extensively by existing approaches.

## Our proposal

We aim to address the shortcomings listed above, and propose

(1) a heterogeneous GNN module as base model to fully capture heterogeneous information,

(2) a general framework that can be applied across different tasks and graphs,

(3) a GAN-based contrastive module to leverage unsupervised information, and

(4) a structure module to employ graph structural information.

We refer to the above meta-learning framework for few-shot learning problems on HINs as META-HIN. While the tasks addressed by each of the components may have been studied individually in the context of previously proposed models, our work is the *first* to incorporate different modules to address all of the limitations under a single framework and to demonstrate the benefits of this unified approach.

Concretely, we first sample subgraphs from the original graphs to form a support set and query set for both meta-training and meta-testing datasets. Before sampling, we first rank the nodes of a certain neighborhood based on three importance evaluation metrics, i.e., betweenness centrality, eigencentrality, and closeness centrality. Then we adopt rank-guided heterogeneous walks to sample subgraphs in which influential nodes and every type of node will be collected. Note that traditional meta-structure is domain-specific and pre-defined, which is not applicable in meta-learning to transfer to unseen tasks and labels. Next, we apply a structure module on the subgraphs to learn the structural embeddings. Specifically, we adopt an auto-encoder to encode the structural information, based on the intuition that nodes with similar structure will share similar embeddings. Then we apply a heterogeneous GNN module to encode the input subgraphs. We first group the nodes based on their types and apply a Bi-LSTM to each group. The structural information loss by this operation has been preserved by the above structure module. Then we aggregate different type embeddings via a self-attention mechanism to generate the final node embeddings. To make use of unsupervised information, we are the *first* to incorporate a contrastive module in a meta-learning setting before calculating the support loss and query loss. During training, positive samples are the nodes from the given subgraph while negative samples are nodes from other subgraphs. Then we maximize the mutual information between the node embedding and subgraph embedding. Additionally, a GAN-based mechanism is introduced to generate high-quality negative samples that are hard to identify, which further improves the model performance.

### Meta-learning

Different tasks may have different effects on the meta-learner; we employ a self-attention mechanism to calculate the weights of tasks that will be incorporated into the meta-learning process to boost the model performance. Unlike previous models that view a task as a batch of node representations, for META-HIN, a task is a batch of subgraphs. By doing so, META-HIN can adapt to new tasks rapidly and thus have a broader applicability. This explains META-HIN's ability to handle different tasks across different graphs. To the best of our knowledge, ours is the *first* model to simultaneously conduct the following three tasks: (i) node classification, (ii) link prediction, and (iii) anomaly detection.[1]

For the link prediction and anomaly detection tasks, we use a contrastive loss to distinguish positive samples (existing links and abnormal nodes) and negative samples (non-existing links and normal nodes).

### Experiments

To demonstrate that META-HIN is transferable across HINs, aside from using the open academic graph (OAG) dataset, we combine the OAG, DBLP and Aminer dataset to build a new dataset named ODA. These three datasets are all bibliographic networks; we further introduce two datasets from different domains: YELP and YAGO. META-HIN consistently and significantly outperforms the state-of-the-art models on the three tasks across different datasets.

### Contributions

In short, our main contributions can be summarized as follows:

- We propose a meta-learning model META-HIN to deal with few-shot leaning problems in heterogeneous information networks (HINs).
- We sample subgraphs to be trained so that META-HIN is applicable to three tasks and transferable across different HINs.

[1]With minor additional efforts, META-HIN can be adapted to handle more downstream tasks under the proposed framework, which is left to future work.

- We adopt a structural module, a heterogeneous module, and a GAN-based contrastive module to capture the structural information, heterogeneous features, and unlabeled information of a subgraph, respectively.
- We show that META-HIN significantly and consistently outperforms state-of-the-art alternatives on three tasks across multiple HINs.

**Organization**

Section 4.2 introduces related work and Section 4.3 provides a detailed account of the META-HIN framework. Section 4.4 presents the experimental setup; Section 4.5 analyses the experimental results. Section 4.6 concludes the chapter.

## 4.2 Related work

### 4.2.1 Graph neural networks

To learn the representation of a network, many researchers propose variants of GNN models which have shown promising performance.

Some models are based on spectral graph theory. One line of GNN research is based on spectral graph theory [7, 14, 41, 55, 59, 60]. For example, Bruna et al. [7] conduct the spectral convolution operations on the whole graph. To improve scalability, graph convolutional networks (GCNs) [55] leverages the first-order approximation of spectral graph convolutions. Such spectral operation always conducts on the whole graph which may cause inefficiency issue. Therefore, another line of research named as spatial GNN is proposed [29, 38, 71, 73]. For example, GraphSAGE [38] adopts a random walk to sample the neighboring nodes to be processed further via CNN or LSTM operation. Gao et al. [29] sample subgraphs and adapts them to be processed via normal convolution layer.

There are also many variants of GNN [57, 101, 117], fusing neighboring substructures to learn the network embedding. For example, GAT [101] harnesses masked self-attention layers to learn the weights of nodes in a neighborhood. GIN [109] applies injective multiset functions for neighborhood aggregation by parameterizing universal multiset functions with neural networks.

However, these models are all devised for homogeneous networks by aggregating nodes or walks regardless of their types. Few methods focus on the representation of HINs. Therefore, we propose a heterogeneous GNN encoder to fully capture the type information behind a HIN.

### 4.2.2 Contrastive learning

Contrastive learning is a line of self-supervised learning which could capture the unlabeled information of a HIN. In natural language processing, to learn word embeddings, word2vec [70] uses co-occurring words and negative sampling to capture similarity from data. In computer vision, self-supervised image representation is learned by minimizing the distance between two views of the same image [36, 40, 96]. For example, [40] presents momentum contrast (MoCo) for visual representative learning by constructing a dynamic dictionary and a moving-averaged encoder.

Recently contrastive learning approaches devised for graph data are also widely proposed. GCC [79] adopts the InfoNCE loss [100] and uses the instance discrimination task [108] to distinguish the nodes from the same subgraph and other subgraphs. DGI [102] aimis to maximize the mutual information between node representation and graph representation. GMI [76] proposes to make a contrast between contrast between center node representation and its local patch representation learned from structural information and node features. Infograph [90] is based on DGI, which maximize the mutual information between the local representation and global representation, and use GIN as the base GNN encoder. GraphCL [37] generates two randomly perturbed subgraphs of a given node, and maximize the mutual information between them. Hassani and Ahmadi [39] propose structure-space augmentation to increase the number of views and contrast node and graph embeddings across views. Mu et al. [72] propose to an alignment strategy based on mutual information maximization, which aligns the explicit disentangled representations based on knowledge with the implicit disentangled representations learned from user-item interaction

To the best of our knowledge, we are the first to incorporate the contrastive learning with meta-learning setting to leverage the unsupervised information.

### 4.2.3 Few-shot learning

Few-shot learning models can be grouped into two types, i.e., (i) based on metric-based learning, and (ii) based on gradient-based learning. The former is to learn a generative metric that is able to compare and match few-samples, while the latter leverages a specific meta-learner to learn the well-initialized parameters of the base model, which will be further adapted to new tasks.

In this chapter, we mainly focus on gradient-based learning for graph data. Cui et al. [12] address the cold-start issues in sequential knowledge graphs under the meta-learning framework. Niu et al. [74] address a few-shot knowledge graph completion problem, using relational learning incorporated with the gated and attentive neighbor aggregator. Jiang et al. [53] also focus on knowledge graph completion, and propose a meta pattern learning framework to predict new facts of relations under a few-shot setting. Meta-GNN [122] deals with the node classification problem; it obtains prior knowledge by training similar few-shot learning tasks and classifies nodes from new classes with a few labeled samples. It applies a GNN encoder on the entire graph, while META-HIN trains subgraphs individually via a GNN encoder. AMM-GNN [104] and RALE [67] also only deal with the node classification task; the former employs attribute matching and the latter adopts the relative location of a hub-node. Meta-Graph [6] focuses on the link prediction task; it introduces a graph signature function to bootstrap fast adaptation to new graphs. For Meta-GNN and Meta-Graph, a task is a batch of node representations while for META-HIN it is a batch of subgraphs. This important difference allows META-HIN to be adapted to new tasks rapidly, leading to broader applicability on different meta-learning problems and graphs. Meta-GNN and Meta-Graph are only applicable to a single graph.

G-META [48] is applied to node classification and link prediction; it follows a

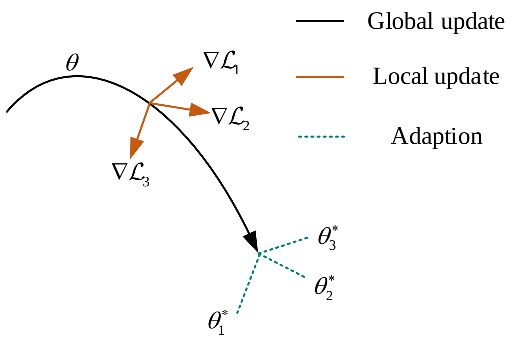


Figure 4.1: The meta-learning framework.

standard MAML framework and uses a GNN as its base model. It also trains on the subgraphs, so it can be applied across different graphs. However, in its own setting, for the multiple-graph scenario, it chooses different protein networks that are from the same domain. GFL [114] faces the same issue as it employs auxiliary graphs of the same distribution to facilitate its performance. In our experiments, the multiple-graph scenario is composed of graphs from the same domain as well as different domains. Meta-MGNN [35] focuses on the molecular property prediction task on a single graph, which can be regarded as node classification. It also makes use of unlabeled information via generative learning. This unsupervised module is incorporated with a MAML framework to deal with a few-shot problem. Meta-GDN [17] focuses on anomaly detection on a single graph. It introduces a new graph neural network, namely a graph deviation network (GDN), as its base model. Zhuang et al. [127] propose a dataset designed for few-shot learning on HINs. However, it is not practical as it creates 80 classes for every node, which is quite rare in real-world settings.

What we contribute to few-shot learning on top of the work listed above is that we offer the first model to deal with three tasks on HINs. We also equip our model with a structural module to fully capture the structural information. And we are the first to incorporate contrastive learning into meta-learning.

## 4.3 The proposed model META-HIN

### 4.3.1 Preliminaries and overview

Let $G = (V, E, T)$ denote a *heterogeneous information network* (HIN), in which $V$ and $E$ denote the node set and edge set, respectively; $T_V$ and $T_E$ denote the node type set and edge type set, respectively. A HIN is a network where $|T_V| > 1$ and/or $|T_E| > 1$. We use $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$ to denote a set of graphs and $\mathcal{Y} = \{y_1, y_2, \ldots, y_M\}$ to denote the label set. Only a handful of labeled nodes are given, and the goal of our work is to learn the initial parameters $\theta$ of a meta-learner, and then adapt the learner to new graphs, tasks, and labels, that is, to correctly map new nodes to new labels given the limited number of labeled nodes.

The gradient based meta-learning is shown as Figure 4.1. During the meta-learning

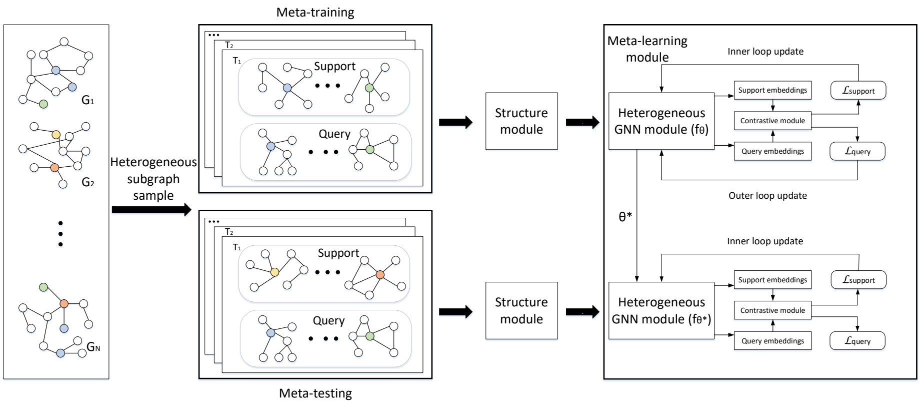


Figure 4.2: The META-HIN framework. The heterogeneous subgraph sampling strategy (left) is detailed in Section 4.3.2; the structure module (center, top) in Section 4.3.3; the meta-learning module (right) in Section 4.3.4, with the heterogeneous GNN module (right, bottom) described in Section 4.3.4 and the contrastive module (right) in Section 4.3.4.

process, there exist two kinds of update operations, that is, local update and global update. In specific, firstly the model locally updates the parameter $\theta$ to $\theta_i^*$ on the support set (learning process), and it searches for the task-specific desired parameters. Based on the task-specific parameter $\theta_i^*$, the model further updates its global parameters to minimize the loss of query set over different tasks (learning-to-learn process). Finally, the learned global parameters $\theta$ can fit into various tasks.

A visual sketch of the proposed meta-learning framework for heterogeneous information networks (META-HIN) is given in Figure 4.2. Under the meta-learning framework, in a $k$-shot meta-training phase, for each task $\mathcal{T}_\tau$ sampled from distribution $p(\mathcal{T})$, only $k$ data samples will be used for training (i.e., the *support* set, denoted as $\mathcal{G}_\tau$), and the remainder will be used for evaluation (i.e., the *query* set, denoted as $\mathcal{G}'_\tau$). During meta-training, the parameters $\theta$ will first be updated on the support set, and then further optimized on the query set using some loss function. After a sufficient amount of training, meta-testing employs the learned parameters $\theta^*$ as initialization to adapt to new tasks quickly with only $k$ samples (as the support set). As meta-training and meta-testing data are not naturally available, a sampling strategy is needed to construct the datasets.

## 4.3.2 Sampling strategy

To construct meta-training and meta-testing datasets, we sample the subgraphs from the given graph sets $\mathcal{G}$. As illustrated in [48], sampling local subgraphs will not cause a loss of necessary information compared to the entire graph. Since it is tricky to propagate information through the whole graph using only a handful of nodes, in this chapter, we also choose to sample subgraphs for the training and testing. In addition, sampling subgraphs also enables our model to transfer knowledge across different graphs [48].

Specifically, to form the subgraph, we first rank the neighboring nodes of the given labeled node based on their structural importance. To measure structural importance, we adopt the concept of node centrality in [5]. We include three centrality metrics, i.e., (i) betweenness centrality, (ii) eigencentrality, and (iii) closeness centrality. *Betweenness centrality* is used to measure the fraction of shortest paths passing through a given node. *Eigencentrality* calculates the influence of a node on its neighbors. *Closeness* centrality is used to compute the total length of the shortest paths between the given node and others. We assign learnable weights to these metrics.[2]

Next, we adopt a so-called *rank-guided heterogeneous walk* to construct the subgraph, which is meant to capture heterogeneous and structural features of a node's neighbor. The walk is with restart; it will iteratively travel from a node to its neighbors. It starts from a given node, and it first reaches out to a node with a higher rank, which is what makes the walk rank-guided. This walk will not stop until it collects a predetermined number of nodes. In order to equip the model with a sense of heterogeneity, we constrain the number of different types to be collected in the sequence so that every type of node can be included. The nodes are sorted based on each node's rank, which serves as a kind of sequential information.

Unlike traditional sampling strategies like random walks, breadth-first search or depth-first search, our sampling strategy is able to extract important and influential neighboring nodes for each node by selecting nodes with a higher rank; this allows us to capture more representative structural information of a neighborhood. Our sampling strategy collects all types of nodes for each neighborhood while traditional strategies ignore the nodes' types. Empirical results with an analysis are provided in Section 4.5.2.

Note that meta-paths, meta-graphs, or network schemas can also be used to explore the heterogeneity of a HIN. However, they are domain-specific and usually pre-defined by domain experts, which makes it hard to apply them in the meta-learning setting to transfer the prior knowledge to new tasks.

### 4.3.3 Structure module

Before applying the meta-learning module to transfer knowledge to different tasks and graphs, we first introduce a structure module to preserve the structural information of a subgraph. Specifically, we adopt the autoencoder mechanism to preserve the graph structure. For each subgraph, we have its adjacency matrix $A = \{a_1, a_2, \ldots, a_n\}$, in which $a_i$ is a row of $A$ representing the nodes that are adjacent with node $i$. Then it is fed into the encoder to obtain the latent representation of node $i$ as follows:

$$h_i^{(1)} = \delta(W^{(1)} a_i + b^{(1)}), \tag{4.1}$$

$$h_i^{(k)} = \delta(W^{(k)} h_i^{(k-1)} + b^{(k)}), \tag{4.2}$$

in which $k$ is the number of layers of the encoder, which we set to 2 in practice; $W^{(k)}$ is the parameter matrix of the $k$-th layer; $b^{(k)}$ is the bias of the $k$-th layer; and $\delta$ is the activation function.

In the decoder we reverse the above process to obtain the reconstructed output $\hat{a}_i$.

[2]Additional metrics that are of particular interest to the user could also be added to the above three metrics. In our implementation, we use the above three metrics to illustrate the effectiveness of META-HIN.

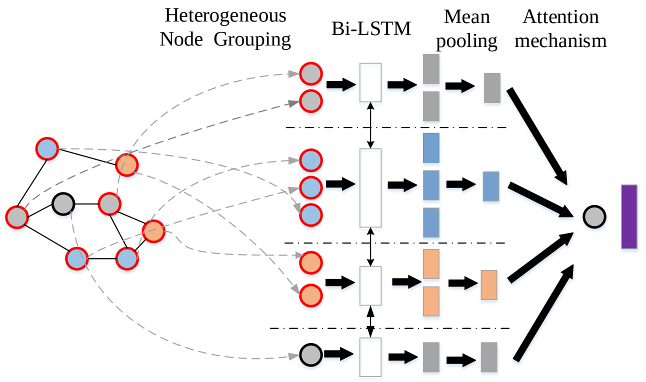


Figure 4.3: Framework of the heterogeneous GNN module.

The objective function of the autoencoder is to minimize the reconstruction error of the input and output. Mathematically,

$$\mathcal{L}_{structure} = \sum_{i=1}^{n} | (\hat{a}_i - a_i) |_2^2 . \tag{4.3}$$

This reconstruction criterion forces nodes with a similar structure to have similar representations. To alleviate the sparsity issue, we impose more penalty on non-zero elements than on zero elements. Then the loss function is defined as:

$$\mathcal{L}_{structure} = \sum_{i=1}^{n} | (\hat{a}_i - a_i) \odot b_i |_2^2 \tag{4.4}$$

$$= | (\hat{A} - A) \odot B |_F^2, \tag{4.5}$$

in which $b_i = \{b_{i,j}\}_{j=1}^{n}$, and $b_{i,j} = 1$ if there exists no edge between nodes $i$ and $j$, otherwise $b_{i,j} = \beta > 1$; $\odot$ is element-wise product.

### 4.3.4 Meta-learning module

After representing the structural information of the subgraphs, we introduce the meta-learning framework to deal with the few-shot setting. We first introduce a heterogeneous GNN module to encode subgraphs. Then comes the contrastive module to make use of unlabeled information.

#### Heterogeneous GNN module

We first group the nodes of the subgraph based on their types, that is, nodes with the same type are grouped together. The structure of the subgraph may be affected by such operations, however, the structural information has already been preserved by the structure module. Then we apply a Bi-LSTM on these groups to extract type-specific features. A Bi-LSTM can capture the feature interactions between nodes in the subgraphs. The representation of the $i$-th node of the group is denoted as $x_i$. Then the hidden representation $h^v_{T_j}$ of type $T_j$ after the Bi-LSTM layer can be represented as:

$$h^v_{T_j} = \frac{\sum_{i \in S^v_{T_j}} \text{Bi-LSTM}\{x_i\}}{|S^v_{T_j}|}, \tag{4.6}$$

where $S^v_{T_j}$ denotes the node group having type $T_j$; the Bi-LSTM is composed of a forward LSTM and a backward LSTM. The forward LSTM is specified as follows:

$$\begin{aligned}
\mathbf{j}_i &= \delta(\mathbf{W}_{xj}x_i + \mathbf{W}_{hj}h_{i-1} + \mathbf{W}_{cj}c_{i-1} + \mathbf{b}_j), &(4.7)\\
\mathbf{f}_i &= \delta(\mathbf{W}_{xf}x_i + \mathbf{W}_{hi}h_{i-1} + \mathbf{W}_{cf}c_{i-1} + \mathbf{b}_f), &(4.8)\\
\mathbf{z}_i &= \tanh(\mathbf{W}_{xc}x_i + \mathbf{W}_{hc}h_{i-1} + \mathbf{b}_c), &(4.9)\\
c_i &= \mathbf{f}_i \odot c_{i-1} + \mathbf{j}_i \odot \mathbf{z}_i, &(4.10)\\
\mathbf{o}_i &= \delta(\mathbf{W}_{xo}x_i + \mathbf{W}_{ho}h_{i-1} + \mathbf{W}_{co}c_i + \mathbf{b}_o), &(4.11)\\
h_i &= \mathbf{o}_i \tanh(c_i), &(4.12)
\end{aligned}$$

where $h_i \in \mathbb{R}^{d/2}$ denotes the output hidden representation, $\mathbf{W} \in \mathbb{R}^{(d/2)\times(d/2)}$ and $b \in \mathbb{R}^{d/2}$ are learnable parameters, denoting weight and bias, respectively; $\delta$ denotes the activation function; $\mathbf{j}_i$, $\mathbf{f}_i$, $\mathbf{o}_i$ are the input gate vector, forget gate vector and output gate vector, respectively. The output of the forward and backward LSTMs are concatenated, along with a mean-pooling layer to generate the hidden embedding of a specific type.

We combine the type-specific embeddings via an attention mechanism to generate the final representation of the given node. The reason for using an attention mechanism is that different types of node may have a different impact on the given node. Mathematically,

$$h_v = \sum_{T_j \in \{T\}} \alpha^{v,j} h^v_{T_j}, \tag{4.13}$$

$$\alpha^{v,j} = \frac{\exp\{\delta(u^{\mathrm{T}} h^v_{T_j})\}}{\sum_{T_l \in \{T\}} \exp\{\delta(u^{\mathrm{T}} h^v_{T_l})\}}, \tag{4.14}$$

where $\delta$ denotes the activation function, for which we use a *LeakyReLU*; $u \in \mathbb{R}^d$ is the attention parameter.

### Contrastive module

Many previously proposed meta-learning models ignore unsupervised information in a few-shot setting. However, only employing supervised signals may limit the performance because only a handful of labeled nodes may be available. Therefore, we introduce a contrastive module to make full use of unlabeled nodes in the graph.

After the structural module and heterogeneous GNN module, we obtain the node embeddings of a subgraph denoted as $H = \{h_1, h_2, \ldots, h_n\}$. We summarize the node embeddings of a subgraph using a *readout function* to generate the subgraph embedding $g$. That is, $g = \text{READOUT}(H)$, where READOUT may be any permutation invariant function; we simply use mean pooling here. Our goal is to maximize the mutual

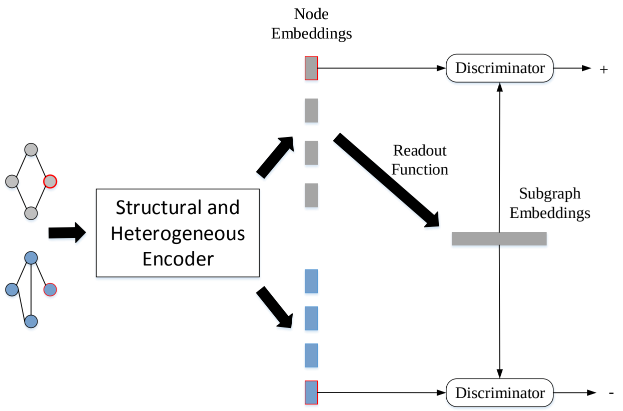


Figure 4.4: Framework of the contrastive module.

information between the node representation and the subgraph representation. In this way, the subgraph representation is able to represent aspects of the data that are shared across all substructures.

For contrastive learning, for a given subgraph, the positive sample is the node that belongs to the subgraph, and the negative sample is a node from other subgraphs. Note that there will not be duplicate nodes in a single task. The negative node embeddings of another subgraph are denoted as $H' = \{h'_1, h'_2, \dots, h'_n\}$.

Next, we present a *discriminator* $\mathcal{D}(h_i, g)$ that calculates the probability score between this node and subgraph pair. The score will be higher if the node belongs to the subgraph.

To define the loss function, we adopt a noise-contrastive type objective with a standard binary cross-entropy loss between positive and negative samples [42]. Mathematically, the combination can be represented as:

$$\mathcal{L}_{contrastive} = \sum_{i=1}^{n} \mathbb{E}_H \left[\log \mathcal{D}(h_i, g)\right] + \sum_{j=1}^{n} \mathbb{E}_{H'} \left[\log(1 - \mathcal{D}(h'_j, g))\right]. \tag{4.15}$$

By doing so, the mutual information between node embedding $h_i$ and subgraph embedding $g$ is maximized via Jensen-Shannon divergence between positive and negative samples [75]. Note that each node embedding obtained in the given subgraph is required to compute the mutual information with the representation of this subgraph.

To further boost the model performance, we introduce a GAN-based mechanism to generate negative samples that are hard to be distinguished. In addition to the discriminator discussed above, a generator is introduced. They are trained in an alternating fashion. Aside from distinguishing original negative and positive samples, the discriminator is also first trained to identity the negative samples generated from the generator. Then the generator is trained to generate high-quality negative samples to fool the discriminator.

### Supervised loss

Here we use the supervised signal to calculate the training loss. For the node classification task, we first apply a *multi-layer* perception (MLP) on top of the subgraph embedding $s$, denoted as: $\hat{y} = \mathrm{MLP}(s)$. Next, we leverage the cross entropy loss between the predicted labels and ground-truth labels:

$$\mathcal{L}_{supervised}^{classification} = -\frac{1}{m}\sum_{i=1}^{m} \mathrm{CrossEntropy}(y_i, \hat{y_i}), \tag{4.16}$$

where $m$ denotes the number of samples.

The link prediction and anomaly detection could be regarded as binary classification tasks. However, instead of simply adopting binary entropy loss, we borrow the idea of contrastive learning and employ the contrastive loss. Specifically, for link prediction, an existing link is the positive signal, and for anomaly detection, an abnormal node is regarded as positive. Note that link prediction could be realized by a pair of nodes in the subgraph. Then we introduce a Noise-Contrastive Estimation (NCE) Loss [100] as follows:

$$\mathcal{L}_{supervised}^{contrastive} = -\log \frac{\exp(q^{\mathrm{T}}k^{+}/\gamma)}{\sum_{i=1}^{m} \exp(q^{\mathrm{T}}k_i/\gamma)}, \tag{4.17}$$

where $k$ is the node embedding of the support set, $q$ is the node embedding of the query set, and $\gamma$ is the temperature hyperparameter; we set $\gamma = 0.07$.

### Joint loss

For a given task $\mathcal{T}_i$, the joint loss of the meta-training process is a combination of the contrastive loss and supervised loss. Mathematically,

$$\mathcal{L}_{\mathcal{T}_\tau}(\theta) = \lambda \mathcal{L}_{constrative} + (1-\lambda)\mathcal{L}_{supervised}, \tag{4.18}$$

where $\lambda$ is the trade-off parameter; we set $\lambda = 0.2$.

### Optimization-based meta-learning

Given the support set $\mathcal{G}_\tau$ and query set $\mathcal{G}'_\tau$, the optimization approach first adapts the initial model parameters $\theta$ to $\theta'_\tau$ for each learning task in support $\mathcal{T}_\tau$ independently. A batch of training samples will be leveraged to calculate the updated parameter $\theta'_\tau$. This process can be represented as:

$$\theta'_\tau = \theta - \epsilon \nabla_\theta \mathcal{L}_{\mathcal{T}_\tau}(\theta), \tag{4.19}$$

where $\epsilon$ is the step size. Suppose we have a task distribution $\mathcal{T}_\tau \sim p(\mathcal{T})$, then we employ stochastic gradient descent (SGD) to update the model parameters across all tasks in the query set. Mathematically,

$$\theta = \theta - \mu \nabla_\theta \sum_{\mathcal{T}_\tau \sim p(\mathcal{T})} \eta(\mathcal{T}_\tau)\mathcal{L}'_{\mathcal{T}_\tau}(\theta'_\tau), \tag{4.20}$$

where $\mu$ is the meta-learning rate and $\mathcal{L}'_{\mathcal{T}_\tau}$ is the joint loss over query set of $\mathcal{T}_\tau$. Note that different tasks may contribute differently to the meta-learner. We add a self-attention layer to measure the task weight $\eta(\mathcal{T}_\tau)$. We first calculate the task embedding. Given the task $\mathcal{T}_\tau$, its representation $H_{\mathcal{T}_\tau}$ is calculated as the average of all node embeddings of $\mathcal{T}_\tau$. Mathematically,

$$H_{\mathcal{T}_\tau} = \mathrm{AVERAGE}(h_{\mathcal{T}_\tau}{}_{i=1}^{n}). \tag{4.21}$$

Then $\eta(\mathcal{T}_\tau)$ is computed as follows:

$$\eta(\mathcal{T}_\tau) = \frac{\exp(\delta(\omega^{\mathrm{T}} H_{\mathcal{T}_\tau}))}{\sum_{\mathcal{T}_{\tau'} \in \mathcal{T}} \exp(\delta(\omega^{\mathrm{T}} H_{\mathcal{T}_{\tau'}}))}, \tag{4.22}$$

where $\delta$ is the activation function and we leverage *LeakyReLU*; $\omega \in \mathbb{R}^d$ is the attentive parameter.

During the meta-testing phase, we repeat the above process using the final updated parameter $\theta^*$. We learn $\theta^*$ from knowledge across all meta-training tasks and it is the optimal parameter to adapt to unseen tasks quickly.

#### Summary

Algorithm 5 presents the overall learning procedure of META-HIN. Given heterogeneous graphs and randomly initialized parameters $\theta$ as input, the algorithm constructs a support set and a query set via its heterogeneous subgraph sampling strategy (line 1). Then we sample a batch of tasks (line 3) and for all tasks, we sample $k$ support graph instances (line 5). After that we iterate over the sampled graphs (line 6–10), where in each graph we first generate structural node embeddings and then feed them into the heterogeneous GNN module for an update. We obtain the task embedding based on the node embeddings (line 11). We calculate the loss function (line 12) and update the parameter based on the loss (line 13). After this, we sample $n$ query graph instances (line 14) and follow a similar procedure as with the support set to learn the node embedding and loss function (line 15–20). Then we calculate the task weight (line 22) to update the final adapted parameters $\theta$ (line 23).

## 4.4 Experimental setup

We detail our datasets, baseline models, and parameter settings.

### 4.4.1 Datasets

We first adopt three bibliographic graph networks, i.e., (i) OAG,[3] (ii) DBLP,[4] and (iii) AMiner,[5] which are all heterogeneous information networks with four types of nodes (i) authors, (ii) papers, (iii) venues, and (iv) topics. As for the label information, the authors in these networks are split into five areas: (i) information retrieval, (ii) database,

[3]`https://www.openacademic.ai/oag/`
[4]`http://dblp.uni-trier.de`
[5]`https://www.aminer.cn/data`

**Algorithm 5:** The learning algorithm of META-HIN.

```
Input :Heterogeneous Graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$, Randomly
       initialized $\theta$;
1  Construct support set $\mathcal{G}_\tau$ and query set $\mathcal{G}'_\tau$ via Heterogeneous subgraph
     sampling strategy;
2  while not done do
3      Sample batch of tasks $\mathcal{T}_\tau \sim p(\mathcal{T})$;
4      for all $\mathcal{T}_\tau$ do
5          Sample $k$ instances $\{\mathcal{G}_{\tau 1}, \mathcal{G}_{\tau 2}, \ldots, \mathcal{G}_{\tau k}\} \in \mathcal{G}_\tau$;
6          for i=1 to k do
7              Generate structural node embedding via structure module;
8              Feed structural embedding into heterogeneous GNN module;
9              Calculate node representation via Eq. (4.13);
10         end
11         Calculate task embedding via Eq. (4.21);
12         Calculate $\mathcal{L}_{\mathcal{T}_\tau}$ via Eq. (4.18);
13         $\theta'_\tau = \theta - \epsilon \nabla \mathcal{L}_{\mathcal{T}_\tau}$;
14         Sample $n$ instances $\{\mathcal{G}'_{\tau 1}, \mathcal{G}'_{\tau 2}, \ldots, \mathcal{G}'_{\tau k}\} \in \mathcal{G}'_\tau$;
15         for j=1 to n do
16             Generate structural node embedding via structure module;
17             Feed structural embedding into heterogeneous GNN module;
18             Calculate node representation via Eq. (4.13);
19         end
20         Calculate $\mathcal{L}'_{\mathcal{T}_\tau}$ via Eq. (4.18);
21     end
22     Calculate task weight $\eta(\mathcal{T}_\tau)$ via Eq. (4.22) ;
23     $\theta = \theta - \mu \nabla_\theta \sum_{\mathcal{T}_\tau \sim p(\mathcal{T})} \eta(\mathcal{T}_\tau) \mathcal{L}'_{\mathcal{T}_\tau}(\theta'_\tau)$.
24 end
```

(iii) natural language processing, (iv) data mining, and (v) machine learning. Unlike most previous research focusing on a single graph, here we mix the OAG, DBLP, and AMiner datasets to construct a new dataset named ODA.

The OAG, DBLP, and AMiner datasets are all from the same domain. We include two other social graphs from different domains. One is YELP,[6] containing restaurant reviews and four types of node: (i) customers, (ii) restaurants, (iii) reviews, and (iv) food-related keywords. The restaurants are labeled as (i) Chinese food, (ii) fast food, and (iii) sushi bar. The other social graph that we add is YAGO,[7] which contains movie information with nodes having five types: (i) movie, (ii) director, (iii) actor, (iv) producer, and (v) composer. The movies are labeled as (i) action, (ii) adventure, (iii) sci-fi, (iv) crime, and (v) horror.

Next, we create two mixed datasets. One combines OAG and YELP, denoted as

[6]https://www.yelp.com/dataset_challenge
[7]https://old.datahub.io/dataset/yago

Table 4.1: Dataset statistics.

| Dataset | #nodes | #edges | #node types | # label |
|---|---|---|---|---|
| OAG | 432,362 | 1,837,362 | 5 | 5 |
| DBLP | 357,362 | 1,547,364 | 5 | 5 |
| AMiner | 263,473 | 1,022,362 | 5 | 5 |
| ODA | 1,053,197 | 4,407,088 | 5 | 5 |
| YELP | 213,476 | 1,622,327 | 4 | 3 |
| OYE | 645,838 | 3,459,689 | 9 | 8 |
| YAGO | 273,276 | 2,452,371 | 5 | 5 |
| OYA | 705,638 | 4,289,733 | 10 | 10 |

OYE; the other combines OAG and YAGO, denoted as OYA.

To conduct the tasks on a single large graph, we only choose OAG to be analyzed for simplicity.[8]

The dataset statistics are shown in Table 4.1.

We have three tasks to be analyzed, i.e., node classification, link prediction and anomaly detection. To simplify our presentation, for our single-graph experiments on those tasks, we opt to zoom in on a single large graph, namely OAG.

### 4.4.2 Algorithms used for comparison

We adopt two GNN models that are directly trained on the supervised information, i.e., GCN [55], which is a spectral graph model, and GraphSage [38], which is a spatial model.

Three baselines specifically designed for heterogeneous information network representation are also compared, i.e., HetGNN [115], HAN [105], and HGT [47].

We also include baselines designed for few-shot problem on graphs, i.e., Meta-Graph [6], Meta-GNN [122], G-META [48], Meta-MGNN [35], and Meta-GDN [17]. Detailed information about these models can be found in Section 4.2.

To increase the number of models used for comparison in a multi-graph scenario, we also include meta-learning baselines that are not specifically designed for graph data. KNN [97] first embeds the meta-training set using a GNN, and each query example is represented via the label of the voted K-closest example in the support set. Finetune [97] first learns the embeddings of the meta-training set and then the models are fine-tuned on the meta-testing set. ProtoNet [89] employs prototype learning on each subgraph embedding and then follows the standard few-shot learning procedure. MAML [27] chooses a MAML framework as the meta-learner.

### 4.4.3 Parameters

As our sampling strategy, we set the length of our rank-guided heterogeneous walk to 20, in other words, the number of nodes in a subgraph is 20. The update step in training

[8]For the user with particular interest, single-graph experiment on DBLP, AMiner, YELP, YAGO is also welcomed. In our implementation, we use OAG to illustrate the model effectiveness on a single graph.

tasks is set to 10 and the update step in testing tasks to 20. We use 10 shots for each task.

In node classification, nodes of two labels will be used for meta-testing datasets and nodes of other labels will be involved in meta-training tasks. In link prediction, we separate $30\%$ of the edges for the support set and $70\%$ of the edges for the query set; the negative edges are randomly sampled having the same number of positive edges. The detailed settings for link prediction can be found in [118]. In both link prediction and anomaly detection, there are no overlapping edges to be predicted and anomalies to be detected between meta-training and meta-testing datasets.

The dimensions of all embeddings in META-HIN are set to 128. We use grid search to find the best parameter configuration. The task number is chosen from $\{4, 8, 16, 32, 64\}$; the inner update learning rate $\epsilon$ is chosen from $\{0.01, 0.005, 0.001, 0.0005\}$, and the meta-level learning rate $\mu$ is also chosen from $\{0.01, 0.005, 0.001, 0.0005\}$. The optimal parameters are task-specific and dataset-specific. For the other models, we adopt the best configurations reported in the source publications.

We report on statistical significance with a paired two-tailed t-test and we mark a significant improvement of META-HIN over the best baseline for $p < 0.05$ with $^{\blacktriangle}$.

## 4.5 Results and analysis

We present the results of META-HIN on three tasks: (i) node classification, (ii) link prediction, and (iii) anomaly detection. We also conduct an ablation analysis and study the parameter sensitivity.

### 4.5.1 Results of tasks

#### Node classification

Here we report on the results for the multi-label node classification task. We adopt multi-class classification accuracy, ACC, and F1 value as evaluation metrics (five-fold average).

Table 4.2 presents the experimental results on the node classification task; the highest scores are set in bold. N/A means the model does not work in the graph meta-learning problem.

META-HIN consistently and significantly outperforms the baselines on every dataset, which verifies the model's effectiveness. Concretely, GCN, and GraphSage, which lack specific few-shot learning facilities, achieve the worst performance; they are trained in an end-to-end supervised manner but there are only handful of labeled nodes that can be leveraged. HetGNN, HAN, and HGT perform slightly better for leveraging heterogeneous features, but are still incomparable to models equipped with few-shot techniques. Meta-GNN trains on the entire graph, which limits its performance and speaks to the effectiveness of our subgraph sampling strategy; a subgraph is sufficient to capture local features while an entire graph may introduce noise and may be too sparse to be trained.

On a single graph (OAG), Meta-MGNN is the best performing baseline as it also employs unlabeled information using generative learning. However, META-HIN still

Table 4.2: Results on the multi-label node classification task.

| | OAG | | ODA | | OYE | | OYA | |
|---|---|---|---|---|---|---|---|---|
| Model | ACC | F1 | ACC | F1 | ACC | F1 | ACC | F1 |
| GCN | 0.332 | 0.438 | 0.278 | 0.321 | 0.217 | 0.236 | 0.207 | 0.226 |
| GraphSage | 0.387 | 0.452 | 0.284 | 0.302 | 0.221 | 0.272 | 0.218 | 0.263 |
| HetGNN | 0.395 | 0.463 | 0.292 | 0.307 | 0.228 | 0.279 | 0.224 | 0.278 |
| HAN | 0.398 | 0.468 | 0.289 | 0.305 | 0.232 | 0.283 | 0.228 | 0.283 |
| HGT | 0.407 | 0.472 | 0.298 | 0.313 | 0.239 | 0.288 | 0.235 | 0.290 |
| Meta-GNN | 0.457 | 0.523 | N/A | N/A | N/A | N/A | N/A | N/A |
| Meta-MGNN | 0.559 | 0.637 | N/A | N/A | N/A | N/A | N/A | N/A |
| G-META | 0.538 | 0.621 | 0.468 | 0.521 | 0.372 | 0.381 | 0.362 | 0.379 |
| KNN | 0.497 | 0.589 | 0.427 | 0.463 | 0.304 | 0.323 | 0.293 | 0.321 |
| Finetune | 0.413 | 0.538 | 0.389 | 0.413 | 0.291 | 0.312 | 0.272 | 0.295 |
| ProtoNet | 0.472 | 0.548 | 0.411 | 0.421 | 0.297 | 0.327 | 0.283 | 0.301 |
| MAML | 0.485 | 0.611 | 0.419 | 0.409 | 0.299 | 0.328 | 0.295 | 0.312 |
| META-HIN | **0.581**$^\blacktriangle$ | **0.655**$^\blacktriangle$ | **0.516**$^\blacktriangle$ | **0.551**$^\blacktriangle$ | **0.397**$^\blacktriangle$ | **0.414**$^\blacktriangle$ | **0.395**$^\blacktriangle$ | **0.411**$^\blacktriangle$ |

outperforms it and we attribute this to the fact that in addition to the unsupervised information, META-HIN harnesses structural information and heterogeneous features.

Additionally, for models designed for graph meta-learning, only G-META and META-HIN can be applied across different graphs. META-HIN outperforms G-META consistently on every dataset; this reflects the advantage of our structural module, contrastive module, and heterogeneous GNN module. How these modules contribute to META-HIN's performance is examined in detail in Section 4.5.2. As to other few-shot learning methods – KNN, Finetune, ProtoNet, and MAML –, META-HIN consistently outperforms them across different graphs, which confirms that META-HIN's meta-learning framework is more effective on graph data.

### Link prediction

Here we report on the results for the link prediction task. We adopt the AUC value and the F1 value as evaluation metrics (five-fold average).

We present the link prediction results in Table 4.3, with the highest results set in bold. N/A means that a model does not work in the graph meta-learning problem. As in the node classification task, GCN and GraphSage perform the worst because of the limited supervised information. HetGNN, HAN, and HGT perform poorly with no few-shot tricks applied. Meta-Graph, like Meta-GNN in the node classification task, trains on the entire graph causing its worse performance compared to G-META and META-HIN. G-META still performs worse than META-HIN, consistently across different graphs, which, again, verifies the effectiveness of META-HIN by leveraging structural information, unsupervised information, and heterogeneous features.

Table 4.3: Results on the link prediction task.

| | OAG | | ODA | | OYE | | OYA | |
|---|---|---|---|---|---|---|---|---|
| Model | AUC | F1 | AUC | F1 | AUC | F1 | AUC | F1 |
| GCN | 0.632 | 0.536 | 0.472 | 0.417 | 0.324 | 0.275 | 0.259 | 0.236 |
| GraphSage | 0.648 | 0.547 | 0.483 | 0.421 | 0.327 | 0.281 | 0.264 | 0.243 |
| HetGNN | 0.657 | 0.553 | 0.489 | 0.434 | 0.335 | 0.289 | 0.272 | 0.255 |
| HAN | 0.661 | 0.558 | 0.493 | 0.438 | 0.341 | 0.293 | 0.278 | 0.260 |
| HGT | 0.669 | 0.567 | 0.503 | 0.447 | 0.354 | 0.304 | 0.285 | 0.271 |
| Meta-Graph | 0.723 | 0.645 | N/A | N/A | N/A | N/A | N/A | N/A |
| G-META | 0.771 | 0.673 | 0.621 | 0.523 | 0.421 | 0.396 | 0.393 | 0.385 |
| KNN | 0.711 | 0.623 | 0.534 | 0.473 | 0.397 | 0.357 | 0.348 | 0.333 |
| Finetune | 0.682 | 0.601 | 0.523 | 0.444 | 0.356 | 0.326 | 0.311 | 0.298 |
| ProtoNet | 0.726 | 0.647 | 0.598 | 0.492 | 0.407 | 0.364 | 0.365 | 0.358 |
| MAML | 0.731 | 0.658 | 0.592 | 0.485 | 0.412 | 0.371 | 0.371 | 0.363 |
| META-HIN | **0.808**▲ | **0.696**▲ | **0.657**▲ | **0.558**▲ | **0.449**▲ | **0.426**▲ | **0.414**▲ | **0.408**▲ |

### Anomaly detection

In this task, since there are no ground-truth anomalies available in the original datasets, we adopt an anomaly injection model [16] to insert sets of anomalies. We inject structural anomalies, which are actually a set of small cliques. A small clique is regarded as a typical abnormal sub-structure since nodes in the clique are more closely linked to each other than average; 5% of the datasets are created as anomalies that need to be detected. We adopt AUC-ROC and AUC-PR as evaluation metrics. AUC-ROC evaluates the probability that a randomly chosen anomaly has a higher score than a normal node. AUC-PR is the area under the curve of precision against recall at different thresholds; it only assesses a model's performance on the positive class (abnormal nodes); it is calculated as the average precision defined in [69].

Table 4.4 shows the performance on the anomaly detection task. The highest scores are set in bold. N/A means that the model is not applicable on these datasets. Note that G-META, which is able to deal with node classification and link prediction, cannot be applied to the anomaly detection task. This confirms META-HIN's broad applicability; it can deal with three meta-learning tasks. GCN, GraphSage, HetGNN, HAN, and HGT perform the worst, and other general few-shot learning methods, KNN, Finetune, ProtoNet, and MAML, are not comparable to META-HIN. The best performing baseline on a single graph is Meta-GDN, which is designed for few-shot graph anomaly detection, but it cannot be applied in a multiple-graph scenario. META-HIN performs the best across different graphs, which further confirms META-HIN's effectiveness and broad applicability.

## 4.5.2 Ablation analysis

We analyze the contribution of different components of META-HIN via an ablation analysis.

Table 4.4: Results on the anomaly detection task.

| | OAG | | ODA | | OYE | | OYA | |
|---|---|---|---|---|---|---|---|---|
| Model | ROC | PR | ROC | PR | ROC | PR | ROC | PR |
| GCN | 0.472 | 0.278 | 0.387 | 0.212 | 0.253 | 0.115 | 0.215 | 0.089 |
| GraphSage | 0.488 | 0.283 | 0.389 | 0.218 | 0.262 | 0.121 | 0.217 | 0.095 |
| HetGNN | 0.499 | 0.295 | 0.402 | 0.231 | 0.275 | 0.133 | 0.232 | 0.103 |
| HAN | 0.487 | 0.289 | 0.397 | 0.226 | 0.271 | 0.128 | 0.227 | 0.099 |
| HAT | 0.506 | 0.302 | 0.406 | 0.233 | 0.279 | 0.138 | 0.241 | 0.108 |
| Meta-GDN | 0.654 | 0.445 | N/A | N/A | N/A | N/A | N/A | N/A |
| KNN | 0.606 | 0.378 | 0.444 | 0.278 | 0.278 | 0.196 | 0.248 | 0.152 |
| Finetune | 0.572 | 0.353 | 0.402 | 0.248 | 0.257 | 0.165 | 0.236 | 0.132 |
| ProtoNet | 0.622 | 0.378 | 0.478 | 0.391 | 0.289 | 0.211 | 0.259 | 0.205 |
| MAML | 0.616 | 0.389 | 0.481 | 0.398 | 0.296 | 0.217 | 0.264 | 0.211 |
| META-HIN | **0.671**▲ | **0.459**▲ | **0.532**▲ | **0.432**▲ | **0.334**▲ | **0.255**▲ | **0.293**▲ | **0.242**▲ |

### Evaluation setup

To evaluate the contribution of the structure module, we add a variant, denoted as META-HIN\structure, from which we remove the structure module.

To evaluate the effect of the contrastive module, we introduce a variant, denoted as META-HIN\contrastive, in which we remove the contrastive module. And to evaluate the effect of the GAN mechanism, we introduce a variant denoted as META-HIN\GAN, in which we only use the original negative samples from the graph.

As to the heterogeneous GNN module, since we still need an encoder to encode the subgraph features, this module cannot simply be removed. Instead, we introduce two variants for comparison. That is, our heterogeneous GNN module is replaced by a GCN and GraphSage, respectively, denoted as META-HIN(GCN) and META-HIN(GraphSage), respectively.

To evaluate the effect of our choice of contrastive loss in the link prediction and anomaly detection task, we introduce a variant using the traditional binary cross entropy loss, which is denoted as META-HIN(BCE).

To evaluate the effect of our self-attention mechanism to calculate the task weights, we introduce a variant assigning equal weights for each task, denoted as META-HIN(Equal).

We also assess the effect of our sampling strategy. We introduce three variants for comparison: (i) a breadth-first search (BFS) strategy, META-HIN(BFS); (ii) a depth-first search (DFS) strategy, META-HIN(DFS); and (iii) randomly sampling the neighboring nodes, META-HIN(random). Recall that we adopt three centrality metrics to measure the importance of a node before applying our sampling strategy. Here, we introduce three variants to assess the contribution of these centrality metrics. The first excludes the betweenness centrality denoted as META-HIN\betweenness; the second excludes the eigencentrality denoted as META-HIN\eigen; the third excludes closeness centrality denoted as META-HIN\closeness. Each variant assigns equal weights to the two remaining centrality metrics.

Table 4.5: Results of the ablation analysis on the OYE dataset. Abbreviations used: MC – multi-label node classification, LP – link prediction, AD – anomaly detection.

| | MC | | LP | | AD | |
|---|---|---|---|---|---|---|
| Model | ACC | F1 | AUC | F1 | AUC-ROC | AUC-PR |
| META-HIN\structure | 0.359 | 0.383 | 0.398 | 0.378 | 0.279 | 0.195 |
| META-HIN\contrastive | 0.353 | 0.364 | 0.386 | 0.371 | 0.255 | 0.191 |
| META-HIN\GAN | 0.382 | 0.401 | 0.442 | 0.420 | 0.319 | 0.247 |
| META-HIN(GCN) | 0.368 | 0.383 | 0.402 | 0.394 | 0.298 | 0.208 |
| META-HIN(GraphSage) | 0.371 | 0.390 | 0.408 | 0.401 | 0.305 | 0.218 |
| META-HIN(BCE) | N/A | N/A | 0.427 | 0.406 | 0.317 | 0.236 |
| META-HIN(Equal) | 0.375 | 0.395 | 0.428 | 0.411 | 0.318 | 0.239 |
| META-HIN(BFS) | 0.372 | 0.384 | 0.411 | 0.393 | 0.296 | 0.227 |
| META-HIN(DFS) | 0.364 | 0.380 | 0.402 | 0.384 | 0.285 | 0.225 |
| META-HIN(random) | 0.353 | 0.375 | 0.397 | 0.366 | 0.272 | 0.216 |
| META-HIN\betweenness | 0.382 | 0.402 | 0.432 | 0.408 | 0.306 | 0.242 |
| META-HIN\eigen | 0.377 | 0.397 | 0.436 | 0.413 | 0.312 | 0.233 |
| META-HIN\closeness | 0.385 | 0.404 | 0.426 | 0.402 | 0.316 | 0.238 |
| META-HIN | **0.397** | **0.414** | **0.449** | **0.426** | **0.334** | **0.255** |

### Results

We report the experimental outcomes on the two combined datasets with different domains, i.e., OYE and OYA datasets; the findings on the OAG and ODA are qualitatively similar.

Table 4.5 presents the experimental results of our ablation analysis on the OYE dataset. Both the structure module and the contrastive module play a vital role for the meta-learning tasks, as one captures the graph structural information and the other describes the unsupervised information. More concretely, the contrastive module has a greater impact than the structure module as removing it causes a bigger drop in performance. Additionally, incorporating the GAN mechanism with a contrastive module also contributes to the performance improvement.

As for the replacements of our heterogeneous GNN module, both of the variants, i.e., META-HIN(GCN) and META-HIN(GraphSage), perform worse than META-HIN, which we attribute to the fact that META-HIN leverages heterogeneous information.

The variant of META-HIN that uses BCE loss instead of a contrastive loss, performs worse than META-HIN, which is due to the fact that a contrastive loss function is better able to distinguish the binary labels by maximally separating the positive samples from the negative samples.

Assigning equal weights to the node classification, link prediction, and anomaly detection tasks reduces the performance of META-HIN, which supports our intuition that different tasks contribute differently to the meta-learner.

As to alternative sampling strategies, META-HIN(BFS) performs better than both

Table 4.6: Results of the ablation analysis on the OYA dataset. Abbreviations used: MC: multi-label node classification, LP: link prediction, AD: anomaly detection.

| | MC | | LP | | AD | |
|---|---|---|---|---|---|---|
| Model | ACC | F1 | AUC | F1 | AUC-ROC | AUC-PR |
| META-HIN\structure | 0.352 | 0.367 | 0.376 | 0.362 | 0.236 | 0.183 |
| META-HIN\contrastive | 0.347 | 0.356 | 0.359 | 0.352 | 0.219 | 0.177 |
| META-HIN\GAN | 0.377 | 0.393 | 0.403 | 0.388 | 0.266 | 0.203 |
| META-HIN(GCN) | 0.359 | 0.375 | 0.382 | 0.372 | 0.252 | 0.195 |
| META-HIN(GraphSage) | 0.364 | 0.381 | 0.387 | 0.379 | 0.257 | 0.201 |
| META-HIN(BCE) | N/A | N/A | 0.402 | 0.386 | 0.269 | 0.227 |
| META-HIN(Equal) | 0.368 | 0.381 | 0.395 | 0.392 | 0.276 | 0.224 |
| META-HIN(BFS) | 0.365 | 0.369 | 0.386 | 0.371 | 0.257 | 0.204 |
| META-HIN(DFS) | 0.357 | 0.362 | 0.374 | 0.362 | 0.245 | 0.192 |
| META-HIN(random) | 0.345 | 0.357 | 0.365 | 0.348 | 0.235 | 0.178 |
| META-HIN\betweenness | 0.371 | 0.384 | 0.399 | 0.391 | 0.273 | 0.225 |
| META-HIN\eigen | 0.365 | 0.380 | 0.404 | 0.394 | 0.374 | 0.214 |
| META-HIN\closeness | 0.375 | 0.386 | 0.392 | 0.384 | 0.278 | 0.229 |
| META-HIN | **0.395** | **0.411** | **0.414** | **0.408** | **0.293** | **0.242** |

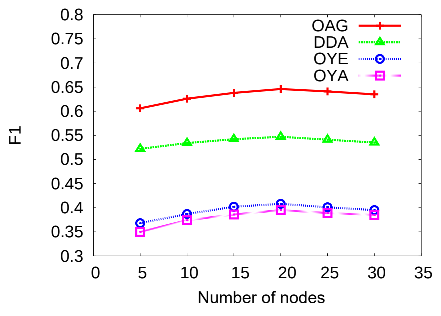
META-HIN(DFS) and META-HIN(random), which illustrates that aggregating a node's closest neighbors is more representative than sampling far-away nodes or randomly chosen nodes. META-HIN, in turn, outperforms META-HIN(BFS); generating the subgraph by selecting nodes with a higher importance results in better depiction of node's neighborhood information.

In summary, the individual design choices we made for the components of META-HIN are justified in the sense that obvious alternatives consistently lead to a performance drop.

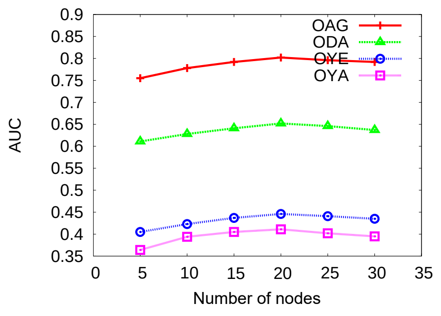
### 4.5.3 Parameter sensitivity analyses

Next, we conduct a parameter sensitivity analysis of META-HIN. Two parameters are chosen for our analysis: (i) the maximum number of nodes of a subgraph, and (ii) the number of shots used for meta-learning. For each task, we only choose one metric for assessment: F1 value for multi-label node classification, AUC for link prediction, and AUC-ROC for anomaly detection. Figure 4.5 and 4.6 present the results of our parameter analyses.

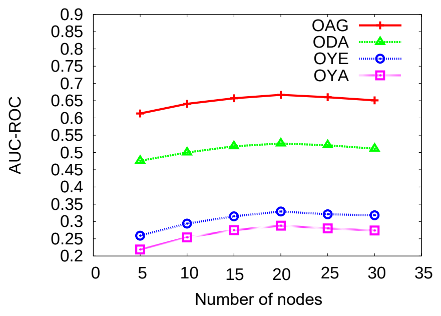
Figure 4.5 shows the sensitivity to the maximum number of nodes of the input subgraph. As the number of nodes increases from 0 to 20 (on the X-axis), the model performance correspondingly improves rapidly. A subgraph with only a few nodes may not be able to fully capture the features of a node's neighborhood. However, when the number of nodes exceeds 20, it negatively impacts the model performance. Including too many far-away nodes may introduce noise: the immediate neighborhood already



(a) Node classification



(b) Link prediction



(c) Anomaly detection

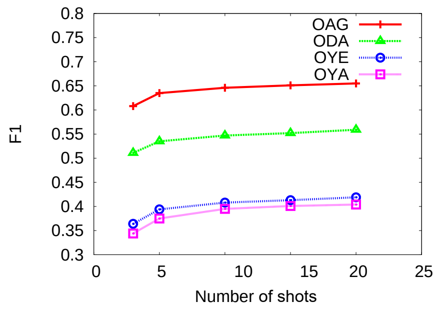
Figure 4.5: Sensitivity w.r.t. the maximum number of nodes of sequences.

provides sufficient neighborhood information. Based on this analysis, we select the maximum number of nodes of the subgraph to be 20 in our experiments.

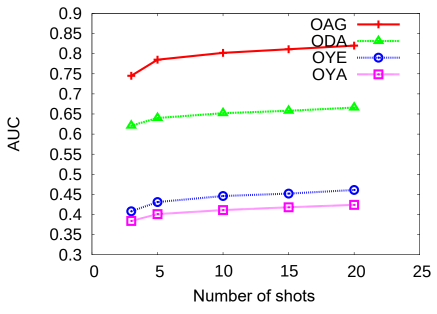
As to the number of shots used for meta-learning, unsurprisingly, including more shots leads to better performance (see Figure 4.6). The use of more training and testing samples consistently helps to improve the performance. However, in a few-shot setting, the shots that could be chosen are limited (by definition); choosing the number of shots to be 10 is a reasonable balance between effectiveness and practical limitations.

## 4.6 Conclusion

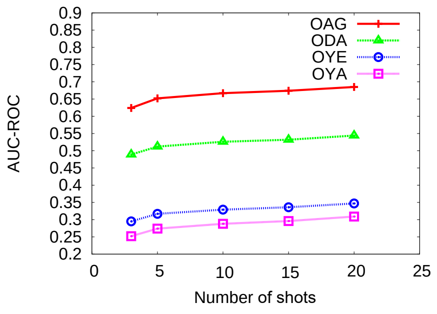
In this chapter, to answer **RQ3**, we propose a novel meta-learning model, META-HIN, to address the few-shot learning problem for HINs. META-HIN is the first model that is applicable to three tasks (i.e., node classification, link prediction, and anomaly detection) across different HINs. As part of META-HIN, we have introduced a structure module, a heterogeneous GNN module, and a GAN-based contrastive module to make effective use of structural, heterogeneous and unsupervised information in a network, respectively. For the link prediction and anomaly detection tasks, we choose a contrastive loss to train the meta-learner. A self-attention mechanism is applied on the training tasks as different tasks have different influences on the meta-learner.



(a) Node classification



(b) Link prediction



(c) Anomaly detection

Figure 4.6: Sensitivity w.r.t. the number of shots for meta-learning.

In our experiments, META-HIN consistently and significantly outperforms state-of-the-art methods on every task across datasets, which demonstrates the advantage of META-HIN. We conduct an ablation analysis to evaluate the effect of different modules or strategies we adopt. The results verify the effectiveness of META-HIN's design. META-HIN can deal with both single-graph scenario and multiple-graph scenarios, while in a multi-graph scenario, aside from graphs from the same distribution, graphs from other domains can also be processed. In addition, META-HIN is able to handle three tasks in a general framework, with each . We also conduct parameter sensitivity analyses to demonstrate the stability of our model.

This work fills the gap on addressing the few-shot learning issues on HINs. It can be used in broader real-life applications such as recommendation. Many recommender systems face cold-start issue with limited items or users, which is a typical few-shot scenario. Our technique is promising to alleviate the above issue if regard the recommender system as a HIN. With only a handful of labels, when coming a new item, META-HIN could help predict which user might be interested like the link prediction task. In addition, anomaly detection could also be conducted by META-HIN in real-life financial or social networks to find the abnormal elements which are rare and limited labeled.

META-HIN could also be improved in several ways. First of all, attribute information exists in many HINs, which could be further incorporated into our framework in future work. Secondly, we aim to realize automatic hyperparameter optimization and reduce the number of parameters to further improve META-HIN's accuracy and efficiency. Furthermore, In contrastive module, we could design more sophisticated negative samples for the model to contrast and differentiate, so as to enhance the model to mine the unsupervised information. Additionally, real-life networks are always evolving, so it is also of interest to design an algorithm to deal with the few-shot learning on dynamic HINs.

In this chapter, we have focused on answering **RQ3**, and have proposed a novel model, META-HIN, to address few-shot learning challenges for HINs. META-HIN samples subgraphs while preserving heterogeneous features and splits them into meta-training and meta-testing sets, with each set divided into a support set and query set. The pre-processed datasets are sent into a meta-learning framework to learn the representation of HINs in a few-shot learning setting. In the following chapter, we will turn to how to leverage the textual information that may come with HINs.

# 5 Textual Representation Learning

This chapter is set up to answer **RQ4**:

**RQ4** How to make use of the textual information when learning the representation of heterogeneous information networks (HINs)?

Textual HINs are graphs with abundant text information. Current approaches to mining textual features from HINs require sufficient labeled data, thus they are not applicable to few-shot learning scenarios. We propose a prompt-learning framework P-HIN that provides a new angle to leverage textual information and that fits few-shot problems. The proposed framework P-HIN is composed of a text encoder and a graph encoder, and utilizes contrastive learning to align the graph-text pair. This pre-training operation naturally fits our few-shot learning setting. In experiments, P-HIN consistently and significantly outperforms state-of-the-art alternatives on all real-life datasets.

## 5.1 Introduction

Textual HINs are graphs with textual information, e.g., a title and an abstract of a paper node in an academic network, which can provide fruitful additional information for downstream tasks. Most current work on HINs ignores such textual information and maps the node of a graph into low-dimensional representations based only on structural information.

To fill this gap, some models for mining HINs propose to integrate textual information into the node representations. For example, TADW [111] adopts a matrix factorization framework to incorporate structural information and textual information. CANE [99] applies a mutual attention mechanism to learn text-aware node embeddings. NEIFA [110] adopts a deep neural architecture to integrate structural and textual information into a single representation.

### Current limitations

The textual network embedding models mentioned above face a number of limitations. First, they are all designed for a supervised scenario, which requires abundant labeled

This chapter was submitted as .

data for training. In other words, they are not applicable to the few-shot learning setting. However, in real-life applications, it is common that only a handful of labels are available, which poses serious challenges to keeping up the performance. Second, these methods are all originally designed for homogeneous networks, with no prior work yet trying to solve few-shot learning issues on textual HINs.

### Our proposal

We aim to address the shortcomings listed above, and propose

(1) a text encoder to leverage textual information,

(2) a graph encoder that encodes structural and heterogeneous features, along with the self-supervised information,

(3) a contrastive learning mechanism to align the text and graph embeddings, and

(4) a learnable continuous prompt learning framework to solve the few-shot problem on textual HINs.

We refer to the above prompt learning framework for few-shot learning problems on textual HINs as P-HIN. Our work is the *first* to apply prompt learning to integrate graph data with text data, and benefit downstream tasks.

P-HIN is composed of a text encoder and a graph encoder, which encode text and graphs into low-dimensional vectors, respectively. We adopt Sentence-Bert (SBERT) [82] as the text encoder to produce the textual embeddings. For our graph encoder, we first sample subgraphs to be processed and choose a random walks with restart (RWR) sampling strategy. We also force it to sample all types of nodes to ensure heterogeneity. Then we apply an autoencoder [103] mechanism to explore structural features, based on the intuition that nodes with a similar structure will share similar embeddings. After this, to capture the heterogeneity of a graph, we first group the nodes based on their types. This operation may corrupt the structure of the subgraph, but we have already preserved the structural information via an autoencoder. Then we apply a Bi-LSTM on each group to explore the interactions between nodes. We also introduce two graph pre-training tasks, i.e., masked node modeling and edge reconstruction, to leverage the node-level and edge-level self-supervised information.

To align the text embedding and node embedding, we introduce a contrastive learning framework to pre-train the model. Specifically, given a pair of text and subgraph, if they both belong to the given node, they are matched. The contrastive learning framework is meant to maximize the similarity score of a matched pair and minimize the sore of an unmatched pair. To facilitate the training process, we introduce high-quality negative samples to replace the random samples. The negative samples of texts and subgraphs are chosen from other nodes having the same label as the given node.

Through the above pre-training process, P-HIN is able to transfer to downstream tasks, while fitting in a few-shot or even zero-shot setting. For each new classification task, the classification weights can be generated by feeding natural language statements describing classes of interest to the text encoder, and comparing them with the structural and heterogeneous features generated by the graph encoder.

During the optimization phase, we need to carefully design the prompts; this is crucial to the downstream tasks. A slight change of words in the prompts may influence the model performance [124]. Instead of designing handcrafted prompts like “a paper of [CLASS] domain”, we introduce learnable and continuous prompts that are generated automatically. Specifically, a prompt’s text is designed as vectors having the same dimension as the word embeddings, and it is trained using cross-entropy loss to minimize the prediction error. Our automated prompt engineering leads to more task-relevant and efficient transfer for our pre-trained model.

To leverage the contextual nodes information in a graph to prompt the language model, we introduce the residual connection between the text and graph encoder.

### Experiments

To demonstrate the effectiveness of P-HIN, we consider three real-life datasets. OAG is a bibliographic network with papers to be classified into research domains. YELP is a venue check-in dataset with restaurants to be categorized into different types. These two datasets only have 5 labels. We also consider Reddit, a forum dataset with 41 labels, to verify that P-HIN can be applied in many practical scenarios. We choose three downstream tasks, i.e., node classification, link prediction, and keyword generation. Keyword generation is the task that cannot be realized by traditional textual network embedding methods. P-HIN is able to generate keywords thanks to the autoregressive generation abilities under the prompt-tuning framework. We find that P-HIN consistently and significantly outperforms the state-of-the-art models on every dataset that we consider on all downstream tasks.

### Contributions

Our main contributions are as follows:

- We propose a prompt learning framework P-HIN to leverage textual information in textual heterogeneous information networks (HINs), and deal with few-shot leaning problems simultaneously.

- We introduce a graph encoder that can capture structural and heterogeneous features of HINs, meanwhile preserving the node-level and edge-level self-supervised information.

- We show that P-HIN significantly and consistently outperforms state-of-the-art alternatives on different textual HINs.

### Organization

Section 5.2 introduces related work and Section 5.3 provides a detailed account of the P-HIN framework. Section 5.4 presents the experimental setup and analyses the experimental results. Section 5.5 concludes the chapter.

## 5.2 Related work

### Textual network embedding

Many graphs have rich textual information [111], which raises the question of computing textual network embeddings [99]. TADW [111] incorporates textual features into the representation using a matrix factorization framework. CENE [92] treats texts as nodes to integrate textual and structural information. CANE [99] learns a text-aware node embedding through a mutual attention mechanism that models the semantics of nodes. WANE [88] incorporates textual features into node representations by matching important words between text sequences for all pairs of nodes. NEIFA [110] proposes a deep neural architecture to effectively fuse structural information and textual information into a single representation. Finally, DetGP [10] proposes a Gaussian Process to dynamically model the structural and textual information. During recent two years, no textual network embedding methods have been published in top venues.

Integrating text and graph data could be regarded as a multi-modal problem. Multi-modal prompt learning models [80, 123] focus on aligning the image and text under the contrastive learning framework. P-HIN is the first work to apply prompt technique on both graph and text data.

Sun et al. [91] applies prompt learning on graph data by adjusting the downstream node classification task to look similar as its pre-training task. It cannot utilize text information and ignores the heterogeneity of a graph.

### Prompt learning

Given close-style prompts, prompt learning is used to induce pre-trained language models to generate answers, which can facilitate many downstream tasks. The design of prompts plays a vital role in downstream tasks. Jiang et al. [52] adopt text mining and paraphrasing to form candidate prompts and choose the best ones based on the training accuracy. However, discrete prompts need to be designed by expertise and the prompts are by no means guaranteed to be optimal. Therefore, a line of work [58, 62, 121] adopts learnable continuous prompts, all of which optimize the prompt vectors in word embedding space. The technique is widely used in natural language processing. See [65] for a comprehensive survey.

## 5.3 The proposed model P-HIN

### 5.3.1 Text encoder

P-HIN is composed of two encoders, i.e., a text encoder and a graph encoder. The text encoder maps natural language text to a low-dimensional representation. We use the Sentence-BERT (SBERT) [82] model to generate the fixed-size text embeddings.

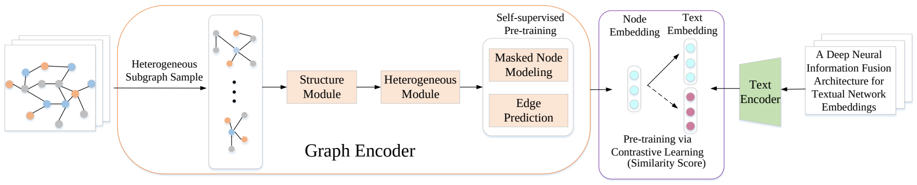

Heterogeneous Subgraph Sample
Structure Module
Heterogeneous Module
Self-supervised Pre-training
Masked Node Modeling
Edge Prediction
Graph Encoder
Node Embedding
Text Embedding
Pre-training via Contrastive Learning (Similarity Score)
Text Encoder
A Deep Neural Information Fusion Architecture for Textual Network Embeddings

Figure 5.1: The training framework of P-HIN.

### 5.3.2 Graph encoder

The graph encoder maps graph data into a low-dimensional embedding. For a given node, we need to first sample the subgraphs around it, then the subgraph is processed by the graph encoder to generate the node representation. After sampling the subgraphs, the nodes within a subgraph will be ranked via centrality metrics which evaluate a node's importance [5]. The rank of each node will be used as position information in the following pipeline.

The sampling strategy we use is random walks with restart (RWR). It will iteratively travel the neighborhood of a given node $v$, with some probability traveling back to the starting node $v$. To sample nodes with higher importance, we let the RWR reach out to high-rank nodes first. To equip the encoder with a sense of heterogeneity, we constrain the walk to sample all types of nodes.

We first adopt an autoencoder [103] to capture the structural information of the subgraph. Given the adjacency matrix $A$ of a subgraph, it will first be processed by an encoder to generate the latent embeddings by several layers. Then, the decoder reverses the above process to obtain the reconstructed output $\hat{A}$. The autoencoder aims to minimize the reconstruction error of the input and output, so that nodes with similar structure will have similar embeddings. Mathematically,

$$\mathcal{L}_{structure} = \mid (\hat{A} - A) \odot B \mid_F^2, \tag{5.1}$$

in which $B$ is the penalty assigned on none-zero elements to alleviate the sparsity issue; $\odot$ is element-wise product.

To explore the heterogeneous features of a graph, nodes with the same type are first grouped together. This operation may corrupt the structure of the subgraph, however, the autoencoder we have adopted before has already preserved the structural features. Then we apply a Bi-LSTM on each group to model the type-specific features. A Bi-LSTM is able to capture the interactions of node features and has broad expressive capability. Given a node group $S^v_{T_j}$ with type $T_j$, the representation $h^v_{T_j}$ of node $v$ can be calculated as

$$h^v_{T_j} = \frac{\sum^{S^v_{T_j}} \text{Bi-LSTM}\{v\}}{|S^v_{T_j}|}. \tag{5.2}$$

Then we apply an attention mechanism to aggregate all type groups to generate the representation of the given node $h_v$.

$$h_v = \sum_{T_j \in \{T\}} \alpha^{v,j} h^v_{T_j}, \tag{5.3}$$

$$\alpha^{v,j} = \frac{\exp\{\delta(u^{\mathrm{T}} h^v_{T_j})\}}{\sum_{T_i \in \{T\}} \exp\{\delta(u^{\mathrm{T}} h^v_{T_i})\}}, \tag{5.4}$$

where $\delta$ denotes the activation function, for which we use a *LeakyReLU*; $u \in \mathbb{R}^d$ is the attention parameter.

In addition, we further apply the transformer encoder to pre-train the subgraphs based on the self-supervised information. In specific, we introduce two pre-training tasks, i.e., masked node modeling (MNM) and edge reconstruction (ER), which realize node-level and edge-level exploration of the graph.

For masked node modeling, we sort the nodes in the subgraph based on their rank and we randomly choose 15% of the nodes with [MASK] token. The sorted sequential nodes will be fed into a Transformer encoder, in which the embeddings produced by Bi-LSTM will be taken as the token embeddings and the rank information is taken as the position embedding. The learned hidden state $h_v^{Trans}$ after the transformer encoder will be fed into a feedforward layer to predict the target node. Mathematically,

$$z_v = \mathrm{Feedforward}(h_v^{Trans}), \tag{5.5}$$

$$\mathbf{p_v} = \mathrm{softmax}(\mathbf{W}^{\mathrm{MNM}} z_v), \tag{5.6}$$

where $z_v$ is the output of the feedforward layer, $\mathbf{W}^{\mathrm{MNM}} \in V_v \times d$ is the classification weight shared with the input node embedding matrix, $V_v$ is the number of nodes in the subgraph, $d$ is the dimension of the hidden state size, $\mathbf{p_v}$ is the predicted distribution of $v$ over all nodes. We use the cross-entropy between the one-hot label $\mathbf{y}_i^t$ and the prediction $\mathbf{p}_i^t$:

$$\mathcal{L}_{\mathrm{MNM}} = -\sum_i y_i \log p_i, \tag{5.7}$$

where $y_i$ and $p_i$ are the $i$-th components of $\mathbf{y}_i$ and $\mathbf{p}_i$, respectively.

For edge reconstruction, we sample the positive and negative edges in a subgraph. Positive edges do exist in the original subgraph while the negative edges do not exist in the original subgraph. Given the positive edges and negative edges united as $E_S$, in practice, we set $|E_S| = 6$ with positive edges and negative edges split equally. We calculate the score of edge reconstruction via the inner product between the pair of nodes, i.e., $\hat{e}_{uv} = h_v \odot h_u$. To compute the edge reconstruction loss, we adopt the binary cross-entropy loss between the predicted edges and ground-truth edges:

$$\mathcal{L}_{\mathrm{ER}} = -\frac{1}{|E_S|} \sum_{e_{uv} \in |E_S|} \mathrm{BinaryCrossEntropy}(e_{uv}, \hat{e}_{uv}). \tag{5.8}$$

We combine the MNM and ER loss with equal weights when training.

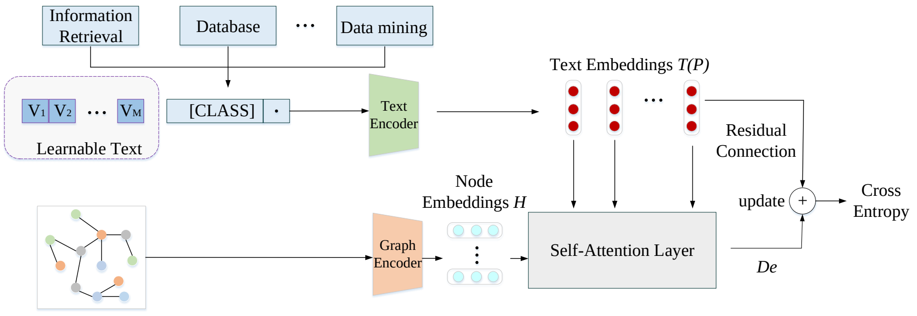


Figure 5.2: The optimization framework of P-HIN.

### 5.3.3 Training

P-HIN is trained to align the representation spaces of text and graphs, while the learning objective is designed as a contrastive loss. Specifically, given a batch of text-graph pairs, P-HIN needs to maximize the similarity scores for matched pairs, while minimizing the scores for unmatched pairs. We adopt cosine similarity. In a contrastive learning setting, high-quality negative samples are helpful to boost the model performance. The text and subgraphs we use in a batch are chosen from nodes with the same label to make them hard to be distinguished. The training framework is illustrated in Figure 5.1.

### 5.3.4 Few-shot inference

P-HIN can be applied in few-shot or even zero-shot settings as it is pre-trained to predict whether a node's subgraph matches a textual description. This can be realized by comparing the node embedding generated by the graph encoder with the classification weights produced by the text encoder. The textual description can be used to specify node classes of interest. Given a node $v$, its node embedding learned by the graph encoder is denoted as $H$, and the weight vectors generated by the text encoder is denoted as $\{w_i\}_{i=1}^{K}$, where $K$ denotes the number of classes. Each weight $w_i$ is learned from the prompt, e.g., "a paper of [CLASS] domain", and the class token can be the specific class name, such as "information retrieval", "database" or "data mining". To facilitate the link prediction downstream task, the prompt could also be designed as "The two nodes are [CLASS] ", and it is a binary class token like "connected" and "unconnected". Mathematically, the prediction probability can be calculated as

$$p(y = i|v) = \frac{\exp(\langle w_i, H\rangle/\tau)}{\sum_{j=1}^{K}\exp(\langle w_j, H\rangle/\tau)}, \tag{5.9}$$

in which $\tau$ is a temperature parameter learned by P-HIN, and $\langle\cdot,\cdot\rangle$ denotes the similarity score.

### 5.3.5 Optimization

Instead of manual prompts that should be designed by experts, here we choose continuous vectors to replace the text words which could be learned end-to-end from data. Figure 5.2 illustrates the optimization framework. Specifically, the prompt $P$ sent to the text encoder should be devised as

$$P = [V_1][V_2] \dots [V_M][CLASS], \tag{5.10}$$

where $[V_M]$ is a vector with the same dimension as the word embeddings in the training phase, and $M$ is a hyper-parameter that denotes the number of the continuous vectors in the prompt. After sending the continuous prompt $P$ to the text encoder $Text(\cdot)$, the classification weight vector representing a node's concept can be obtained. Mathematically, the prediction probability is calculated as

$$p(y = i|v) = \frac{\exp(\langle Text(P_i), H \rangle)/\tau}{\sum_{j=1}^{K} \exp(\langle Text(P_j), H \rangle / \tau)}, \tag{5.11}$$

where the class token within each prompt $P_i$ is replaced by the word embeddings of the $i$-th class name.

To optimize the text vectors, we conduct training to minimize the standard classification loss based on cross-entropy. The gradients can be back-propagated through the text encoder $Text(\cdot)$, in which the rich knowledge encoded in the parameters could be leveraged. Our choice of continuous vectors also enables the full exploration of the word embedding space, which improves the learning of task-relevant text.

### 5.3.6 Residual connection

Considering the context nodes of the given node, e.g., the author nodes of the paper node, will help the text encoder become more accurate. Therefore, to further prompt the pre-trained language model, we leverage the context of the given node's subgraph based on residual connection between text encoder and graph encoder. We first send the text embeddings of class label and node embeddings to the Transformer decoder, using the class information as the queries. Such operation facilitates the text features to find the most relevant contexual nodes of the given node. Having the output of the transformer decoder $De$, we then update the text features via a residual connection as follows

$$Text(P) \leftarrow Text(P) + \lambda De, \tag{5.12}$$

where $\lambda$ is a learnable parameter to control the scaling of the residual. We initialize the $\lambda$ as a small value $10^{-4}$ so that the language priors from the text features could be maximally maintained.

## 5.4 Experiments

We first detail our datasets, baseline models, and parameter settings. Then we introduce the experimental results and analysis.

Table 5.1: Dataset statistics.

| Dataset | #nodes | #edges | #labels |
|---|---|---|---|
| OAG | 432,362 | 1,837,362 | 5 |
| YELP | 382,472 | 2,412,428 | 5 |
| Reddit | 232,965 | 11,606,919 | 42 |

### 5.4.1 Experimental setup

#### Datasets

We consider three real-world datasets, i.e., OAG,[1] YELP,[2] and Reddit.[3] OAG is a bibliographic graph network having four types of nodes. We choose the title and abstract as texts, and classify the corresponding paper node into five types: (i) information retrieval, (ii) database, (iii) natural language processing, (iv) data mining, and (v) machine learning. YELP is a venue check-in network and has four types of nodes. The textual descriptions are the reviews of the restaurant, while the restaurants are divided into five types: (i) Chinese food, (ii) Indian food, (iii) French food, (iv) fast food, and (v) sushi bar. Reddit is a dataset extracted from the online forum Reddit; textual descriptions are comments of posts with the posts classified into different communities. OAG and YELP only have 5 labels for few-shot classification, while Reddit has 42 labels, which could verify that P-HIN could fit into different practical scenarios.

We split the datasets into 80% training datasets, 10% validation datasets and 10% testing datasets. Table 5.1 summarizes the information about the above datasets.

#### Baselines

We choose several baselines specifically designed for textual graphs, that is, TADW [111], CENE [92], CANE [99], WANE [88], NEIFA [110] and DetGP [10]. Further details of these models have been introduced in Section 5.2. We also include GPPT [91] which applies prompt technique on graph data while ignoring text information. GPPT is only applicable to node classification task. Details of these models have been introduced in Section 5.2.

#### Parameters

The latent dimensions of all representations are fixed to 512. For the text encoder, the vocabulary size is 49,152 and each text sequence is fixed to 77 with the [SOS] and [EOS] encompassed. The text vectors in optimization are initialized by a zero-mean Gaussian distribution with standard deviation equal to 0.02. The number of text tokens is set to 8. We adopt SGD for training with the initial learning rate of 0.002; it is decayed using the cosine annealing rule. The maximum epoch is set to 200. To mitigate explosive gradients observed in the early training iterations, we use the warmup trick by

[1]`https://www.openacademic.ai/oag`
[2]`https://www.yelp.com/dataset_challenge`
[3]`https://www.reddit.com`

Table 5.2: Results on the node classification task.

| | OAG | | YELP | | Reddit | |
|---|---|---|---|---|---|---|
| Model | ACC | MAC-F1 | ACC | MAC-F1 | ACC | MAC-F1 |
| TADW | 0.541 | 0.638 | 0.521 | 0.576 | 0.459 | 0.523 |
| CENE | 0.554 | 0.644 | 0.534 | 0.584 | 0.468 | 0.539 |
| CANE | 0.565 | 0.653 | 0.547 | 0.601 | 0.475 | 0.554 |
| WANE | 0.584 | 0.686 | 0.563 | 0.626 | 0.481 | 0.567 |
| NEIFA | 0.604 | 0.706 | 0.574 | 0.658 | 0.489 | 0.579 |
| DetGP | 0.612 | 0.711 | 0.584 | 0.667 | 0.496 | 0.591 |
| GPPT | 0.603 | 0.707 | 0.579 | 0.663 | 0.487 | 0.575 |
| P-HIN | **0.652**$^{\blacktriangle}$ | **0.749**$^{\blacktriangle}$ | **0.624**$^{\blacktriangle}$ | **0.716**$^{\blacktriangle}$ | **0.535**$^{\blacktriangle}$ | **0.644**$^{\blacktriangle}$ |

fixing the learning rate to $1e-5$ during the first epoch. Three labels are used for training in OAG and YELP, with the rest for testing. Data with 31 labels is used for training on the Reddit dataset, with the rest used for testing. For the baselines, we directly adopt the best parameter configurations reported in their original papers. We use 5-shot samples per class for training. Code will be released online prior to the final publication.

We use an Intel(R) Xeon(R) Platinum 8268 CPU and Tesla V100 to run experiments with both the pre-training and downstream tasks.

## 5.4.2 Node classification

We evaluate the performance of P-HIN and the baselines on the node classification task. ACC and Macro-F1 score are adopted as evaluation metrics (averaged over five folds). Table 5.2 presents the experimental results on the node classification task; the highest scores are set in bold. We report on statistical significance with a paired two-tailed t-test and mark a significant improvement of P-HIN over the best baseline DetGP for $p < 0.05$ with $^{\blacktriangle}$.

P-HIN consistently and significantly outperforms the baselines on all datasets, which demonstrates the model's effectiveness. Concretely, all textual network embedding models perform worse than P-HIN, which can be attributed to the fact that they are unable to handle the few-shot problem. GPPT has comparable performance with NEIFA even without text information. This is because GPPT's prompt technique helps to deal with few-shot settings. P-HIN still performs best. We believe it is because that P-HIN provides a new angle to leverage textual information of graph based on prompt learning, meanwhile helping P-HIN to fit in the few-shot setting. Moreover, previous methods are not specifically designed for heterogeneous networks, while P-HIN designs a graph encoder that can handle heterogeneous features.

## 5.4.3 Link prediction

We adjust the fine-tuning stage to enable the model to realize the link prediction task. Specifically, we transfer the link prediction task into a binary classification task, with the label telling if a node pair in a subgraph is connected or not. We adopt the AUC

Table 5.3: Results on the link prediction task.

| | OAG | | YELP | | Reddit | |
|---|---|---|---|---|---|---|
| Model | AUC | MAC-F1 | AUC | MAC-F1 | AUC | MAC-F1 |
| TADW | 0.724 | 0.635 | 0.773 | 0.642 | 0.791 | 0.704 |
| CENE | 0.745 | 0.641 | 0.794 | 0.664 | 0.805 | 0.736 |
| CANE | 0.773 | 0.667 | 0.810 | 0.685 | 0.823 | 0.753 |
| WANE | 0.792 | 0.683 | 0.833 | 0.694 | 0.845 | 0.768 |
| NEIFA | 0.826 | 0.694 | 0.856 | 0.743 | 0.866 | 0.783 |
| DetGP | 0.843 | 0.732 | 0.877 | 0.788 | 0.871 | 0.793 |
| P-HIN | **0.897**$^\blacktriangle$ | **0.792**$^\blacktriangle$ | **0.912**$^\blacktriangle$ | **0.823**$^\blacktriangle$ | **0.901**$^\blacktriangle$ | **0.837**$^\blacktriangle$ |

Table 5.4: Results on the keyword generation task.

| | OAG | |
|---|---|---|
| Model | F1@1 | F1@3 |
| KeyBERT | 0.287 | 0.344 |
| P-HIN | **0.342**$^\blacktriangle$ | **0.401**$^\blacktriangle$ |

value and Macro-F1 value as evaluation metrics (five-fold average).

Table 5.3 presents the link prediction results; the highest scores are set in bold. P-HIN still performs the best among all the baselines. This can be attributed to two main reasons. One is that we adopt the edge reconstruction pre-training task which already explores the edge-level structure, boosting the link prediction performance. The other is similar to what we illustrated in Section 5.4.2: the prompt learning setting helps to effectively mine the text features, leading to a better performance on downstream tasks.

### 5.4.4 Keyword generation

Here we introduce a task that has not been considered in the graph evaluation domain, i.e., keyword generation. P-HIN is able to generate keywords because of its autoregressive generation abilities under the pre-training and fine-tuning framework. In practice, we use the keywords of a paper node in OAG as ground-truth labels. When fine-tuning, we have the text [CLS][MASK], with [CLS] being the first word of the keywords and next work to be predicted in the [MASK] token. This process will stop when [SEP] is returned. The traditional textual network embedding methods are not able to generate keywords. So to enable a comparison, we use KeyBERT [32], which generates keywords based on BERT-embeddings. It takes the description of each node as input. We adopt the F1 measure on the top 1 and top 3 keywords predictions as our evaluation metric.

Table 5.4 presents the results for the keyword generation task. P-HIN outperforms KeyBERT as P-HIN uses both textual information and subgraph information, while KeyBERT is only able to leverage the textual information.

Table 5.5: Results of our ablation study on the node classification task.

| | OAG | | YELP | | Reddit | |
|---|---|---|---|---|---|---|
| Model | ACC | MAC-F1 | ACC | MAC-F1 | ACC | MAC-F1 |
| P-HIN-Manual | 0.631 | 0.724 | 0.602 | 0.685 | 0.512 | 0.616 |
| P-HIN-Random | 0.635 | 0.728 | 0.610 | 0.692 | 0.515 | 0.621 |
| P-HIN\MNM | 0.618 | 0.704 | 0.601 | 0.681 | 0.502 | 0.601 |
| P-HIN\ER | 0.625 | 0.711 | 0.608 | 0.688 | 0.506 | 0.611 |
| P-HIN\Self | 0.603 | 0.692 | 0.588 | 0.667 | 0.491 | 0.586 |
| P-HIN\Residual | 0.627 | 0.725 | 0.603 | 0.688 | 0.511 | 0.614 |
| P-HIN | **0.641**▲ | **0.738**▲ | **0.615**▲ | **0.704**▲ | **0.522**▲ | **0.632**▲ |

### 5.4.5 Ablation study

To evaluate the effectiveness of each component or strategy the model uses, we perform an ablation study. It is conducted on the node classification task. To evaluate the continuous vectors we adopt during prompt tuning, we introduce a variant denoted as P-HIN-Manual that manually creates the prompts. The prompts are designed as “a paper of [CLASS] domain” in OAG, “a restaurant of [CLASS] type” in YELP and “a post belongs to [CLASS] community” in Reddit. During text-subgraph matching pre-training, we limit the negative samples to be sampled from the same label so that they will be harder to be distinguished. To evaluate this strategy, we introduce a variant that randomly forms the negative samples, denoted as P-HIN-Random. During pre-training the graph, we introduce two self-supervision tasks, i.e., masked node modeling and edge reconstruction. To evaluate the effectiveness of these tasks, we introduce three variants denoted as P-HIN\MNM, P-HIN\ER and P-HIN\Self. P-HIN\MNM excludes the masked node modeling task, P-HIN\ER excludes the edge reconstruction task and P-HIN\Self excludes both self-supervision tasks. We also adopt the residual connection between the text encoder and graph encoder to assign the given node’s context sense to the text feature. To evaluate this module, we introduce a variant denoted as P-HIN\Residual that will ignore the residual connection.

Table 5.5 presents the results of the ablation study. P-HIN outperforms P-HIN-Manual, which uses manually crafted prompts. Learning continuous prompts is more flexible and caters to the specific task, which helps to boost the model performance.

P-HIN also outperforms P-HIN-Random, which randomly chooses negative samples to be compared. It can be verified that comparing high-quality negative samples could help to train the contrastive learning framework, thus improving the final experimental results.

The self-supervision tasks do help to improve the model performance. When excluding either of the tasks, the performance drops, with P-HIN\Self performing the worst. Notice that P-HIN\MNM performs worse than P-HIN\ER. MNM explores node-level information and ER explores edge-level information. The node classification task is more related to the MNM task.

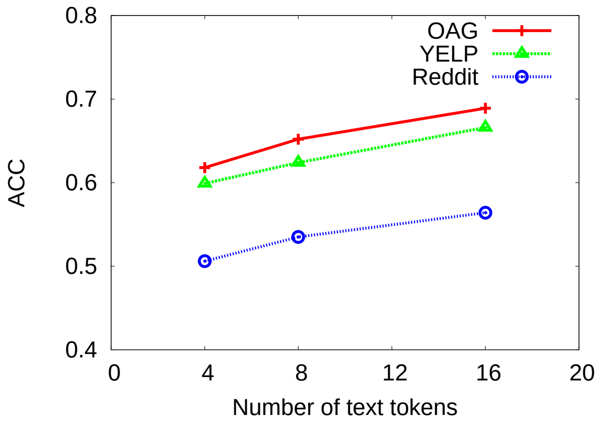


Figure 5.3: Sensitivity w.r.t. number of text tokens.

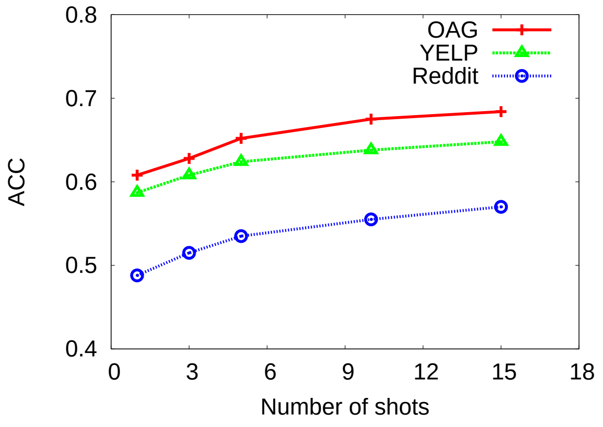


Figure 5.4: Sensitivity w.r.t. number of shots for training.

P-HIN outperforms P-HIN\Residual which confirms that considering the contextual information from the graph to prompt the language model will help to improve the performance.

### 5.4.6 Parameter analysis

Here we conduct a parameter analysis of P-HIN. We choose two parameters to be analyzed, i.e., (i) the number of the text tokens, and (ii) the number of shots used for training. We conduct the parameter analysis on node classification task and adopt ACC as the evaluation metric. Figure 5.3 and Figure 5.4 present the experimental results of the parameter analyses. Figure 5.3 shows the effect of the number of text tokens. As that number increases, the performance improves correspondingly. We set the number to 8 in practice to balance effectiveness and efficiency. One can choose a higher number of text tokens for a better model performance. Figure 5.4 shows that P-HIN obtains a stable performance with varying shots, which indicates that it is a strong few-shot learner and fits multiple real-life few-shot settings.

## 5.5 Conclusion

Textual heterogeneous information networks (HINs) are an essential resource in many real-life scenarios. To answer **RQ4**, we propose a novel prompt learning model, P-HIN, which is able to employ the textual information and handle the few-shot problems simultaneously. It is composed of a text encoder and a graph encoder. The generated textual embeddings and node embeddings are then aligned by a contrastive learning mechanism. We introduce masked node modeling and edge reconstruction tasks to make use of node-level and edge-level self-supervised information. We choose learnable continuous texts instead of manually-designed prompts to facilitate the transfer of the pre-trained model. A residual connection is introduced to leverage the contextual information in graph to prompt the text model.

In our experiments, P-HIN consistently and significantly outperforms state-of-the-art methods on every dataset and on all downstream tasks, which demonstrates the advantages of P-HIN. More broadly, with P-HIN we now have a powerful few-shot learner for textual HINs.

P-HIN has several limitations that we will address in future work. P-HIN is inefficient during pre-training due to the usage of the transformer architecture in both text and graph encoder. The techniques used in variants of BERT to improve the efficiency could be adopted in P-HIN. P-HIN focuses on static textual static HINs; we plan to consider dynamic HINs in future. Finally, we only considered a limited set of downstream tasks; more should be considered in future work.

In this final research chapter, we have focused on answering **RQ4**, and have proposed a novel model P-HIN to make use of textual information of HINs. It first learns graph features and textual features through a graph encoder and text encoder, respectively. Then, the learned embeddings are aligned under a contrastive learning mechanism, so that the graph and text information are integrated. In the next and final chapter of the thesis, we present the conclusion of the thesis and identify directions for future work.

# 6 Conclusions

In this chapter we summarize the thesis by formulating answers to the research questions formulated in Section 1.1. After that we identify some future research directions.

## 6.1 Results

**RQ1** How to learn the representation of dynamic heterogeneous information networks (HINs)?

This question is answered in Chapter 2, where we introduce a dynamic representation model named M-DHIN. We regard a dynamic HIN as a sequence of snapshots with different time stamps. At the initial time stamp, M-DHIN adopts a metagraph-based complex embedding mechanism to learn the initial HIN representation. Then we build a change dataset that records the evolution process of the dynamic HIN. We only train the change set to avoid training the whole network, and in this way we make M-DHIN more scalable. Then we introduce an LSTM-based deep auto-encoder to predict future structure.

In experiments, M-DHIN significantly and consistently outperforms state-of-the-art models on real-life datasets on six downstream tasks. This confirms M-DHIN's ability to capture the features of dynamic HINs. We also conduct an ablation analysis to evaluate the effectiveness of different modules of M-DHIN. And lastly, we conduct a parameter analysis to evaluate the impact of parameters, i.e., the choice of dimension and the negative ratio when sampling.

**RQ2** How to pre-train HINs?

We have answered this question in Chapter 3, where we introduce a self-supervision pre-training and fine-tuning framework named PF-HIN. During a pre-training stage, we first generate input sequences using rank-guided heterogeneous walks, and then group them into mini-sequences based on their types. We design two self-supervised pre-training tasks, i.e., masked node modeling (MNM) and adjacent node prediction (ANP). The MNM task captures the node-level structure while ANP captures edge-level structure. We adopt bi-directional transformer layers to realize the pre-train tasks. During the fine-tuning stage, four downstream tasks are chosen, i.e., link prediction, similarity search, node classification, and node clustering.

PF-HIN outperforms state-of-the-art models on the above tasks on four real-life datasets. The results confirm PF-HIN's ability to pre-train a large-scale unlabeled dataset and then quickly adapt to other datasets with different domains and different downstream tasks. We also conduct an ablation analysis to evaluate the effectiveness of different modules of PF-HIN. And, finally, we conduct a parameter analysis to evaluate the impact of parameters, i.e., the maximum length of the input sequence and the dimension of the node embedding.

**RQ3** How to learn the representation of HINs in a few-shot setting?

Chapter 4 answered the above question. We propose a meta-learning framework based model named META-HIN. META-HIN is applicable to three tasks (i.e., node classification, link prediction, and anomaly detection) across different HINs. A structure module, a heterogeneous GNN module, and a GAN-based contrastive module are part of META-HIN. They allow us to make effective use of structural, heterogeneous, and unsupervised information in a network, respectively. For the link prediction and anomaly detection tasks, we choose a contrastive loss to train the meta-learner. Different tasks have different influences on the meta-learner, so we introduce a self-attention mechanism to assign different weights to different tasks.

In experiments, META-HIN consistently and significantly outperforms state-of-the-art methods on every task across datasets, which demonstrates META-HIN's ability to deal with a few-shot setting. We conduct an ablation analysis to evaluate the effect of different modules and strategies we adopt. The results confirm the effectiveness of META-HIN's design. META-HIN is able to handle both a single-graph scenario and multiple-graph scenarios. In a multi-graph scenario, aside from graphs from the same distribution, graphs from other domains can also be processed. Additionally, META-HIN is able to handle three tasks in a general framework, with each having a specific training loss. We also conduct parameter sensitivity analyses to demonstrate the stability of our model. The parameters chosen are the maximum number of nodes of a subgraph and the number of shots used for meta-learning.

**RQ4** How to make use of the textual information when learning the representation of HINs?

We have answered the above question in Chapter 5, where we propose a novel prompt learning model, P-HIN, which allows one to employ textual information and handle the few-shot problems, simultaneously. It consists of a text encoder and a graph encoder. The generated textual embeddings and node embeddings are then aligned by a contrastive learning mechanism. We introduce masked node modeling and edge reconstruction tasks to make use of node-level and edge-level self-supervised information. Learnable continuous texts instead of manually-designed prompts are adopted to facilitate the transfer of the pre-trained model. We also introduce a residual connection is introduced to leverage the contextual information in graph to prompt the text model.

In experiments, P-HIN consistently and significantly outperforms state-of-the-art alternatives on every dataset and on all downstream tasks, which demonstrates the advantages of P-HIN. The results confirm that P-HIN is a powerful few-shot learner for textual HINs. We then conduct an ablation analysis to evaluate the effectiveness of different modules of P-HIN. We also conduct a parameter analysis to evaluate the impact of parameters, i.e., the number of the text tokens, and the number of shots used for training.

## 6.2 Future work

As with any research, this thesis has some limitations, and we believe that more questions have been generated than answered by our results. In addition to the suggestions for future work that we provided at the end of each of the research chapters, here we provide several general future research directions.

### 6.2.1 Model compression

For training of large-scale HINs, in addition to the cost of training time, the cost of space occupied by the model cannot be ignored. For example, just considering the parameters of the node embedding in the network representation learning model, the 200 dimensional embedding vector of 100 million nodes occupies nearly 200G of memory. If complex heterogeneous information is considered, or more complex neural network-based models are used, the parameter size of the model will be further increased. How can we effectively use the relationship between node representations and the long-tail distribution of nodes to compress the network? And how do such compressions affect the application of network representation learning in practical scenarios.

### 6.2.2 Multi-modal heterogeneous information network attributes

There are various attribute features contained in HINs. In this thesis, we only considered textual attributes as auxiliary information. However, a node may also be connected to other multi-modal attributes, such as images and videos. For example, in a social network, aside from textual introductions, a user node may have a photo as an attribute. Such unstructured information needs image or video encoders to be processed. How to leverage multi-modal attributes to further facilitate the representation learning of HINs remains to be addressed in future.

### 6.2.3 Application-oriented heterogeneous information network representation learning

The training objective of existing network representation learning models generally focuses on the ability to model and reconstruct the network structure, while ignoring the effect of the learned representation vectors in more practical application scenarios, e.g., recommendation, academic search, financial search, product search, and social media search. Therefore, how to combine the network representation learning model with specific application scenarios and improve the corresponding effect of node representations in specific applications remains to be an important challenge.

### 6.2.4 Interpretability of heterogeneous information network representation learning

As HINs incorporate a wealth of information, node representations based on semantic mining methods such as metapaths have strong explanatory properties. For example, a product recommendation engine based on HINs can give clear reasons for its results based on the attention weights of metapaths. This advantage has not been well analyzed or elaborated yet in many tasks. Similarly, we are optimistic that the possibility of misdiagnosis can be reduced if the underlying HIN-based diagnostic reasons are given for disease diagnosis. It is still a challenge how we can best enhance the interpretability of HIN representation learning.

# Bibliography

[1] A. Ahmed, N. Shervashidze, S. M. Narayanamurthy, V. Josifovski, and A. J. Smola. Distributed large-scale natural graph factorization. In *WWW*, pages 37–48, 2013. (Cited on pages 12 and 41.)

[2] L. Akoglu, H. Tong, and D. Koutra. Graph based anomaly detection and description: A survey. *Data Mining and Knowledge Discovery*, 29(3):626–688, 2015. (Cited on page 31.)

[3] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NeurIPS*, pages 585–591, 2001. (Cited on pages 12 and 41.)

[4] R. Bian, Y. S. Koh, G. Dobbie, and A. Divoli. 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 2019, Paris, France, July 21-25, 2019*, pages 861–864, 2019. (Cited on pages 13 and 26.)

[5] P. Bonacich. Power and centrality: A family of measures. *American Journal of Sociology*, 92(5): 1170–1182, 1987. (Cited on pages 44, 68, and 89.)

[6] A. J. Bose, A. Jain, P. Molino, and W. L. Hamilton. Meta-Graph: Few shot link prediction via meta learning. *CoRR*, abs/1912.09867, 2019. (Cited on pages 61, 62, 65, and 75.)

[7] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014. (Cited on pages 42 and 64.)

[8] H. Cai, V. W. Zheng, and K. C. Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Trans. Knowl. Data Eng.*, 30(9):1616–1637, 2018. (Cited on page 1.)

[9] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015*, pages 891–900, 2015. (Cited on page 12.)

[10] P. Cheng, Y. Li, X. Zhang, L. Chen, D. E. Carlson, and L. Carin. Dynamic embedding on textual networks via a gaussian process. In *AAAI, New York, NY, USA, February 7-12, 2020*, pages 7562–7569. AAAI Press, 2020. (Cited on pages 88 and 93.)

[11] M. A. A. Cox and T. F. Cox. *Multidimensional Scaling*. Springer Berlin Heidelberg, 2008. (Cited on pages 12 and 41.)

[12] Y. Cui, H. Sun, Y. Zhao, H. Yin, and K. Zheng. Sequential-knowledge-aware next POI recommendation: A meta-learning approach. *ACM Trans. Inf. Syst.*, 40(2):23:1–23:22, 2022. (Cited on page 65.)

[13] A. M. Dai and Q. V. Le. Semi-supervised sequence learning. In *NeurIPS*, pages 3079–3087, 2015. (Cited on page 39.)

[14] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, pages 3837–3845, 2016. (Cited on pages 42 and 64.)

[15] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186, 2019. (Cited on pages 40, 46, and 47.)

[16] K. Ding, J. Li, and H. Liu. Interactive anomaly detection on attributed networks. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019*, pages 357–365. ACM, 2019. (Cited on page 78.)

[17] K. Ding, Q. Zhou, H. Tong, and H. Liu. Few-shot network anomaly detection via cross-network meta-learning. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 2448–2456. ACM / IW3C2, 2021. (Cited on pages 61, 62, 66, and 75.)

[18] Y. Dong, N. V. Chawla, and A. Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*, pages 135–144, 2017. (Cited on pages 1, 10, 12, 25, 41, and 49.)

[19] Y. Dong, Z. Hu, K. Wang, Y. Sun, and J. Tang. Heterogeneous network representation learning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4861–4867. ijcai.org, 2020. (Cited on page 41.)

[20] M. Elseidy, E. Abdelhamid, and et al. GRAMI: frequent subgraph and pattern mining in a single large graph. *PVLDB*, 7(7):517–528, 2014. (Cited on page 16.)

[21] Y. Fan, Y. Ye, Q. Peng, J. Zhang, Y. Zhang, X. Xiao, C. Shi, Q. Xiong, F. Shao, and L. Zhao. Metagraph aggregated heterogeneous graph neural network for illicit traded product identification in underground market. In *20th IEEE International Conference on Data Mining, ICDM 2020, Sorrento, Italy, November 17-20, 2020*, pages 132–141. IEEE, 2020. (Cited on page 12.)

[22] Y. Fang, X. Zhao, Y. Chen, W. Xiao, and M. de Rijke. PF-HIN: Pre-training for heterogeneous information networks. *IEEE Trans. Knowl. Data Eng.*, 35(8):8372–8385.

[23] Y. Fang, X. Zhao, P. Huang, W. Xiao, and M. de Rijke. 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 2019, Paris, France, July 21-25, 2019*, pages 913–916, 2019. (Cited on page 10.)

[24] Y. Fang, X. Zhao, P. Huang, W. Xiao, and M. de Rijke. Scalable representation learning for dynamic heterogeneous information networks via metagraphs. *ACM Trans. Inf. Syst.*, 40(4):64:1–64:27, 2022.

[25] Y. Fang, X. Zhao, W. Xiao, and M. de Rijke. Few-shot learning for heterogeneous information networks. *ACM Trans. Inf. Syst.*, Under review.

[26] Y. Fang, X. Zhao, W. Xiao, and M. de Rijke. Prompt learning for textual heterogeneous information networks. *IEEE Trans. Knowl. Data Eng.*, Under review.

[27] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 2017. (Cited on pages 62 and 75.)

[28] T. Fu, W. Lee, and Z. Lei. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *CIKM*, pages 1797–1806, 2017. (Cited on pages 1, 10, 12, 26, 41, and 49.)

[29] H. Gao, Z. Wang, and S. Ji. Large-scale learnable graph convolutional networks. In *KDD*, pages 1416–1424, 2018. (Cited on pages 12, 42, and 64.)

[30] P. Goyal, N. Kamra, X. He, and Y. Liu. Dyngem: Deep embedding method for dynamic graphs. *CoRR*, abs/1805.11273, 2018. (Cited on pages 13 and 26.)

[31] P. Goyal, S. R. Chhetri, and A. Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge Based Systems*, 187, 2020. (Cited on pages 13 and 26.)

[32] M. Grootendorst. KeyBERT: Minimal keyword extraction with BERT. `https://doi.org/10.5281/zenodo.4461265`, 2020. (Cited on page 95.)

[33] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864, 2016. (Cited on pages 1, 9, 12, 25, 41, 49, and 50.)

[34] Q. Guo, X. Zhao, Y. Fang, S. Yang, X. Lin, and D. Ouyang. Learning hypersphere for few-shot anomaly detection on attributed networks. In *CIKM 2022: 31st ACM International Conference on Information and Knowledge Management*, pages 635–645. ACM, 2022.

[35] Z. Guo, C. Zhang, W. Yu, J. Herr, O. Wiest, M. Jiang, and N. V. Chawla. Few-shot graph learning for molecular property prediction. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 2559–2567. ACM / IW3C2, 2021. (Cited on pages 61, 62, 66, and 75.)

[36] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, pages 1735–1742. IEEE Computer Society, 2006. (Cited on page 64.)

[37] H. Hafidi, M. Ghogho, P. Ciblat, and A. Swami. GraphCL: Contrastive self-supervised learning of graph representations. *CoRR*, abs/2007.08025, 2020. (Cited on page 65.)

[38] W. L. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1024–1034, 2017. (Cited on pages 42, 49, 64, and 75.)

[39] K. Hassani and A. H. K. Ahmadi. Contrastive multi-view representation learning on graphs. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 4116–4126. PMLR, 2020. (Cited on page 65.)

[40] K. He, H. Fan, Y. Wu, S. Xie, and R. B. Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, pages 9726–9735. IEEE, 2020. (Cited on page 64.)

[41] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *CoRR*, abs/1506.05163, 2015. (Cited on pages 42 and 64.)

[42] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio. Learning deep representations by mutual information estimation and maximization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. (Cited on page 71.)

[43] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. In *ACL*, pages 328–339, 2018. (Cited on page 39.)

[44] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. S. Pande, and J. Leskovec. Strategies for pre-training graph neural networks. In *ICLR*, 2020. (Cited on page 42.)

[45] Z. Hu, C. Fan, T. Chen, K. Chang, and Y. Sun. Pre-training graph neural networks for generic structural feature extraction. *CoRR*, abs/1905.13728, 2019. (Cited on page 43.)

[46] Z. Hu, Y. Dong, K. Wang, K. Chang, and Y. Sun. GPT-GNN: generative pre-training of graph neural

networks. In *KDD*, pages 1857–1867, 2020. (Cited on pages 39, 43, and 50.)

[47] Z. Hu, Y. Dong, K. Wang, and Y. Sun. Heterogeneous graph transformer. In *WWW*, pages 2704–2710, 2020. (Cited on pages 42, 43, 49, and 75.)

[48] K. Huang and M. Zitnik. Graph meta learning via local subgraphs. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. (Cited on pages 61, 65, 67, and 75.)

[49] P. Huang, X. Zhao, M. Hu, Y. Fang, X. Li, and W. Xiao. Extract-select: A span selection framework for nested named entity recognition with generative adversarial training. In *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 85–96. Association for Computational Linguistics, 2022.

[50] Z. Huang and N. Mamoulis. Heterogeneous information network embedding for meta path based proximity. *CoRR*, abs/1701.05291, 2017. (Cited on pages 1, 10, 12, 41, and 49.)

[51] X. Jiang, T. Jia, Y. Fang, C. Shi, Z. Lin, and H. Wang. Pre-training on large-scale heterogeneous graph. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 756–766. ACM, 2021. (Cited on pages 43 and 50.)

[52] Z. Jiang, F. F. Xu, J. Araki, and G. Neubig. How can we know what language models know. *Trans. Assoc. Comput. Linguistics*, 8:423–438, 2020. (Cited on page 88.)

[53] Z. Jiang, J. Gao, and X. Lv. Metap: Meta pattern learning for one-shot knowledge graph completion. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, pages 2232–2236. ACM, 2021. (Cited on page 65.)

[54] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. (Cited on page 20.)

[55] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=SJU4ayYgl`. (Cited on pages 1, 12, 42, 49, 61, 64, and 75.)

[56] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *ICLR*, 2020. (Cited on page 47.)

[57] J. B. Lee, R. A. Rossi, and X. Kong. Graph classification using structural attention. In *KDD*, pages 1666–1674, 2018. (Cited on pages 42 and 64.)

[58] B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 3045–3059. Association for Computational Linguistics, 2021. (Cited on page 88.)

[59] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Trans. Signal Process.*, 67(1):97–109, 2019. (Cited on pages 42 and 64.)

[60] R. Li, S. Wang, F. Zhu, and J. Huang. Adaptive graph convolutional neural networks. In *AAAI*, pages 3546–3553, 2018. (Cited on pages 42 and 64.)

[61] X. Li, D. Ding, B. Kao, Y. Sun, and N. Mamoulis. Leveraging meta-path contexts for classification in heterogeneous information networks. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, pages 912–923. IEEE, 2021. (Cited on page 61.)

[62] X. L. Li and P. Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 4582–4597. Association for Computational Linguistics, 2021. (Cited on page 88.)

[63] Z. Li, F. Nie, X. Chang, Y. Yang, C. Zhang, and N. Sebe. Dynamic affinity graph construction for spectral clustering using multiple features. *IEEE Trans. Neural Networks Learn. Syst.*, 29(12): 6323–6332, 2018. (Cited on page 13.)

[64] Z. Li, W. Zheng, X. Lin, Z. Zhao, Z. Wang, Y. Wang, X. Jian, L. Chen, Q. Yan, and T. Mao. Transn: Heterogeneous network representation learning by translating node embeddings. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 589–600. IEEE, 2020. (Cited on page 61.)

[65] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *CoRR*, 2021. (Cited on

page 88.)

[66] Z. Liu, C. Chen, X. Yang, J. Zhou, X. Li, and L. Song. Heterogeneous graph neural networks for malicious account detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*, pages 2077–2085. ACM, 2018. (Cited on page 12.)

[67] Z. Liu, Y. Fang, C. Liu, and S. C. H. Hoi. Relative and absolute location embedding for few-shot node classification on graph. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 4267–4275. AAAI Press, 2021. (Cited on page 65.)

[68] M. Luo, F. Nie, X. Chang, Y. Yang, A. G. Hauptmann, and Q. Zheng. Adaptive unsupervised feature selection with structure regularization. *IEEE Trans. Neural Networks Learn. Syst.*, 29(4):944–956, 2018. (Cited on page 13.)

[69] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008. (Cited on page 78.)

[70] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013. (Cited on page 64.)

[71] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *CVPR*, pages 5425–5434, 2017. (Cited on pages 42 and 64.)

[72] S. Mu, Y. Li, W. X. Zhao, S. Li, and J. Wen. Knowledge-guided disentangled representation learning for recommender systems. *ACM Trans. Inf. Syst.*, 40(1):6:1–6:26, 2022. (Cited on page 65.)

[73] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, pages 2014–2023, 2016. (Cited on pages 42 and 64.)

[74] G. Niu, Y. Li, C. Tang, R. Geng, J. Dai, Q. Liu, H. Wang, J. Sun, F. Huang, and L. Si. Relational learning with gated and attentive neighbor aggregator for few-shot knowledge graph completion. In *SIGIR '21*, pages 213–222. ACM, 2021. (Cited on page 65.)

[75] S. Nowozin, B. Cseke, and R. Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 271–279, 2016. (Cited on page 71.)

[76] Z. Peng, W. Huang, M. Luo, Q. Zheng, Y. Rong, T. Xu, and J. Huang. Graph representation learning via graphical mutual information maximization. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pages 259–270. ACM / IW3C2, 2020. (Cited on page 65.)

[77] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: online learning of social representations. In *KDD*, pages 701–710, 2014. (Cited on pages 1, 9, 12, 25, 41, and 49.)

[78] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *NAACL-HLT*, pages 2227–2237, 2018. (Cited on page 39.)

[79] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang. GCC: graph contrastive coding for graph neural network pre-training. In *KDD*, pages 1150–1160, 2020. (Cited on pages 39, 43, 50, and 65.)

[80] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 2021. (Cited on page 88.)

[81] M. Rahman, T. K. Saha, M. A. Hasan, K. S. Xu, and C. K. Reddy. Dylink2vec: Effective feature representation for link prediction in dynamic networks. *CoRR*, 2018. (Cited on page 13.)

[82] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3980–3990. Association for Computational Linguistics, 2019. (Cited on pages 86 and 88.)

[83] R. Reinanda, E. Meij, and M. de Rijke. Knowledge graphs: An information retrieval perspective. *Foundations and Trends in Information Retrieval*, 14(4):289–444, October 2020. (Cited on page 1.)

[84] T. Rooker. Review of neurocomputing: Foundations of research. *AI Mag.*, 10(4):64–66, 1989. (Cited on page 20.)

[85] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000. (Cited on pages 12 and 41.)

[86] A. Sankar, X. Zhang, and K. C. Chang. Meta-gnn: metagraph neural network for semi-supervised

learning in attributed heterogeneous information networks. In *ASONAM '19: International Conference on Advances in Social Networks Analysis and Mining, Vancouver, British Columbia, Canada, 27-30 August, 2019*, pages 137–144. ACM, 2019. (Cited on page 12.)

[87] J. Shang, M. Qu, J. Liu, L. M. Kaplan, J. Han, and J. Peng. Meta-path guided embedding for similarity search in large-scale heterogeneous information networks. *CoRR*, 2016. (Cited on page 12.)

[88] D. Shen, X. Zhang, R. Henao, and L. Carin. Improved semantic-aware network embedding with fine-grained word alignment. In *EMNLP, Brussels, Belgium, October 31 - November 4, 2018*, pages 1829–1838. Association for Computational Linguistics, 2018. (Cited on pages 88 and 93.)

[89] J. Snell, K. Swersky, and R. S. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4077–4087, 2017. (Cited on page 75.)

[90] F. Sun, J. Hoffmann, V. Verma, and J. Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *ICLR*, 2020. (Cited on pages 42 and 65.)

[91] M. Sun, K. Zhou, X. He, Y. Wang, and X. Wang. GPPT: graph pre-training and prompt tuning to generalize graph neural networks. In A. Zhang and H. Rangwala, editors, *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, pages 1717–1727. ACM, 2022. doi: 10.1145/3534678.3539249. URL `https://doi.org/10.1145/3534678.3539249`. (Cited on pages 88 and 93.)

[92] X. Sun, J. Guo, X. Ding, and T. Liu. A general framework for content-enhanced network representation learning. *CoRR*, abs/1610.02906, 2016. (Cited on pages 88 and 93.)

[93] Y. Sun and J. Han. *Mining Heterogeneous Information Networks: Principles and Methodologies*. Synthesis Lectures on Data Mining and Knowledge Discovery. Morgan & Claypool Publishers, 2012. (Cited on page 1.)

[94] J. Tang, M. Qu, and Q. Mei. PTE: predictive text embedding through large-scale heterogeneous text networks. In *KDD*, pages 1165–1174, 2015. (Cited on pages 12 and 41.)

[95] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. LINE: large-scale information network embedding. In *WWW*, pages 1067–1077, 2015. (Cited on pages 1, 9, 12, 41, and 49.)

[96] Y. Tian, D. Krishnan, and P. Isola. Contrastive multiview coding. In *ECCV*, volume 12356 of *Lecture Notes in Computer Science*, pages 776–794. Springer, 2020. (Cited on page 64.)

[97] E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, U. Evci, K. Xu, R. Goroshin, C. Gelada, K. Swersky, P. Manzagol, and H. Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. In *ICLR*, 2020. (Cited on page 75.)

[98] T. Trouillon, C. R. Dance, and et al. Knowledge graph completion via complex tensor factorization. *JMLR*, 18:1–38, 2017. (Cited on pages 10 and 15.)

[99] C. Tu, H. Liu, Z. Liu, and M. Sun. CANE: context-aware network embedding for relation modeling. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1722–1731, 2017. (Cited on pages 85, 88, and 93.)

[100] A. van den Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748, 2018. (Cited on pages 65 and 72.)

[101] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *ICLR*, 2018. (Cited on pages 1, 42, 49, and 64.)

[102] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm. Deep graph infomax. In *ICLR*, 2019. (Cited on page 65.)

[103] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *KDD*, pages 1225–1234, 2016. (Cited on pages 12, 41, 86, and 89.)

[104] N. Wang, M. Luo, K. Ding, L. Zhang, J. Li, and Q. Zheng. Graph few-shot learning with attribute matching. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 1545–1554. ACM, 2020. (Cited on page 65.)

[105] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu. Heterogeneous graph attention network. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 2022–2032. ACM, 2019. (Cited on pages 12 and 75.)

[106] X. Wang, N. Liu, H. Han, and C. Shi. Self-supervised heterogeneous graph neural network with co-contrastive learning. In *KDD '21*, pages 1726–1736. ACM, 2021. (Cited on page 42.)

[107] D. J. Watts and S. H. Strogatz. Collective dynamics of —[lsquo]—small-world—[rsquo]— networks.

*Nature*, 393:440–442, 1998. (Cited on page 10.)

[108] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin. Unsupervised feature learning via non-parametric instance discrimination. In *CVPR*, pages 3733–3742. IEEE Computer Society, 2018. (Cited on page 65.)

[109] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *ICLR*, 2019. (Cited on pages 42, 49, and 64.)

[110] Z. Xu, Q. Su, X. Quan, and W. Zhang. A deep neural information fusion architecture for textual network embeddings. In *EMNLP-IJCNLP, Hong Kong, China, November 3-7, 2019*, pages 4697–4705. Association for Computational Linguistics, 2019. (Cited on pages 85, 88, and 93.)

[111] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang. Network representation learning with rich text information. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2111–2117, 2015. (Cited on pages 85, 88, and 93.)

[112] C. Yang, Y. Xiao, Y. Zhang, Y. Sun, and J. Han. Heterogeneous network representation learning: Survey, benchmark, evaluation, and beyond. *CoRR*, 2020. (Cited on page 41.)

[113] H. Yao, X. Wu, Z. Tao, Y. Li, B. Ding, R. Li, and Z. Li. Automated relational meta-learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020. (Cited on page 61.)

[114] H. Yao, C. Zhang, Y. Wei, M. Jiang, S. Wang, J. Huang, N. V. Chawla, and Z. Li. Graph few-shot learning via knowledge transfer. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 6656–6663. AAAI Press, 2020. (Cited on page 66.)

[115] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 793–803. ACM, 2019. (Cited on pages 1, 12, 40, 42, 44, 49, and 75.)

[116] D. Zhang, J. Yin, and et al. Metagraph2vec: Complex semantic path augmented heterogeneous network embedding. In *PAKDD*, pages 196–208, 2018. (Cited on page 12.)

[117] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D. Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. In *UAI*, pages 339–349, 2018. (Cited on pages 42 and 64.)

[118] M. Zhang and Y. Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 5171–5181, 2018. (Cited on page 76.)

[119] Y. Zhang, Q. Yao, W. Dai, and L. Chen. Autosf: Searching scoring functions for knowledge graph embedding. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 433–444. IEEE, 2020. (Cited on page 61.)

[120] Z. Zhang, P. Cui, J. Pei, X. Wang, and W. Zhu. TIMERS: error-bounded SVD restart on dynamic networks. In *Proceedings of the Thirty-Second 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), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 224–231, 2018. (Cited on page 13.)

[121] Z. Zhong, D. Friedman, and D. Chen. Factual probing is [MASK]: learning vs. learning to recall. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 5017–5033. Association for Computational Linguistics, 2021. (Cited on page 88.)

[122] F. Zhou, C. Cao, K. Zhang, G. Trajcevski, T. Zhong, and J. Geng. Meta-GNN: On few-shot node classification in graph meta-learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, pages 2357–2360. ACM, 2019. (Cited on pages 61, 62, 65, and 75.)

[123] K. Zhou, J. Yang, C. C. Loy, and Z. Liu. Learning to prompt for vision-language models. *CoRR*, abs/2109.01134, 2021. URL `https://arxiv.org/abs/2109.01134`. (Cited on page 88.)

[124] K. Zhou, J. Yang, C. C. Loy, and Z. Liu. Learning to prompt for vision-language models. *Int. J. Comput. Vis.*, 130(9):2337–2348, 2022. (Cited on page 87.)

[125] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang. Dynamic network embedding by modeling triadic closure process. In *AAAI*, pages 571–578, 2018. (Cited on pages 10, 13, 17, 26, and 28.)

[126] L. Zhu, D. Guo, J. Yin, G. V. Steeg, and A. Galstyan. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Trans. Knowl. Data Eng.*, 28(10):2765–2777, 2016.

(Cited on page 13.)

[127] Z. Zhuang, X. Xiang, S. Huang, and D. Wang. Hinfshot: A challenge dataset for few-shot node classification in heterogeneous information network. In *ICMR '21: International Conference on Multimedia Retrieval, Taipei, Taiwan, August 21-24, 2021*, pages 429–436. ACM, 2021. (Cited on page 66.)

# Summary

Heterogeneous information networks are ubiquitous. Social networks, knowledge graphs, and interactions between users and items in search and recommender systems can be modeled as networks with multiple types of nodes and edges. In order to mine the rich information captured by a heterogeneous information network, network representation learning embeds a network into a low-dimensional space. Network representation learning has drawn a significant amount of interest from the research community. However, traditional network representation learning models simply learn network embeddings based on a given network, ignoring various real-life scenarios and limitations.

The work in this thesis provides a series of algorithms that are able to deal with different heterogeneous information network scenarios and different downstream tasks. We first focus on dynamic heterogeneous information networks as real-life networks are always evolving. M-DHIN is proposed as a method to learn dynamic embeddings; it is also able to predict the future network.

After that, we study the pre-training problem in heterogeneous information networks. Since most networks are unlabeled, we propose a pre-training and fine-tuning framework PF-HIN that uses two self-supervised pre-training tasks, i.e., masked node modeling and adjacent node prediction. The pre-trained encoder can quickly be adapted to datasets of different domains and different downstream tasks.

We also investigate the few-shot problem for heterogeneous information networks as in practice, only a handful of nodes are labeled. We propose META-HIN, which uses a meta-learning framework that is able to handle the few-shot learning task in both a single-graph scenario and multiple-graph scenarios.

In the final research chapter, we study how to leverage meaningful textual information that may be contained in a heterogeneous information network. We propose a novel prompt learning framework P-HIN that is able to simultaneously employ the textual information and handle the few-shot problems. We align the text representation and node representation using a contrastive learning mechanism, so that textual information is incorporated in an effective manner.

Finally, we suggest a number of directions for future research. These include model compression to improve the model efficiency, the use of multi-modal HIN attributes, application-oriented HIN representation learning, and interpretability of HIN representation learning.

# Samenvatting

Heterogene informatienetwerken zijn overal te vinden. Sociale netwerken, kennisgrafen, en interacties tussen gebruikers en items in zoek- en aanbevelingssystemen kunnen worden gemodelleerd als netwerken met meerdere soorten knopen en kanten. Om de rijke informatie te ontginnen die is vastgelegd met een heterogeen informatienetwerk, bedden we bij het leren van netwerkrepresentaties een netwerk in in een laagdimensionale ruimte. Het leren van netwerkrepresentaties trekt veel aandacht binnen de onderzoeksgemeenschap. Traditionele modellen voor het leren van netwerkrepresentaties leren netwerkinbeddingen terwijl ze voorbijgaan aan realistische scenario's en beperkingen.

Het werk in dit proefschrift biedt een reeks algoritmen die in staat zijn om om te gaan met verschillende heterogene informatienetwerkscenario's en verschillende *downstream*-taken. We richten ons eerst op dynamische heterogene informatienetwerken, aangezien real-life netwerken altijd in ontwikkeling zijn. We stellen M-DHIN voor als een methode om dynamische inbeddingen te leren; de methode is ook in staat om toekomstige netwerken te voorspellen.

Daarna bestuderen we het *pre-trainings*-probleem in heterogene informatienetwerken. Aangezien de meeste netwerken niet-gelabeld zijn, stellen we een *pre-training* en *fine-tuning* framework PF-HIN voor dat gebruik maakt van twee *pre-trainings*-taken met zelf-supervisie, namelijk het modelleren van gemaskeerde knopen en het voorspellen van aangrenzende knopen. De vooraf getrainde *encoder* kan snel worden aangepast aan datasets van verschillende domeinen en voor verschillende *downstream*-taken.

We onderzoeken ook het *few shot*-probleem voor heterogene informatienetwerken, aangezien in de praktijk slechts een handvol knopen is gelabeld. We stellen META-HIN voor, dat een meta-leerraamwerk gebruikt dat in staat is om leertaken aan de hand van een paar voorbeelden aan te kunnen in het scenario met één enkele graaf en in scenario's met meerdere grafen.

In het laatste onderzoekshoofdstuk bestuderen we hoe we zinvolle tekstuele informatie kunnen gebruiken die zich in een heterogeen informatienetwerk kan bevinden. We stellen een nieuw P-HIN-raamwerk voor voor het leren van zogenaamde *prompt* waarmee tegelijkertijd tekstuele informatie gebruikt kan worden en leerproblemen met weinig voorbeelden kunnen worden opgelost. We stemmen de tekstrepresentatie en representatie van knopen op elkaar af met behulp van een contrastief leermechanisme, zodat tekstuele informatie op een effectieve manier wordt opgenomen.

Tot slot stellen we een aantal richtingen voor toekomstig onderzoek voor, waaronder modelcompressie om de efficiëntie van modellen te verbeteren, het gebruik van multimodale HIN-attributen, toepassingsgericht leren van HIN-representaties, en interpreteerbaarheid van het leren van HIN-representaties.

COMPLEX INFORMATION OFTEN INVOLVES MULTIPLE TYPES OF OBJECTS AND RELATIONS. SUCH INFORMATION CAN BE REPRESENTED VIA HETEROGENEOUS INFORMATION NETWORKS (HINS). IN THIS THESIS, WE EXPLORE DIFFERENT LEARNING MECHANISMS FOR NETWORK REPRESENTATION LEARNING TO FIT INTO DIFFERENT SCENARIOS AND DIFFERENT MACHINE LEARNING TASKS OF HINS. WE FIRST STUDY REPRESENTATION LEARNING OF DYNAMIC HINS, AS REAL-LIFE NETWORKS ARE ALWAYS EVOLVING. THEN, WE PRE-TRAIN A HIN WITHOUT USING LABELED INFORMATION, AND A PRE-TRAINED GNN MODEL CAN EASILY BE ADAPTED TO DIFFERENT DATASETS AND DIFFERENT DOWNSTREAM TASKS. WE ALSO INVESTIGATE FEW-SHOT LEARNING OF HINS, WHERE ONLY A HANDFUL OF LABELS ARE GIVEN. AND FINALLY, WE MAKE USE OF AUXILIARY TEXTUAL INFORMATION TO FURTHER FACILITATE THE LEARNING OF HINS.