UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

Machine learning tasks and representations for heterogeneous information networks

Fang, Y.

Publication date
2023

Link to publication

Citation for published version (APA):
Fang, Y. (2023). *Machine learning tasks and representations for heterogeneous information networks*. [Thesis, fully internal, Universiteit van Amsterdam].

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (https://dare.uva.nl)

Download date:13 Oct 2024

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, HetGNN [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

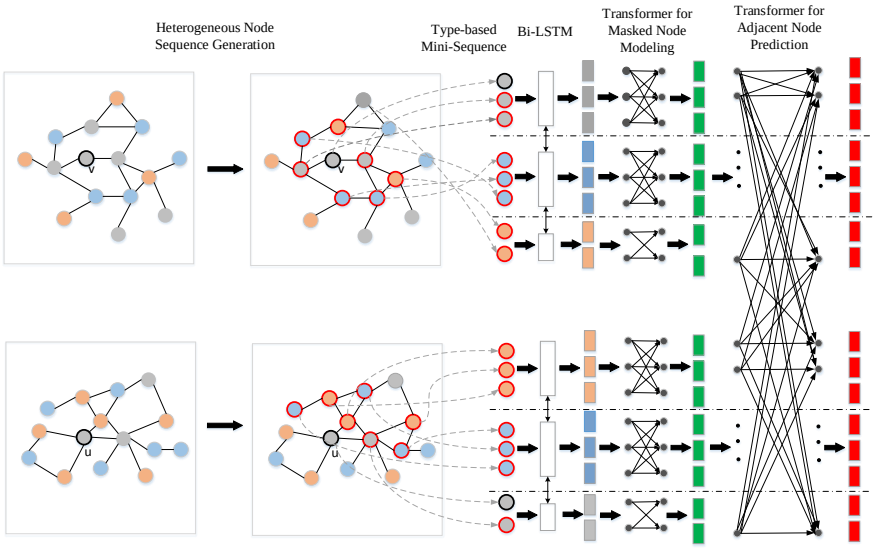


Figure 3.1: The pre-training procedure that makes up PF-HIN. The generation of node sequences is detailed in Section 3.3.1. Bi-LSTM based learning of input embeddings is described in Section 3.3.2. Masked node modeling (MNM) is discussed in Section 3.3.3 and followed by adjacent node prediction (ANP).

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 $\mathbf{p}_i^t$:

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

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 W^{ANP}, the scoring function s_τ of whether the node pair is adjacent is shown as follows:

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

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:

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

where y_τ is the label (positive or negative) of that node pair.

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

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

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 Θ (for downstream tasks);

```
Generates the node sequences via rank guided heterogeneous walk;
for each pair of sampled sequences do
    Apply Bi-LSTM on type-based mini-sequences to learn the input
      embeddings of each node in the sequence;
    for each sequence do
        Mask nodes in the type-based mini-sequences;
        Feed the mini-sequences into transformer layers;
        Calculate the masked node modeling loss by Eq. (3.9);
    end
    Feed the two sequenes into transformer layers;
    Calculate the ajacent node prediction loss by Eq. (3.11);
    Update the parameters Θ by Adam.
end
return Optimized pre-trained model parameters Θ*.
```

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 ▲.

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**▲	**0.613**▲	**0.719**▲

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**▲	**0.318**▲	0.311	**0.330**▲	**0.354**▲	**0.516**▲	**0.509**▲

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**▲	**0.816**▲	**0.927**▲	**0.946**▲

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

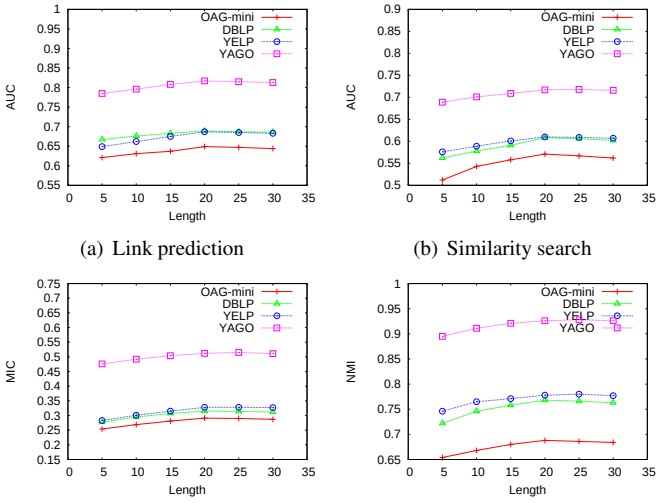
There are two kinds of fine-tuning setting, freezing and full fine-tuning. Freezing fine-tuning is to freeze the parameters of the pre-trained model when fine-tuning, denoted as PF-HIN(Freeze). Full fine-tuning is to train the model with the downstream classifier in an end-to-end manner. PF-HIN leverages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

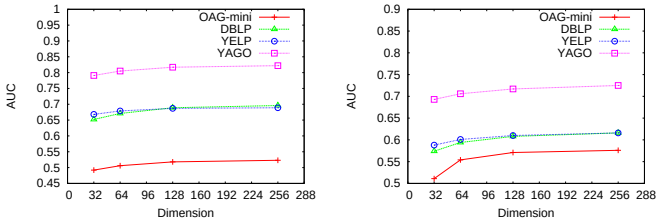
According to this analysis, we choose the length of the input sequence to be 20 so as to balance effectiveness and efficiency.

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.

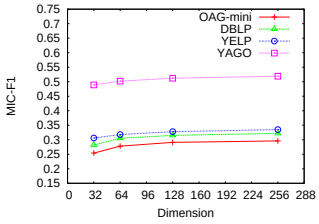
3.6 Conclusions

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

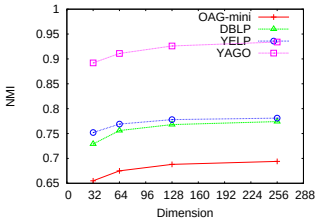


(a) Link prediction

(b) Similarity search



(c) Node classification



(d) Node clustering

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

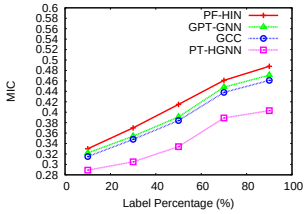


Figure 3.4: Parameter analysis of the percentage of fine-tuning data

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.