PF-HIN: Pre-Training for Heterogeneous Information Networks

Yang Fang, Xiang Zhao, Yifan Chen, Weidong Xiao, and Maarten de Rijke

Abstract—In network representation learning we learn how to represent heterogeneous information networks in a low-dimensional space so as to facilitate effective search, classification, and prediction solutions. 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. 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.

Index Terms—Heterogeneous information network, pre-training, transformer

1 INTRODUCTION

Complex information often involves multiple types of objects and relations. Such information can be represented via heterogeneous information networks (HINs) [1]. In a heterogeneous information network (HIN) different types of nodes (objects) are connected by edges (relations) [2]. Compared to homogeneous networks that only feature a single type of node, HINs provide a richer modeling tool, leading to more effective solutions for search, classification, and prediction tasks [3].

In order to mine the rich information captured by a HIN, network representation learning (NRL) embeds a network into a low-dimensional space. Network representation learning (NRL) has drawn a significant amount of interest from the research community. Classical network embedding models like DeepWalk [4], LINE [5], and node2vec [6] 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 [7], [8], [9]. 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 [7], proximity distance [8], and the Hardmard function [9]. Because of the limited ability of metapaths to capture the neighborhood structure of a node, the performance of these NRL methods is limited.

Recently, graph neural networks (GNNs) have shown promising results on modeling the structure of a network [10], [11], [12]. 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. However, 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.

Inspired by advances in pre-training frameworks in language technology [13], [14], [15], there is a trend to investigate pre-trained models for NRL. In particular, graph contrastive coding (GCC) [16] and GPT-GNN [17] are the most advanced solutions in this stream. Nevertheless, they

1. The experiments on downstream tasks using GPT-GNN [17] were conducted on the same dataset that had been employed for pre-training.
are mainly proposed for 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 5 below).

We aim to overcome the shortcomings listed above, and propose

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

ii) 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 (Pre-training and Fine-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 [12]. 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 [18]. 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.

2 RELATED WORK

2.1 Network Representation Learning

Research on NRL traces back to dimensionality reduction techniques [19], [20], [21], [22], which utilize feature vectors of nodes to construct an affinity graph and then calculate eigenvectors. Graph factorization models [23] 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 [5].

Random walks or paths in a network are being used to help preserve the local and global structure of a network. DeepWalk [4] leverages random walks and applies the Skip-Gram word2vec model to learn network embeddings. node2vec [6] extends DeepWalk; it adopts a biased random walk strategy to explore the network structure. LINE [5] 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 [24] 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 [7] has a heterogeneous SkipGram with its context window restricted to one specific type. HINE [8] uses metapath-based proximity and minimizes the distance between nodes’ joint probability and empirical probabilities. HIN2Vec [9] uses Hadamard multiplication of nodes and metapaths to capture features.
More introduction of heterogeneous representation learning could be found in [25], [26].

Some models employ a self-supervision technique to realize the heterogeneous representation [27], 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).

### 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. [28] propose a spectral graph theory-based graph convolution operation. Graph convolutional networks (GCNs) [10] adopt localized first-order approximations of spectral graph convolutions to improve scalability. There is a line of research to improve spectral GNN models [29], [30], [31], [32], but it processes the whole graph simultaneously, leading to efficiency bottlenecks. To address the problem, spatial GNN models have been proposed [33], [34], [35], [36]. GraphSAGE [36] leverages a sampling strategy to iteratively sample neighboring nodes instead of the whole graph. Gao et al. [35] 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 [11], [37], [38]. The main difference with convolutional-based models is that graph attention networks introduce attention mechanisms to assign higher weights to more important nodes or walks. GAT [11] harnesses masked self-attention layers to apply different weights to different nodes in a neighborhood, to improve efficiency on graph-structured data. GIN [39] 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 [12] 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 [40] 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.

### 2.3 Graph Pre-Training

There exist relatively few approaches for pre-training a GNN model for downstream tasks. InfoGraph [41] maximizes the mutual information between graph-level embeddings and sub-structure embeddings. Hu et al. [42] pre-train a GNN at the level of nodes and graphs to learn local and global features, showing performance improvements on 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. [43] design three pre-training tasks: denoising link reconstruction, centrality score ranking, and cluster preserving. GPT-GNN [17] adopts HGT [40] as its base GNN and uses attribute generation and edge generation as pre-training tasks. Hu et al. [17] only conduct their downstream tasks on the same dataset that was used for pre-training. GCC [16] 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 [44] adopts network schemes 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. THE PROPOSED MODEL PF-HIN

#### 3.1 Preliminaries

A heterogeneous information network (HIN) is a graph $G = (V, E, T)$, where $V$ 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 HIN is a network where $|T_V| > 1$ and/or $|T_E| > 1$.

A visual presentation of the proposed model, pre-training and finetuning heterogeneous information network (PF-HIN), is given in Fig. 1. Below, we describe the node sequence generation procedure, the input representation, followed by the pre-training and fine-tuning stages of PF-HIN.

#### 3.2 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 [45]. This framework makes use of three centrality metrics,\(^2\), i.e.,

- betweenness centrality,
- eigencentrality, and
- closeness centrality.

\(^2\) 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.
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 [12]. 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 5.3.

### 3.3 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:

\[ j_i = \delta(W_{xj}x_i + W_{hj}h_{i-1} + W_{c_j}c_{i-1} + b_j), \]
\[ f_i = \delta(W_{xj}x_i + W_{hj}h_{i-1} + W_{c_j}c_{i-1} + b_j), \]
\[ z_i = \tanh(W_{xj}x_i + W_{hj}h_{i-1} + b_i), \]
\[ c_i = f_i \odot c_{i-1} + j_i \odot z_i, \]
\[ o_i = \delta(W_{xj}x_i + W_{hj}h_{i-1} + W_{c_o}c_i + b_o), \]
\[ h_i = o_i \tanh(c_i), \]

where \( h_i \in \mathbb{R}^{d/2} \times 1 \) is the output hidden state of node \( i \), \( \odot \) represents the element-wise product, \( W \in \mathbb{R}^{(d/2)\times(d/2)} \) and \( b \in \mathbb{R}^{d/2-1} \) are learnable parameters, which denote weight and bias, respectively; \( j_i, f_i, o_i \) 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.

### 3.4 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 \( h_i^c \) 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

\[ z_i = \text{Feedforward}(h_i^c), \]
\[ p_i^t = \text{softmax}(W_{\text{MNM}} z_i^t), \]

where \( z_i^t \) is the output of the feedforward layer, \( W_{\text{MNM}} \in \mathbb{R}^{V^t \times d} \) 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, \( p_i^t \) is the predicted distribution of \( x_i^t \) over all nodes.

For training, we use the cross-entropy between the one-hot label \( y_i^t \) and the prediction \( p_i^t \)

\[ L_{\text{MNM}} = - \sum_m y_m^t \log p_m^t, \]

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

### 3.5 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 adjacent node prediction (ANP) and masked node modeling (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 \( u \) with sequence \( X_u \) and node \( v \) with sequence \( X_v \), 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 \( W_{\text{ANP}} \), the scoring function \( s_r \) of whether the node pair is adjacent is shown as follows:

\[ s_r = \text{sigmoid}(CW_{\text{ANP}}^T), \]

where \( s_r \in \mathbb{R}^2 \) is a binary vector with \( s_{r0}, s_{r1} \in [0, 1] \) and \( s_{r0} + s_{r1} = 1 \), \( C \in \mathbb{R}^{d \times d} \) denotes the hidden vector of classification label used in a transformer architecture [18]. Considering the positive adjacent node pair \( S^+ \) and a negative adjacent node pair \( S^- \), we calculate a cross-entropy loss as follows:

\[ L_{\text{ANP}} = - \sum_{r \in S^+ \cup S^-} (y_r \log(s_{r0}) + (1 - y_r) \log(s_{r1})), \]

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

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

\[ L = L_{\text{MNM}} + L_{\text{ANP}}. \]

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.

### 3.6 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 [46]. Instead of setting the node embedding size \( Q \) to be equal to the hidden layer size \( H \) like BERT [18] 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 [46], 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 1.

### Algorithm 1: 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. (9);
8: end
9: Feed the two sequences into transformer layers;
10: Calculate the adjacent node prediction loss by Eq. (11);
11: Update the parameters $\Theta$ by Adam.
12: end
13: return Optimized pre-trained model parameters $\Theta^*$.

### 3.7 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:

i) link prediction,
i) similarity search,
iii) node classification, and
iv) 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 4 below.

### 4 Experimental Setup

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

#### 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,
i) DBLP,
i) 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 restaurants are separated into

i) Chinese food,

3. https://www.openacademic.ai/oag/
4. http://dblp.uni-trier.de
5. https://www.yelp.com/dataset_challenge
TABLE 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           |

ii) fast food, and
iii) sushi bar.

YAGO 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 1. In this paper, 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.

4.2 Algorithms Used for Comparison

We first choose network embedding methods to directly train the downstream datasets for specific tasks as baselines: DeepWalk [4], LINE [5] and node2vec [6]; 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 [7], HINE [8], HIN2Vec [9]. They are all based on metapaths, but differ in the way they use metapath features: metapath2vec adopts heterogeneous Skip-Grams, 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 [10], GAT [11], GraphSAGE [36] and GIN [39], 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 [12], HGT [40] 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.

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 [17], GCC [16] and PT-HGNN [44] 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 adopts node-level and schema-level contrastive learning as the pre-training task.

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 initialized 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 prediction and modeling 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 ▲.

5 RESULTS AND ANALYSIS

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

i) link prediction,
ii) similarity search,
iii) node classification, and
iv) node clustering.

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

5.1 Downstream Tasks

5.1.1 Link Prediction

This task is to predict which links will occur in the future. Unlike previous work [6] that randomly samples a certain
outperforms GCC. This is because our pre-training model on learning heterogeneous node type features. Compared to other methods, PF-HIN outperforms an average of 4% on OAG-mini and DBLP datasets. 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. HGT and PT-HGNN outperform the above methods, which shows the power of combining GNN for similarity search. 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.

5.1.2 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. We define the AUC value as

$$\text{AUC} = \frac{1}{|V|} \sum_{u \in V} \frac{\sum_{v \in V : g(u) = g(v), g(u) \neq g(v')} \text{sim}(u, v) > \text{sim}(u, v')} {\sum_{v' \in V : g(u) = g(v), g(u) \neq g(v')} 1}.$$  

Table 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. HGT and PT-HGNN 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.

5.1.3 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 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 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 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, HINE2Vec); 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. HGT and PT-HGNN 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 datasets. This is because that these two datasets share the same domain of 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.
micro-F1 (MIC) and macro-F1 (MAC) as our evaluation metrics.

Table 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.

### 5.1.4 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 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: Results for the Similarity Search Task

| Model     | OAG-mini | DBLP | YELP | YAGO |
|-----------|----------|------|------|------|
|           | MIC      | MAC  | MIC  | MAC  |
| 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|

### Table 4: Results for the Multi-Label Node Classification Task

| Model     | OAG-mini | DBLP | YELP | YAGO |
|-----------|----------|------|------|------|
|           | MIC      | MAC  | MIC  | MAC  |
| DeepWalk  | 0.175    | 0.173| 0.193| 0.191|
| LINE      | 0.177    | 0.172| 0.184| 0.179|
| node2vec  | 0.180    | 0.178| 0.201| 0.198|
| metapath2vec | 0.195  | 0.194| 0.209| 0.207|
| HINE      | 0.198    | 0.192| 0.204| 0.201|
| HIN2Vec   | 0.204    | 0.201| 0.246| 0.241|
| GCN       | 0.209    | 0.205| 0.257| 0.256|
| GraphSage | 0.211    | 0.207| 0.267| 0.269|
| GAT       | 0.213    | 0.211| 0.271| 0.273|
| GIN       | 0.218    | 0.223| 0.274| 0.277|
| HetGNN    | 0.234    | 0.237| 0.285| 0.282|
| HGT       | 0.239    | 0.241| 0.283| 0.278|
| GPT-GNN   | 0.276    | 0.274| 0.304| 0.299|
| GCC       | 0.266    | 0.264| 0.295| 0.291|
| PT-HGNN   | 0.297    | 0.287| 0.316| 0.315|
| PF-HIN    | 0.294    | 0.292| 0.318| 0.311|

### Table 5: Results for the Node Clustering Task

| Model     | OAG-mini | DBLP | YELP | YAGO |
|-----------|----------|------|------|------|
|           | NMI      | ARI  | NMI  | ARI  |
| DeepWalk  | 0.601    | 0.613| 0.672| 0.686|
| LINE      | 0.598    | 0.609| 0.678| 0.693|
| node2vec  | 0.604    | 0.616| 0.673| 0.689|
| metapath2vec | 0.628  | 0.632| 0.711| 0.738|
| HINE      | 0.634    | 0.638| 0.718| 0.741|
| HIN2Vec   | 0.637    | 0.640| 0.721| 0.744|
| GCN       | 0.616    | 0.641| 0.701| 0.719|
| GraphSage | 0.619    | 0.643| 0.705| 0.722|
| GAT       | 0.622    | 0.645| 0.709| 0.728|
| GIN       | 0.625    | 0.650| 0.711| 0.735|
| HetGNN    | 0.644    | 0.678| 0.729| 0.748|
| HGT       | 0.648    | 0.682| 0.734| 0.753|
| GPT-GNN   | 0.673    | 0.712| 0.756| 0.761|
| GCC       | 0.652    | 0.707| 0.741| 0.756|
| PT-HGNN   | 0.694    | 0.731| 0.773| 0.776|
| PF-HIN    | 0.691    | 0.728| 0.770| 0.773|

8380 IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 35, NO. 8, AUGUST 2023
### 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 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 that 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. We also compare the pre-training time of these pre-training frameworks in Table 7. PF-HIN is the most efficient one since the complexity of the transformer encoder we use is lower than that of HGT in GPT-GNN, that of GIN in GCC, and that contrastive framework in PT-HGNN.

### 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.

#### 5.3.1 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\ MNN and PF-HIN\ ANP. PF-HIN \ MNN 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 8 shows the experimental results of the ablation analysis over pre-training tasks. For the link prediction task, ANP is more important than MNN 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, MNN plays a more important role, and this is because MNN directly models node features and hence PF-HIN is better able to explore them.

#### 5.3.2 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 9 presents the 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.

#### 5.3.3 Effect of Pre-Training Strategy

In PF-HIN, the MNN 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.
The strategy selected for the downstream classifier in an end-to-end manner.

In this paper, 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\textunderscore betweenness; the second is to remove the eigencentrality denoted as PF-HIN\textunderscore eigen; the third is to remove the closeness centrality denoted as PF-HIN\textunderscore closeness. Each variant assigns equal weights for the left two metrics.

Table 12 presents the experimental results. Removing a metric influences the experimental results, which illustrates that each metric is necessary for ranking the 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.

5.3.6 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 13 shows the experimental results. PF-HIN consistently outperforms PF-HIN (Freeze), which shows that full fine-tuning is helpful to boost the model performance.
5.4 Parameter Sensitivity

Finally, we conduct a sensitivity analysis of the hyperparameters 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. Figs. 2 and 3 show the results of our parameter analysis.

### TABLE 13
Ablation Analysis of the Fine-Tuning Setting on the DBLP Dataset

| Model          | LP AUC | SS F1 | MC AUC | NC MIC | NC MAC | NC NMI | NC 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  |

Same abbreviations used as in Table 6.

Fig. 2. Sensitivity w.r.t. maximum length of sequences.

Fig. 3. Sensitivity w.r.t. dimension of node embeddings.
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.

ACKNOWLEDGMENTS

All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

REFERENCES

[1] R. Reinanda, E. Meij, and M. de Rijke, “Knowledge graphs: An information retrieval perspective,” Found. Trends Inf. Retrieval, vol. 14, no. 4, pp. 289–444, Oct. 2020.
[2] Y. Sun and J. Han, Mining Heterogeneous Information Networks: Principles and Methodologies. San Rafael, CA, USA: Morgan & Claypool, 2012.
[3] H. Cai, V. W. Zheng, and K. C. Chang, “A comprehensive survey of graph embedding: Problems, techniques, and applications,” IEEE Trans. Knowl. Data Eng., vol. 30, no. 9, pp. 1616–1637, Sep. 2018.
[4] B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk: Online learning of social representations,” in Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2014, pp. 701–710.
[5] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, “LINE: Large-scale information network embedding,” in Proc. 24th Int. Conf. World Wide Web, 2015, pp. 1067–1077.
[6] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2016, pp. 855–864.
[7] Y. Dong, N. V. Chawla, A. Swami, “metapath2vec: Scalable representation learning for heterogeneous networks,” in Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2017, pp. 135–144.
[8] Z. Huang and N. Mamoulis, “Heterogeneous information network embedding for meta path based proximity,” 2017, arXiv:1701.05291.
[9] T. Fu, W. Lee, and Z. Lei, “HIN2Vec: Explore meta-paths in heterogeneous information networks for representation learning,” in Proc. ACM Int. Conf. Knowl. Manage., 2017, pp. 1797–1806.
[10] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proc. Int. Conf. Learn. Representations, 2017.
[11] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” in Proc. Int. Conf. Learn. Representations, 2018.
[12] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, “Heterogeneous graph neural network,” in Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2019, pp. 793–803.
[13] A. M. Dai and Q. V. Le, “Semi-supervised sequence learning,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2015, pp. 3079–3087.
[14] M. E. Peters et al., “Deep contextualized word representations,” in Proc. Conf. North Amer. Assoc. Comput. Linguistics: Hum. Lang. Technol., 2018, pp. 2227–2237.
[15] J. Howard and S. Ruder, “Universal language model fine-tuning for text classification,” in Proc. Conf. Assoc. Comput. Linguistics, 2018, pp. 328–339.
[16] J. Qiu et al., “GCC: Graph contrastive coding for graph neural network pre-training,” in Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2020, pp. 1150–1160.
[17] Z. Fu, Y. Dong, K. Wang, K. Chang, and Y. Sun, “GPT-GNN: Generative pre-training of graph neural networks,” in Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2020, pp. 1857–1867.
[18] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Proc. Conf. North Amer. Assoc. Comput. Linguistics: Hum. Lang. Technol., 2019, pp. 4171–4186.
[19] M. Belkin and P. Niyogi, “Laplacian eigenmaps and spectral techniques for embedding and clustering,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2001, pp. 585–591.
