Discovering Localized Information for Heterogeneous Graph Node Representation Learning

Lin Meng[1]
Florida State University
Tallahassee, FL, USA
lin@ifmlab.com

Ning Yan
Futurewei Technologies Inc.
Santa Clara, CA, USA
nyan@futurewei.com

Masood Mortazavi
Futurewei Technologies Inc.
Santa Clara, CA, USA
masood.mortazavi@futurewei.com

Jiawei Zhang
Florida State University
Tallahassee, FL, USA
jiawei@ifmlab.com

2020 IEEE 6th International Conference on Collaboration and Internet Computing (CIC) | 978-1-7281-4146-6/20/$31.00 ©2020 IEEE | DOI: 10.1109/CIC50333.2020.00020

***Abstract*—Representation learning for heterogeneous graphs aims at learning meaningful node (or edge) representations to facilitate downstream tasks such as node classification, node clustering, and link prediction. While graph neural networks (GNNs) have recently proven to be effective in representation learning, one of the limitations is that most investigations focus on homogeneous graphs. Existing investigations on heterogeneous graphs often make direct use of meta-path type structures. Meta-path-based approaches often require *a priori* designation of meta-paths based on heuristic foreknowledge regarding the characteristics of heterogeneous graphs under investigation.**

In this paper, we propose a model without any *a priori* selection of meta-paths. We utilize locally-sampled (heterogeneous) context graphs "centered" at a target node in order to extract relevant representational information for that target node. To deal with the heterogeneity in the graph, given the different types of nodes, we use different linear transformations to map the features in different domains into a unified feature space. We use the classical Graph Convolution Network (GCN) model as a tool to aggregate node features and then aggregate the context graph feature vectors to produce the target node's feature representation. We evaluate our model on three real-world datasets. The results show that the proposed model has better performance when compared with four baseline models.

***Index Terms*—Graph Mining, Graph Neural Network, Heterogeneous Node Embedding**

I. Introduction

Heterogeneous graphs contain multi-typed nodes and structural relations as well as abundant content associated with nodes [1]–[3]. Many real-world applications are associated with heterogeneous graphs [1]–[4]. For example, the academic graph shown in Fig. 1 is an example, where papers, authors, venues, and terms are four types of nodes. The corresponding edges include author-paper (write/written), paper-venue (publish), and paper-terms (contain). Structures of this kind (involving multiple types of nodes and edges) give rise to new categories of problems such as node classification, relation inference, and personalized recommendation [1], [2], [4]–[6].

Traditionally, works on classification and representation in heterogeneous graphs has relied on manual feature engineering. These works use graph specifications and statistics of node type (or properties) in order to learn node embeddings which can then be used for downstream tasks.

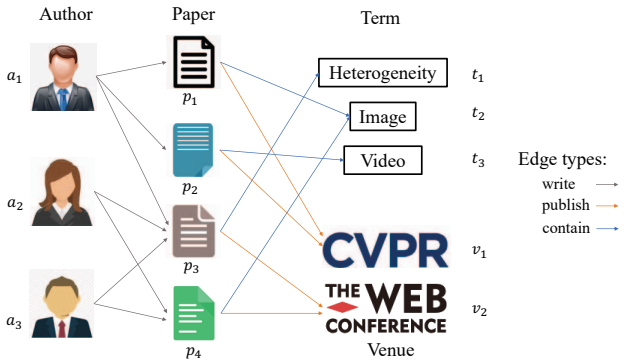


Fig. 1: An Example of DBLP Heterogeneous Graph.

Starting from homogeneous graphs, many works have used random walk-based approaches or their variants to accomplish the target embeddings. For instance, DeepWalk [7] first samples a set of short random walk paths and then feed these paths into a SkipGram Model to learn node embedding by approximating the node co-occurrence probability in these random walk paths. Later, Y. Sun, et al. [8] proposed to build meta-paths specifically designed for heterogeneous graphs. Meta-paths are connected sequences of node types. Other similar works have been proposed based on such meta-path structures. Metapath2vec [9] feeds meta-path guided random walk paths into a SkipGram model. Recent works also utilize Graph Neural Networks (GNNs) techniques for analyzing heterogeneous graphs. One advantage of GNN is that it can constantly update the node embedding by aggregating the embedding of neighboring nodes [10], [11]. HetGNN [12] samples a number of nodes for all node types via random walks and it uses the bidirectional LSTM to encode node content features. It further aggregates node content features of selected nodes to arrive at the target node embedding. Extensive works have been proposed for applying GNN together with meta-path. HAN [13] uses pre-defined meta-paths

[1]The work is performed during an internship at Futurewei Technologies Inc.

978-1-7281-4146-6/20/$31.00 ©2020 IEEE
DOI 10.1109/CIC50333.2020.00020

to construct homogeneous graphs for target nodes. It then aggregates them via both node level attention (when aggregating nodes connected via the same type of meta-paths) or semantic level attention (when aggregating all the meta-paths). MAGNN [14] makes some further improvements on HAN. It argues that the intermediate nodes in the meta-paths also play an important role when learning node embeddings. However, it relies on domain experts' pre-defining of meta-paths and on the costly computation for obtaining all the instances of meta-paths. As such, the automatic recognition and learning of the meta-paths themselves becomes a meaningful challenge along with the related node embedding that depends on them. Graph Transformer Network (GTN) [5] selects edge types by 1×1 convolution kernels and multiplies the matrices of edge types to build a matrix containing meta-path instance count. Graph Convolutional Network [10] applies graph convolution to aggregate node features via the learnt paths. Heterogeneous Graph Transformer (HGT) [15] incorporates more complex data settings of multi-type nodes and multi-relation edges and proposes a graph transformer model for million-scale datasets. HGT uses both node type-based self-attention and edge type-based message passing to get to node embeddings. Unfortunately, this model cannot easily adapt to distributed parallel computation and requires special engineering.

To sum up, the previously proposed methods suffer from at least one of the following limitations: (1) Purely random walks do not capture the heterogeneity (or type) structure in the graphs, which causes loss (or at best, discounting) of information. (2) Meta-path-guided random walk methods, while achieving better performance, need *a priori* designation of the meta-paths. This requires manual efforts and domain knowledge to choose the best meta-paths. It is biased to that knowledge and maynot be able to discover or capture novel, significant structures. (3) Current proposed models that can automatically select meta-paths through judicious use of neural network architectures are potentially hard to parallelize as they often rely on the global graph matrix. On the contrary, an architecture that only relies on local graph information is easier for parallel training. To overcome these limitations, we seek a novel graph neural network-based approach that does not require *a priori* meta-path designation, can easily be parallelized during training for large datasets and can scale and maintain good performance all around.

In this paper, we propose a heterogeneous graph neural networks (HetLGN) that only utilizes the local information to learn the node representations without any pre-knowledge or manual efforts for feature engineering. Our proposed approach can be easily adapted to any parallel training systems. We first sample a context graph for each target node, and then obtain the corresponding node content and structural features. To deal with the heterogeneity in the graph, we project the features of different node types into a unified space. We then use a common GNN module, e.g., graph convolution networks (GCN), to learn the context graph embedding whose aggregation produces the target node embedding. We conduct experiments on three real-world datasets. Our results show the effectiveness of our proposed model.

The rest of the paper is organized as follows: We discuss the related work in Section II. We define the notations and problem in Section III. We give a detailed model in Section IV. We show the experimental setting and result in Section V. Section VI concludes the paper.

II. Related Work

Our work relates to three parts: (1) graph representation learning; (2) heterogeneous graph mining; (3) graph neural networks.

A. Graph Representation Learning

Graph representation learning aims at learning the node or edge representations for general tasks. Early works like [16], [17] uses the matrix factorization (MF) method to compute the node embedding, but the MF methods are strictly restricted to relatively small graphs. To deal with large graphs, works like [7], [18] use sampling based methods. DeepWalk [7] samples a set of sequences and uses skipgram-based model to approximate the similarity between nodes. Node2vec [18] improves the sampling method of DeepWalk. LINE [19] calculates the 1- and 2-hop neighbors to make it applicable to large graphs. Many works also propose "deep learning" (DL) techniques. SDNE [20] uses auto-encoder model as well as a Laplacian eigenmap to learn node embedding in a semi-supervised manner. Spectral CNN [21] proposes two constructions, one based upon a hierarchical clustering of the domain, and the other based on the spectrum of graph Laplacian to learn node embedding.

B. Heterogeneous Graph Mining

Heterogeneous graph mining has been studied for over a decade to deal with various tasks such as node classification [12], [13], item recommendation [22], [23], which have no simple analogy in homogeneous graphs, and link prediction [6], [24] which can be used for graph completion or in recommendation systems. PathSim [8] proposes to use a meta-path based similarity score to find similar nodes. With the development of DL techniques, many DL models based on meta-path have been proposed [2], [22]. HERec [2] uses meta-path-based random walks in order to select node sequences. This work uses extended Matrix Factorization (MF) to learn the node embedding. MCRec [22] proposes a novel neural networks with co-attention mechanism for leveraging rich meta-path information for Top-K recommendation. HAN [13] proposes a two level attention mechanism to fuse the meta-path based sequences with their semantic meaning. MAGNN [14] improves HAN by aggregating all the nodes in meta-paths together. HetGNN [12] uses Bi-directional LSTM as a feature encoding unit and an aggregation tool. However, its pre-defined meta-path greatly limits the potential impact of meta-path structures. GTN [5] selects the meta-paths by softly selecting edge types. HGT [15] does not require pre-defined meta-path. It proposes to use stacked multiple heterogeneous graph transformer layers in order to co-compute the meta-paths along with other potential tasks.

C. Graph Neural Networks

Graph Neural Network (GNN) is a different type of neural network compared with other deep learning models such as CNN, LSTM. It aims at handling data represented as graph structures.

GNNs usually take two steps during graph data processing: (1) feature projection, and (2) aggregation An early GNN work is Graph Convolutional Network (GCN) [10] which calculates weights for edges and aggregates one-hop neighboring nodes based on these weights. This thread of research has been extended by other works [11], [24]–[30]. A major difference among GNN variations lies in their propagation methods. ClusterGCN [28] splits nodes into several clusters and proposes an inter-propagation for clusters, which accelerates the training procedure for large graphs. SimpleGCN [29] accelerates the learning process by removing the redundant linear transformation. Deep Graph Informax [30] learns the node embedding by maximizing the mutual information of nodes. GraphSAGE [26] samples neighboring nodes in order to reduce the computation complexity for large homogeneous graphs.

Moreover, there are some other works for graph-level representation learning. To learn the graph representations, GNNs adopt different types of readout or pooling function. EigenGCN [31] uses an eigen-pooling to maintain the graph structural information by eigen decomposition. GIN [32] proposes several pooling methods and proves they have similar powers to Weiferiler-Leman Kernel [33]. JK-Net [34] argues that local information suffers losses during the propagation process. It proposes skip-connections from layer to layer. A more recent work [35] learns both node and graph representations by contrasting embedding between node representations from one view and graph representations from another view.

III. Terminology and Problem Definition

A. Notations and Definitions

To formulate our problem, we define the heterogeneous graph as follows. [1]

Definition *1:* (Heterogeneous Graph) Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the node set, $\mathcal{E}$ is the edge set, there exists type mapping functions $\tau : \mathcal{V} \rightarrow \mathcal{A}$ and $\phi : \mathcal{E} \rightarrow \mathcal{R}$ map nodes to types $\mathcal{A}$ and edges to types $\mathcal{R}$, respectively. A graph is heterogeneous when satisfies constraint: $|\mathcal{A}| + |\mathcal{R}| > 2$.

B. Problem Formulation

We formulate the node representation learning problem in heterogeneous graphs as follows.

Definition *2:* (Heterogeneous Node Representation Learning) Given a heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \tau, \phi)$ and associated node contents, our task is to find a mapping function $f : \mathcal{V}_t \rightarrow \mathbb{R}^{|\mathcal{V}_t| \times d}$ that maps the target nodes $\mathcal{V}_t$ into a d-dimensional vector space.

[1]Throughout the paper, we use lowercase letters (e.g., x) to denote scalars. We use lowercase boldface letters (e.g., $\mathbf{x}$) to denote vectors. We use uppercase boldface letters $\mathbf{X}$ to denote matrices. The i-th row of the matrix $\mathbf{X}$ is denoted as $\mathbf{X}(i,:)$ or $\mathbf{x}_i$.

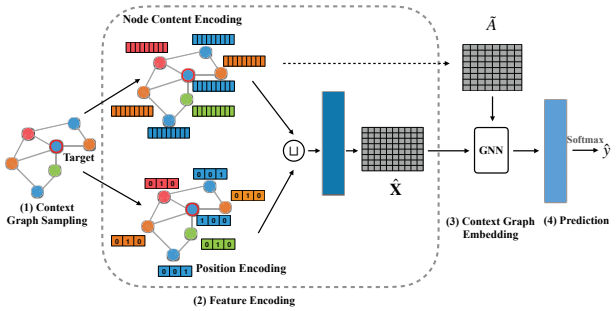


Fig. 2: An illustration of the HetLGN framework.

The learned node representations can be easily applied to any downstream tasks such as node classification, node clustering, link prediction, etc.

IV. The Proposed Method

In this section, we present HetLGN framework illustrated in Figure 2 to address the limitations of previous graph neural network methods described in Section 1. The HetLGN framework consists of four components: (1) context graph sampling, (2) feature encoding, (3) context graph embedding, and (4) prediction. These four components are further elaborated as follows.

A. Context Graph Sampling

Most of the research works for heterogeneous node embedding uses random walk paths and meta-paths as the context to learn the node embedding. However, those path-based methods focus more on sequential information rather than graph structural information. Although using multiple sequences may partially capture certain graph structures, they cannot fully exploit the graph structure and its semantics.

Besides, the purely random walk-based approaches ignore the heterogeneity of node/edge types, and the pre-defined meta-path based approaches require additional manual analysis on the data. Neither are suitable for modern graph analysis tasks over very large heterogeneous datasets.

Therefore, to enhance information obtained from the local sub-graphs, we need to guarantee both heterogeneous and locally dense structures that can be captured by the context graph sampling method. Inspired by the mini-batch sampling used in [15], we propose a *multi-branched* context graph sampling algorithm which aims to sampling a dense local structure while at the same time handling the heterogeneity in the graph.

In order to sample more diversified type information in graph, we follow the intuition that nodes with higher degrees will provide less type information than those with lower degrees when sampled with equal probability. Based on this intuition, we employ a sampling strategy that visits nodes based on a probability derived from a "type budget" selection function.

In addition, to deal with the node heterogeneity, the sampling probability is updated with respect to the node types. This is done in order to ensure that the sampled context graph can have more evenly distributed types of nodes. This approach ensures that the context graphs avoid biases towards branching from types of neighboring nodes with higher degree. Similar to [15], we define a budget-based selection function which guarantees the number of nodes sampled are proportional to the corresponding node types. The detail of this method is shown in Algorithm 2.

The multi-branched context graph sampling method obtains a context graph around a given target node i. The search process starts with node i. It then calculates the sampling probabilities for all 1-hop neighboring nodes around node i. It further randomly sample w nodes from all neighboring nodes of node i according to the calculated probabilities. It continues to search for all the 2-hop neighboring nodes of the previously sampled nodes. Again, it samples w nodes from these 2-hop nodes according to the re-calculated probabilities. Iteratively, it continues to search until the d-hop neighboring nodes are reached. The algorithm is searching in a multi-branched fashion as it expands the search area from all previous sampled nodes stored in the budget B. The details are shown in Algorithm 1.

Having sampled the nodes, we are able to construct the context graph for node i, and its adjacency matrix denoted as $\hat{\mathbf{A}}_i$. Thus, for target node set $\mathcal{V}_t = \{1, \cdots, n\}$, we are able to compute the corresponding context graph adjacency matrices $\{\hat{\mathbf{A}}_1, ..., \hat{\mathbf{A}}_n\}$.

Algorithm 1: Multi-Branches Context Graph Sampling

Input: Adjacency matrix $\mathbf{A}$, target node i, sample depth d and sample width w.

Result: A set of context nodes $\mathcal{C}$=$\{i, c_1, \cdots, c_w\}$ for the target node i, adjacency matrix of sampled context graph $\hat{\mathbf{A}}_i$.

Initialize an empty context node set $\mathcal{C} = \{\}$;
$\mathcal{C}$.add($(\tau(i), i)$) ;
Initialize an empty Budget B storing nodes and their types with normalized degrees;
for $(\tau(t), t) \in \mathcal{C}$ **do**
    Add-In-Budget($B, (\tau(t), t),\ \mathbf{A},\ \mathcal{C}$); `// Add neighbors of t to B`
end
for *l←1 to d* **do**
    for *node* $t \in$ *domain* B **do**
        $prob^{l-1}[(\tau(t), t)] \leftarrow \frac{B[(\tau(t),t)]^2}{\sum_m B[(\tau(m),m)]^2}$; `// Calculate sampling probability all nodes`
    end
    Random sample w nodes $\{c_t\}_{t=1}^{w}$ from B using $prob^{l-1}$;
    for *node* $t \in \{c_j\}_{j=1}^{w}$ **do**
        $\mathcal{C}.add((\tau(t), t))$; `// Add node t as context node`
        Add-In-Budget($B, (\tau(t), t),\ \mathbf{A},\ \mathcal{C}$);
        $B.pop((\tau(t), t))$;
    end
end
Reconstruct the adjacency matrix of the sampled context graph $\hat{\mathbf{A}}_i$ based on sampled context node set $\mathcal{C}$;
return $\mathcal{C}$, $\hat{\mathbf{A}}_i$;

B. Feature Encoding

In heterogeneous graphs, nodes are almost always associated with abundant content information. In order to learn good embedding, the node content features play an important role. On the other hand, the position of a node in the context graph with regard to a target node also reveals structural information. (Structural features are also strong features for inductive learning [6].) However, the features of different node types are from various domains, which needs to be projected into a unified space for proper feature encoding. To handle this situation, we will first obtain the node content features, and then get the structural features according to the relative position with respect to the target node. Finally, we embed both node content features and structural feature according to the node type.

Formally, a sampled context graph $\hat{\mathbf{A}}_i$ contains context nodes $\mathcal{C} = \{i, g_1, g_2, \cdots, g_m\}$ where $m = w \times d$ (sampled w from each of the d levels) for target node i. The node content features can be obtained in a data pre-processing stage according to the raw content feature data types. For example, if the raw content features involve pieces of text, we can apply feature extractor such as transformers or word2vec. If features are images, we can apply deep neural models such as convolutional neural networks (CNNs) in a procedure of pre-trained feature extraction. Therefore, by applying existing models for text or image or other content type processing, we could construct the node content features $\mathbf{x}_i^{(c)}, \mathbf{x}_{g_1}^{(c)}, \cdots, \mathbf{x}_{g_m}^{(c)}$.

For structural features, we want to maintain the distance between the context nodes to the target node. Here, we first compute the shortest distances between the context nodes and the target node. After this computation, we use a one hot encoding to encode the shortest distance of each context node. Note that for target node itself, the distance is zero. As illustrated in Figure 2, we can obtain the node structure features $\{\mathbf{x}_i^{(s)}, \mathbf{x}_{g_1}^{(s)}, \cdots, \mathbf{x}_{g_m}^{(s)}\}$.

Combining both node content features and node structure features, we get the complete node feature $\mathbf{x}$ by concatenation as follows.

$$\mathbf{x}_k = \mathbf{x}_k^{(c)} \sqcup \mathbf{x}_k^{(s)}, \qquad \forall k \in \{i, g_1, g_2, \cdots, g_m\}. \tag{1}$$

Due to the node heterogeneity, we transform the node features into a unified space according to the node types. The projection can be denoted as

$$\mathbf{h}_k = \mathbf{W}_p^{\tau(k)} \mathbf{x}_k + \mathbf{b}_p^{\tau(k)} \tag{2}$$

Algorithm 2: Add-In-Budget($B, (\tau(t), t), \mathbf{A}, \mathcal{C}$)

Input: Adjacency matrix $\mathbf{A}$, target node t, and sampled context node set $\mathcal{C}$.
Result: Updated Budget B.

for *each possible adjacent node type* τ *and edge type* ϕ **do**
  $D_t \leftarrow 1/len(\mathbf{A}_{(\tau,\phi,\tau(t))}[t])$; // get normalized degree of added node t regarding to each type triplet $(\tau(s), \phi(e), \tau(t))$ derived from corresponding node triplet (s, e, t). $\mathbf{A}_{(\tau,\phi,\tau(t))}[t]$ represents all node types adjacency to node t.
  for *all neighboring nodes* k *of node* t **do**
    if k *is not in* $\mathcal{C}$ **then**
      $B[(\tau(k), k)] \leftarrow B[(\tau(k), k)] + D_t$
    end
  end
end
return Updated Budget B ;

where $\mathbf{W}_p^{\tau(k)}$ and $\mathbf{b}_p^{\tau(k)}$ are trainable parameters of weights and bias for node type $\tau(k)$, respectively, where $\tau(k)$ is the type of node k. Therefore, by projecting all nodes in all context graphs, in the same manner, we build feature matrices $\{\mathbf{H}_1, \cdots, \mathbf{H}_n\}$ for all target nodes $\{1, \cdots, n\}$.

C. Context Graph Embedding

After obtaining the context graph, we aim to get the context graph embedding and take it as the target node embedding. In this paper, we try to utilize the context graphs containing richer information than sequences to deal with the node embedding problem. Using graphs to learn the node embedding is reasonable due to the assumption that nodes are similar to their neighbor nodes. Unlike sequences, which only consider the path connections, the graph bears more structural information and it can make use of all information within the graph. Due to the recent development of graph neural networks, people find an efficient way to learn both node embedding and graph embedding for graph structured data. Potentially, any graph neural network can be applied to learn the context graph embedding. In this paper, we simply use the graph convolution network (GCN) [10] to learn the graph embedding for our context graphs.

Formally, we have a context graph adjacency matrix $\hat{\mathbf{A}}_i$ and the corresponding feature matrix $\mathbf{H}_i$. We first need to normalize the context graph adjacency matrix $\hat{\mathbf{A}}_i$. The normalized adjacency matrix can be computed as follows.

$$\tilde{\mathbf{A}}_i = \mathbf{D}_i^{-1/2} \hat{\mathbf{A}}_i \mathbf{D}_i^{-1/2} \tag{3}$$

where $\mathbf{D}_i(j,j) = \sum_j \hat{\mathbf{A}}_i(j,:)$ is a diagonal matrix. Together with the feature matrix $\mathbf{H}_i$, we feed them into a GCN. Here, we use L layers of the GCN to learn a comprehensive embedding as follows.

$$\mathbf{H}_i^{(l)} = \tilde{\mathbf{A}}_i \mathbf{H}_i^{(l-1)} \mathbf{W}_g^{(l-1)}, l = 1, \cdots, L. \tag{4}$$

where $\mathbf{H}_i^{(0)} = \mathbf{H}_i$ and $\mathbf{W}_g^{(l-1)} \in \mathbb{R}^{h \times h}$ denotes the trainable variable at $(l-1)$-th layer. To better fuse the features, we adopt the idea from JK-Net [34] by concatenating all feature matrix from L layers of the GCN network to compute the final context graph embedding by the follow method.

$$\mathbf{h}_i^{(g)} = \sum_j \mathbf{H}_i^{(1)}(j,:) \sqcup \cdots \sqcup \sum_j \mathbf{H}_i^{(L)}(j,:) \tag{5}$$

where the $\mathbf{h}_i^{(g)}$ is the context graph embedding. For each context graph fed into the learning module, our model produces embeddings $\{\mathbf{h}_i^{(g)} | \forall i \in \mathcal{V}_t\}$ for all target nodes in the batch.

D. Prediction

After getting all context graph embeddings, we simply use one linear layer as a classifier, which is formulated as follows.

$$\hat{\mathbf{y}}_i = \mathbf{W}_c \mathbf{h}_i^{(g)} + \mathbf{b}_c \tag{6}$$

where $\hat{\mathbf{y}}_i \in \mathbb{R}^{|C|}$ and $|C|$ denotes the number of classes. We use cross-entropy as the loss function. Given the ground truth $\mathbf{y}_i$, the loss function is

$$\mathcal{L} = -\sum_i^{\mathcal{B}} \sum_j^{C} (\mathbf{y}_i(j) \log \hat{\mathbf{y}}_i(j)) \tag{7}$$

During training, HetLGN samples $\mathcal{B}$ (i.e., batch size) context graphs for the (batch of) target nodes first and then trains the context-embedding neural network based on GCN model. When testing, the same forward procedure is used to predict node labels for nodes in the test set.

V. Experiment

In this section, we introduce the datasets, comparison methods, and evaluation metrics. Then we show the experimental setting, disucss the experimental results on three real-world datasets and provide a parameter study.

A. Dataset

To show the effectiveness of the proposed model, we performed experiments on three real-world heterogeneous graph datasets. Some basic statistics of the datasets are provided in Table I.

- **ACM** is an academic heterogeneous graph. The node types include paper, author, and field. The edge types include paper-paper (citation), paper-author (write), and paper-field (belong to). In this dataset, we select 3025 papers from three areas (Database, Wireless Communication, Data Mining) as the target nodes following the data preprocessing as described in [13]. The content features of paper nodes are constructed by the bag-of-words of keywords. Since only paper nodes have content features, we simply average the paper node content features to its

dataset	Relations(A-B)	Number of A	Number of B	Number of A-B	Feature	#Target Node
ACM	Paper-Author	12499	16910	37055	1903	3025
	Paper-Field	12499	73	12499		
	Paper-Paper	12499	12499	30789		
DBLP	Author-Paper	4057	14328	19645	334	4057
	Paper-Term	14328	8789	88420		
	Paper-Conf	14328	20	14328		
IMDB	Movie-Director	4768	2277	4796	1269	3462
	Movie-Actor	4768	5968	14639		

TABLE I: Dataset Statistics

neighboring nodes. We use the paper nodes as the target nodes and the field of papers as the labels to be predicted.

- **DBLP** is also an academic graph. We use a subset of DBLP which contains four node types (paper, author, term, and conference), and three relation types (paper-author, paper-term, and paper-conference). For this datasets, we construct 4057 authors and their connected nodes. For this dataset, the author features are the bag-of-words representations of the keywords of the connected papers. We also average the author features to its neighboring nodes. We use the author nodes as the target nodes and the area they are working on as labels to be predicted.
- **IMDB** is a graph in film domain. It contains three node types (movie, director, and actor) and two relations (movie director and movie actor). The movie features are built by the bag-of-words representations from the plot descriptions. We average the movie features to other node types as well. We select 3462 movies as the target nodes and the genres (Action, Comedy, and Drama) as the labels to be predicted.

B. Comparison Methods and Evaluation Metrics

Our work is related to meta-path based methods and graph neural network methods, thus we use two state-the-art heterogeneous graph models and two classic graph neural network models as our baselines.

- **GTN**: The Graph Transformer Network (GTN) [5] selects the edge type by 1×1 convolutional kernel, and then aggregates the softly selected edges to form a big adjacency matrix. GCN is applied to compute node embedding.
- **HAN**: Heterogeneous Graph Attention Network (HAN) [13] uses pre-defined meta-paths to construct a homogeneous graph and uses node-level and semantic level attention mechanism to learn a concrete node embedding.
- **GCN**: Graph Convolution Network (GCN) [10] is a graph neural network for homogeneous graphs. It uses the normalized adjacency matrix to weight the connections and aggregates neighboring node features based on it.
- **GAT**: Graph Attention Network (GAT) [11] is designed for homogeneous graphs as well. It adopts an adaptive normalization method by multi-head attention to learn the weights to aggregate features.

To compare the performance of all these methods, we use F1-micro and F1-marco to evaluate node classification tasks, adjusted random index (ARI) and normalized mutual information (NMI) to evaluate the clustering tasks.

C. Experimental Setting

- For our HetLGN, we set the sample depth $d = 2$ and sample width $w = 6$ for ACM dataset, sample depth $d = 2$ and sample width $w = 5$ for DBLP and sample depth $d = 2$ and sample width $w = 4$ for IMDB dataset. The hidden size of all weights is 8 for ACM and IMDB, 4 for DBLP. We use GCN with three layers for all datasets.
- For the GTN model[2], we follow their code use two GTN layers, one GCN layer and two linear layers for all datasets, and set the hidden size 24 for ACM and IMDB datasets and 12 for DBLP.
- For the HAN, we use the same meta-paths they defined in [13] for the three dataset. The embedding sizes are set to 8 for ACM and IMDB, 4 for DBLP. The number of heads are set to be 3 for all datasets.
- For GCN, we use two graph convolutional layers and the embedding sizes of both layers are set to 24 for ACM and DBLP, 12 for DBLP. We use one additional linear layer to prediction the node label.
- For GAT, to make fair comparison, we need to make sure the final embeddings have the same size as others. Therefore, we set the embedding sizes to 8 for ACM and IMDB, 4 for DBLP. We set the number of attention heads to 3. We also use an additional linear layer for the classification task.

D. Node Classification

We conduct experiments on node classification task for all three datasets. We randomly split the dataset with 80% as the training set, 10% as the validation set, and 10% as the testing set. All the results are obtained by the model which has the best validation score during training. The results are shown in Table II.

From the table, we can see that the HetLGN model outperforms all four previous methods on all three dataset unanimously. This indicates the effectiveness of our method. The number of parameters are also improved over the HAN method, and are almost equal or only slightly worse than the other three methods. Moreover, HetLGN achieves these good results without any pre-defined meta-paths which makes it more useful to many real world problems where such pre-definition is too time-consuming and perhaps impractical.

[2]https://github.com/seongjunyun/Graph_Transformer_Networks

Dataset	Metrics	GCN	GAT	HAN	GTN	Het-LGN
ACM	F1-micro	0.8421	0.7763	0.9208	0.9177	**0.9473**
	F1-macro	0.8439	0.7770	0.9215	0.9177	**0.9476**
	#parameter	45771	45825	94843	46977	47195
DBLP	F1-micro	0.8476	0.8083	0.9337	0.9336	**0.9557**
	F1-macro	0.8444	0.8008	0.9292	0.9293	**0.9536**
	#parameter	4059	4089	13940	4402	5608
IMDB	F1-micro	0.6581	0.6327	0.6676	0.6581	**0.6827**
	F1-macro	0.6580	0.6322	0.6673	0.6593	**0.6835**
	#parameter	30555	30609	64411	31761	31963

TABLE II: Node Classification Result

Dataset	Metrics	GCN	GAT	HAN	GTN	Het-LGN
ACM	NMI	0.6430	0.4138	0.7951	0.7813	**0.8159**
	ARI	0.6970	0.3306	0.8368	0.8279	**0.8607**
DBLP	NMI	0.5616	0.3614	0.8056	0.8091	**0.8262**
	ARI	0.5742	0.1718	0.8478	0.8525	**0.8764**
IMDB	NMI	0.2046	0.1888	0.3401	0.5632	**0.6302**
	ARI	0.2160	0.1018	0.3765	0.6329	**0.7065**

TABLE III: Node Clustering Result

Besides, we make some interesting observations. GCN and GAT are not performing well on heterogeneous graphs, and GAT always performs inferior than GCN. This is probably due to the fact that GAT only uses aggregation from 1-hop neighboring nodes while GCN is stacking multiple layers. In this way, GCN can incorporate multi-hop nodes which enriches the information obtained from heterogeneous graphs. Comparing HAN and GTN, it seems that when training data is large enough, HAN always performs better than GTN However, both HAN and GTN makes use of meta-paths and perform better than GCN and GAT.

E. Clustering

We also conduct experiments on the clustering task to evaluate the learned node embeddings. We utilize KMeans algorithm as our node clustering algorithm. The number of the clusters is set to be the number of classes in dataset. In addition, we use the same ground truth as in node classification. The results are shown in Table III.

From the table, we can see that HetLGN outperforms all four previous methods on all three dataset unanimously. It indicates that by learning from context graphs, the learned embeddings could be utilized for node clustering task. This is because the context graphs bring more integrated information, which lets the similar nodes share similar neighboring nodes and similar graph structures thus generates similar representations.

Besides, GAT performs worst among all comparison methods. This is due to the fact that GAT only considers the 1-hop neighboring nodes rather than considering the k-hop away information, which greatly degenerates the model effectiveness. GCN gets better results than GAT but still worse than other three. This indicates incorporating node heterogeneity can improve the clustering performance as well. While GTN and HAN have achieved comparable clustering results on two academic graphs, e.g., ACM and DBLP, GTN reaches much better performance than HAN on IMDB dataset. This indicates the meta-paths defined in [13] for HAN may not be the best meta-paths selection for IMDB dataset. And GTN benefits from learning the meta-paths dynamically.

F. Visualization

To observe the learned embedding in an intuitive way, we visualize the target node embeddings for four compared methods: (a) HetLGN, (b) GTN, (c) HAN, and (d) GCN, respectively. We use t-SNE [36] to visualize the paper embeddings in ACM dataset and color them by the domains they belong to. The visualizations are shown in Figure 3.

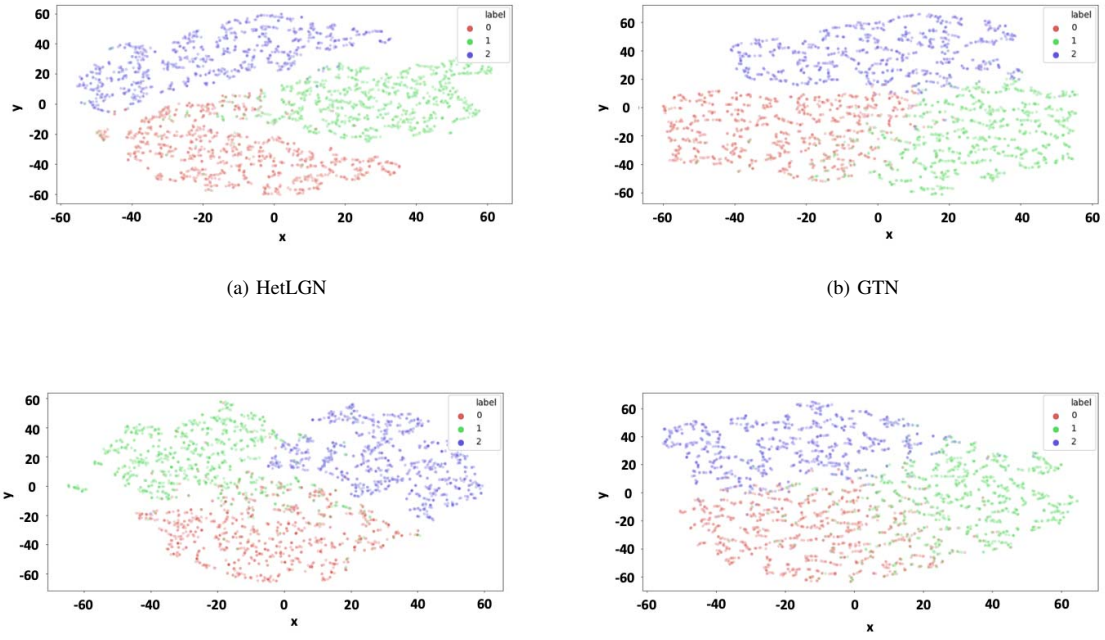
From Figure 3a, we can see that the three node clusters are well separated and the boundaries between clusters are also clear. Moreover, the proposed model does not need to define any meta path in advance, together with the clustering results, showing the merit of the proposed model.

From Figure 3d, we can see that when using GCN all three labels are still mixed together, especially for nodes in red and green color. Clustering results of both GTN and HAN in Figure 3b and Figure 3c are better than GCN, but part of nodes in red and green color are still mixed up. The purple ones are relative well separated, which demonstrates the effectiveness of using meta-paths .

G. Sampling Parameter Discussion

In this section, we investigate the sampling parameter "width" at each search depth. To better discuss the impact of width, we fixed the sample depth as 2. The experimental results on three datasets are shown in Table IV - Table VI.

The results shows that, for ACM dataset, setting width to 6 gets the best results. When setting width less than 6, the performance gets better with the width increases. However, for the other two datasets, the performance increase first and then decrease with the increasing of width, suggesting that sampling more nodes cannot enhance the performance anymore. There is also a greater chance of overfitting.



(a) HetLGN (b) GTN

(c) HAN (d) GCN

Fig. 3: Node Embedding Visualization

Dataset	Depth	Width	F1-micro	F1-marco
ACM	2	2	0.9342	0.9347
		3	0.9309	0.9319
		4	0.9375	0.9379
		5	0.9375	0.9377
		6	**0.9473**	**0.9476**

TABLE IV: ACM Sample Width Result

Dataset	Depth	Width	F1-micro	F1-Marco
DBLP	2	2	0.9189	0.9147
		3	0.9361	0.9298
		4	0.9410	0.9376
		5	**0.9533**	**0.9505**
		6	0.9385	0.9368

TABLE V: DBLP Sample Width Result

Dataset	Depth	Width	F1-micro	F1-Marco
IMDB	2	2	0.6601	0.6602
		3	0.6742	0.6742
		4	**0.6827**	**0.6835**
		5	0.6260	0.6265
		6	0.6232	0.6233

TABLE VI: IMDB Sample Width Result

VI. Conclusion

In this paper, we study the problem of heterogeneous graph node embedding. One measure of quality for any node embedding is whether it can provide a general node representation which can be applicable to many downstream tasks, e.g. node classification.

To improve node representations in heterogeneous graphs, we propose a model called HetLGN. HetLGN uses a multi-branched sampling method in order to sample context graphs for the target nodes. It then encodes both node content and structural features according to the node type. HetLGN uses a GCN module to learn a target node's context graph embedding which is then used as the target node embedding for downstream tasks.

To show the effectiveness of our proposed model, we conduct experiments on three datasets. These empirical results demonstrate that our model outperforms the meta-path-based methods, which have recently been used for heterogeneous graphs, and also other graph neural network techniques, which have been used for homogeneous graphs.

Our investigation indicates that a simple model with no *a priori* meta-path designations can still achieve very good and quite competitive results.

Acknowledgment

The authors would like to express their appreciation to Futurewei's IC-Lab and its managing VP, Dr. Liang Peng, for supporting this research project.

References

[1] H. Zhao, Q. Yao, J. Li, Y. Song, and D. L. Lee, "Meta-graph based recommendation fusion over heterogeneous information networks," in

Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017, pp. 635–644.

[2] C. Shi, B. Hu, W. X. Zhao, and S. Y. Philip, "Heterogeneous information network embedding for recommendation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 2, pp. 357–370, 2018.

[3] X. Kong, J. Zhang, and P. S. Yu, "Inferring anchor links across multiple heterogeneous social networks," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, pp. 179–188.

[4] T. Jiang, T. Liu, T. Ge, L. Sha, B. Chang, S. Li, and Z. Sui, "Towards time-aware knowledge graph completion," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, 2016, pp. 1715–1724.

[5] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 983–11 993.

[6] K. K. Teru, E. Denis, and W. L. Hamilton, "Inductive relation prediction by subgraph reasoning," *arXiv*, pp. arXiv–1911, 2019.

[7] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[8] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks," *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 992–1003, 2011.

[9] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 135–144.

[10] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[11] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[12] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 793–803.

[13] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *The World Wide Web Conference*, 2019, pp. 2022–2032.

[14] X. Fu, J. Zhang, Z. Meng, and I. King, "Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding," in *Proceedings of The Web Conference 2020*, 2020, pp. 2331–2341.

[15] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *Proceedings of The Web Conference 2020*, 2020, pp. 2704–2710.

[16] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, "Matching node embeddings for graph similarity," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[17] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM international on conference on information and knowledge management*, 2015, pp. 891–900.

[18] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[19] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

[20] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1225–1234.

[21] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[22] B. Hu, C. Shi, W. X. Zhao, and P. S. Yu, "Leveraging meta-path based context for top-n recommendation with a neural co-attention model," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1531–1540.

[23] X. Ren, J. Liu, X. Yu, U. Khandelwal, Q. Gu, L. Wang, and J. Han, "Cluscite: Effective citation recommendation by information network-based clustering," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 821–830.

[24] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*. Springer, 2018, pp. 593–607.

[25] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," *arXiv preprint arXiv:1810.05997*, 2018.

[26] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[27] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," *arXiv preprint arXiv:1801.10247*, 2018.

[28] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 257–266.

[29] F. Wu, T. Zhang, A. H. d. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," *arXiv preprint arXiv:1902.07153*, 2019.

[30] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax." in *ICLR (Poster)*, 2019.

[31] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 723–731.

[32] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[33] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels." *Journal of Machine Learning Research*, vol. 12, no. 9, 2011.

[34] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," *arXiv preprint arXiv:1806.03536*, 2018.

[35] K. Hassani and A. H. Khasahmadi, "Contrastive multi-view representation learning on graphs," *arXiv preprint arXiv:2006.05582*, 2020.

[36] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.