# Heterogeneous Graph Structure Learning for Graph Neural Networks

**Jianan Zhao,**[1,2*] **Xiao Wang,**[1*] **Chuan Shi,**[1†] **Binbin Hu,**[3] **Guojie Song,**[4] **Yanfang Ye**[2]
[1] School of CS, Beijing University of Posts and Telecommunications, Beijing, China
[2] Department of CDS, Case Western Reserve University, OH, USA
[3] Ant Group
[4] Key Laboratory of Machine Perception, Ministry of Education, Peking University
jxz1244@case.edu, {xiaowang, shichuan}@bupt.edu.cn,
bin.hbb@antfin.com, gjsong@pku.edu.cn, yanfang.ye@case.edu

## Abstract

Heterogeneous Graph Neural Networks (HGNNs) have drawn increasing attention in recent years and achieved outstanding performance in many tasks. The success of the existing HGNNs relies on one fundamental assumption, i.e., the original heterogeneous graph structure is reliable. However, this assumption is usually unrealistic, since the heterogeneous graph in reality is inevitably noisy or incomplete. Therefore, it is vital to learn the heterogeneous graph structure for HGNNs rather than rely only on the raw graph structure. In light of this, we make the first attempt towards learning an optimal heterogeneous graph structure for HGNNs and propose a novel framework HGSL, which jointly performs Heterogeneous Graph Structure Learning and GNN parameter learning for classification. Different from traditional homogeneous graph structure learning, considering the heterogeneity of different relations in heterogeneous graph, HGSL generates each relation subgraph separately. Specifically, in each generated relation subgraph, HGSL not only considers the feature similarity by generating feature similarity graph, but also considers the complex heterogeneous interactions in features and semantics by generating feature propagation graph and semantic graph. Then, these graphs are fused to a learned heterogeneous graph and optimized together with a GNN towards classification objective. Extensive experiments on real-world graphs demonstrate that the proposed framework significantly outperforms the state-of-the-art methods.

## Introduction

Many real-world data can be viewed as graphs, such as bibliographic graphs, citation graphs, and social media graphs. Graph Neural Network (GNN), as a powerful deep representation learning tool to deal with graph data, has drawn increasing attention and is widely applied to node classification (Kipf and Welling 2017; Velickovic et al. 2018; Xu et al. 2019), graph classification (Duvenaud et al. 2015; Lee, Lee, and Kang 2019), and recommendation (Ying et al. 2018; Fan et al. 2019b; Wang et al. 2019a). Recently, with the proliferation of real-world applications on heterogeneous graphs, which consist of multiple types of nodes and links (Shi et al. 2017), Heterogeneous Graph Neural Networks (HGNNs) are brought forward and have achieved remarkable improvements on series of applications (Wang et al. 2020a; Hu et al. 2019b; Li et al. 2019; Hu et al. 2019a; Fan et al. 2019a; Wang et al. 2020b).

Most HGNNs follow a message-passing scheme where the node embedding is learned by aggregating and transforming the embeddings of its original neighbors (Zhang et al. 2019a; Zhao et al. 2020a; Hong et al. 2020) or metapath-based neighbors (Wang et al. 2019b; Yun et al. 2019; Hu et al. 2020; Fu et al. 2020). These methods rely on one fundamental assumption, i.e. the raw heterogeneous graph structure is good. However, as heterogeneous graphs are usually extracted from complex interaction systems by some pre-defined rules, such assumption cannot be always satisfied. One reason is that, these interaction systems inevitably contain some uncertain information or mistakes. Taking a user-item graph built from recommendation as an example, it is well accepted that users may misclick some unwanted items, bringing noisy information to the graph. The other reason is that, the heterogeneous graphs are often extracted with data cleaning, feature extraction and feature transformation by some pre-defined rules, which are usually independent to the downstream tasks and lead to the gap between the extracted graph and the optimal graph structure for the downstream tasks. Therefore, learning an optimal heterogeneous graph for GNN is a fundamental problem.

Recently, to adaptively learn graph structures for GNNs, graph structure learning (GSL) methods (Franceschi et al. 2019; Jiang et al. 2019; Chen, Wu, and Zaki 2019; Jin et al. 2020) are proposed, most of which parameterize the adjacency matrix and optimize it along with the GNN parameters toward downstream tasks. However, these methods are all designed for homogeneous graphs, which can not be directly applied to heterogeneous graphs with the following challenges: (1) *The heterogeneity in heterogeneous graphs* When learning a homogeneous graph with only one type of relation, we usually only need to parameterize one adjacency matrix. However, a heterogeneous graph consists of multiple relations, each of which reflects one aspect of the heterogeneous graph. Since treating these heterogeneous relations uniformly will inevitably restrict the capability of graph structure learning. How to deal with this heterogeneity is a challenging problem. (2) *The complex interactions in heterogeneous graphs.* Different relations and node fea-

[*] Both authors contributed equally to this research.
[†] Corresponding author.

tures have complex interactions, which drives the formation of different kinds of underlying graph structure (Zhang, Swami, and Chawla 2019). Moreover, the combination of different relations further forms a large number of high-order relationships with diverse semantics, which also implies distinct ways of graph generation. The heterogeneous graph structure will be affected by all these factors, therefore, these complex interactions must be thoroughly considered in heterogeneous graph structure learning.

In this paper, we make the first attempt to investigate **H**eterogeneous **G**raph **S**tructure **L**earning for graph neural networks, and propose a novel framework HGSL. In HGSL, the heterogeneous graph and the GNN parameters are jointly learned towards better node classification performance. Particularly, in the graph learning part, aiming to capture the heterogeneous metric of different relation generation, each relation subgraph is separately learned. Specifically, for each relation, three types of candidate graphs, i.e. the feature similarity graph, feature propagation graphs and semantic graphs, are generated by mining the complex correlations from heterogeneous node features and graph structures. The learned graphs are further fused to a heterogeneous graph and fed to a GNN. The graph learning parameters and the GNN parameters are jointly optimized towards classification objective. Our major contributions are highlighted as follows:

- A suitable heterogeneous graph structure is one basic guarantee of a successful HGNN. To our best knowledge, we make the first attempt to study how to learn an optimal heterogeneous graph structure for GNN towards downstream task.
- We propose a novel heterogeneous graph neural network with heterogeneous graph structure learning, where three kinds of graph structures (feature similarity graph, feature propagation graph, semantic graph) are generated, so as to comprehensively fuse an optimal heterogeneous graph for GNN.
- We conduct extensive experiments on three real-world datasets to validate the effectiveness of HGSL against the state-of-the-art methods.

## Related Work

### Graph Neural Network (GNN)

Most of current GNNs can be generally divided into two categories: spectral GNNs and spatial GNNs (Wu et al. 2019b). Specifically, spectral GNNs learn node representation based on graph spectral theory. For example, (Bruna et al. 2014) designs the graph convolution operation in Fourier domain by the graph Laplacian. Then, ChebNet (Defferrard, Bresson, and Vandergheynst 2016) utilizes Chebyshev polynomials as the convolution filter to improve the efficiency. GCN (Kipf and Welling 2017) simplifies ChebNet by using its first-order approximation. Further, SGC (Wu et al. 2019a) reduces the graph convolution to a linear model and still achieves competitive performance. Spatial GNNs define convolution operations directly on the graph, utilizing spatially close neighbors. For instance, GAT (Velickovic et al. 2018) aggregates neighborhood representations with attention mechanism. GraphSAGE (Hamilton, Ying, and Leskovec 2017) performs inductive graph convolution by aggregating information from sampled neighbors. For better efficiency, FastGCN (Chen, Ma, and Xiao 2018) performs importance sampling on each convolution layer. Readers may refer to these elaborate surveys (Zhang, Cui, and Zhu 2018; Wu et al. 2019b) for a thorough review.

### Heterogeneous Graph Neural Network (HGNN)

HGNNs are proposed to deal with the ubiquitous heterogeneous data. Some HGNNs perform graph convolution directly on the original heterogeneous graphs. HGAT (Hu et al. 2019b) aggregates in node and type level information with attention mechanism for short-text classification. HetGNN (Zhang et al. 2019a) samples heterogeneous neighbors by random walk and then aggregates node and type information. To solve the metapath selection conundrum, HetSANN (Hong et al. 2020) aggregates multi-relational information of projected nodes by means of attention mechanism. NSHE (Zhao et al. 2020a) preserves the pairwise and network schema structure. HGT (Hu et al. 2020) adopts the meta-relation based mutual attention to perform message passing on heterogeneous graphs and learns the implicit meta paths. Other HGNN methods use metapaths to generate graphs and apply GNN afterwards. GraphInception (Zhang et al. 2018) applies graph convolution on metapaths based homogeneous graphs to perform collective classification. HAN (Wang et al. 2019b) applies node-level and semantic-level attention on metapath-based graphs. GTN (Yun et al. 2019) performs metapath generation via stacking multiple graph transformer layers and perform graph convolution afterwards. MAGNN (Fu et al. 2020) applies intra-metapath and inter-metapath aggregation on metapath instances.

### Graph Structure Learning (GSL)

To alleviate the limitation that GNNs rely on the good quality of raw graph structure, several efforts have been made (Zhang et al. 2019b; Zheng et al. 2020; Ye and Ji 2019; Zhao et al. 2020b; Wang et al. 2020c; Pei et al. 2020). Besides these efforts, very recently, graph structure learning was proposed. GSL methods aim to learn the graph structure and GNN parameters jointly. To illustrate, LDS (Franceschi et al. 2019) models each edge inside the adjacency matrix as a parameter and learns them along with GNN parameters in a bi-level fashion. GLCN (Jiang et al. 2019) generates similarity-based graph structure from node features. IDGL (Chen, Wu, and Zaki 2019) iteratively learns the metrics to generate graph structure from node features and GNN embeddings. ProGNN (Jin et al. 2020) jointly learns GNN parameters and a robust graph structure with graph properties. However, these aforementioned GSL methods are all designed for homogeneous graphs.

## Preliminaries

In this section, we introduce some basic concepts and formalize the problem of heterogeneous graph structure learning.

**Definition 1. Heterogeneous Graph** A Heterogeneous graph $G = (V, E, \mathcal{F})$ is composed of a node set $V$, an edge set $E$, and a feature set $\mathcal{F}$, along with the node type mapping function $\phi : V \rightarrow \mathcal{T}$, and the edge type mapping function $\psi : E \rightarrow \mathcal{R}$, where $\mathcal{T}$ and $\mathcal{R}$ denotes the node and edge types, $|\mathcal{T}| + |\mathcal{R}| > 2$. Let $V_\tau$ denote the node set of type $\tau \in \mathcal{T}$, the feature set $\mathcal{F}$ is composed of $|\mathcal{T}|$ feature matrix, $\mathcal{F} = \{\mathbf{F}_\tau, \tau \in \mathcal{T}\}$, $\mathbf{F}_\tau \in \mathbb{R}^{|V_\tau| \times d_\tau}$, where $V_\tau$ stands for all the node with $\tau$ type, $d_\tau$ stands for the feature dimension of $\tau$ nodes.

**Definition 2. Metapath** A metapath $P$ is defined as a path in the form of $v_1 \xrightarrow{r_1} v_2 \xrightarrow{r_2} \cdots \xrightarrow{r_l} v_{l+1}$, which describes a composite relation $r_1 \circ r_2 \circ \cdots \circ r_l$ between two nodes $v_1$ and $v_{l+1}$, where $\circ$ denotes the composition operator on relations.

**Definition 3. Node Relation Triple** A node relation triple $\langle v_i, r, v_j \rangle$, describes that two nodes $v_i$ (head node) and $v_j$ (tail node) are connected by relation $r \in \mathcal{R}$. We further define the type mapping functions $\phi_h, \phi_t : \mathcal{R} \rightarrow \mathcal{T}$ that map the relation to its head node type and tail node type respectively.

**Example**: In a user-item heterogeneous graph, say $r =$ "$UI$" (a user buys an item), then we have $\phi_h(r) =$ "$User$" and $\phi_t(r) =$ "$Item$".

**Definition 4. Relation Subgraph** Given a heterogeneous graph $G = (V, E, \mathcal{F})$, a relation subgraph $G_r$ is a subgraph of $G$ that contains all node-relation triples with relation $r$. The adjacency matrix of $G_r$ is $\mathbf{A}_r \in \mathbb{R}^{|V_{\phi_h(r)}| \times |V_{\phi_t(r)}|}$, where $\mathbf{A}_r[i, j] = 1$ if $\langle v_i, r, v_j \rangle$ exists in $G_r$, otherwise $\mathbf{A}_r[i, j] = 0$. $\mathcal{A}$ denotes the relation subgraph set of all the relation subgraphs in $G$, i.e. $\mathcal{A} = \{\mathbf{A}_r, r \in \mathcal{R}\}$.

**Definition 5. Heterogeneous Graph Structure Learning (HGSL)**. Given a heterogeneous graph $G$, the task of heterogeneous graph structure learning is to jointly learn a heterogeneous graph structure, i.e. a new relational subgraph set $\mathcal{A}'$, and the parameters of GNN for downstream tasks.

# The Proposed Method

## Model Framework

Figure 1 (a) illustrates the framework of the proposed HGSL. As we can see, given a heterogeneous graph, HGSL firstly constructs the semantic embedding matrices $\mathcal{Z}$ by metapath-based node embeddings from $M$ metapaths. Afterwards, the heterogeneous graph structure and GNN parameters are trained jointly. For the graph learning part, HGSL takes the information from the original relation subgraph, the node features, and the semantic embeddings as input and generates relation subgraphs separately. Specifically, taking relation $r_1$ as an example, HGSL learns a feature graph $\mathbf{S}_{r_1}^{Feat}$ and a semantic graph $\mathbf{S}_{r_1}^{Sem}$ and fuse them with the original graph $\mathbf{A}_{r_1}$ to obtain the learned relation subgraph $\mathbf{A}'_{r_1}$. Then, the learned subgraphs are fed into a GNN and a regularizer to perform node classification with regularization. By minimizing the regularized classification loss, HGSL optimizes the graph structure and the GNN parameters jointly.

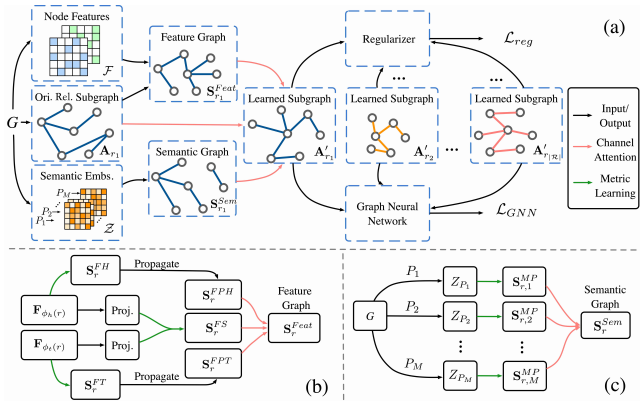



Figure 1: Overview of the HGSL framework. (a) Model framework. (b) Feature graph generator. (c) Semantic graph generator.

## Feature Graph Generator

Since the original graph may not be optimal for downstream task, a natural idea would be to augment the original graph structure via fully utilizing the rich information inside heterogeneous node features. Usually, there are two factors that affect the formation of graph structure based on features. One is the similarity between node features, and the other is the relationship between node feature and relation in HIN (Wang et al. 2020c). As shown in Figure 1 (b), we first propose to generate a feature similarity graph that captures the potential relationship generated by node features via heterogeneous feature projection and metric learning. Then we propose to generate the feature propagation graph, by propagating feature similarity matrices through topology structure. Finally, the generated feature similarity graph and feature propagation graph are aggregated to a final feature graph through a channel attention layer.

**Feature Similarity Graph** The feature similarity graph $\mathbf{S}_r^{FS}$ determines the possibility of an edge with type $r \in \mathcal{R}$ between two nodes based on node features. Specifically, for each node $v_i$ of type $\phi(v_i)$ with feature vector $\mathbf{f}_i \in \mathbb{R}^{1 \times d_{\phi(v_i)}}$, we adopt a type-specific mapping layer to project the feature $\mathbf{f}_i$ to a $d_c$-dimensional common feature $\mathbf{f}'_i \in \mathbb{R}^{1 \times d_c}$:

$$\mathbf{f}'_i = \sigma\left(\mathbf{f}_i \cdot \mathbf{W}_{\phi(v_i)} + \mathbf{b}_{\phi(v_i)}\right), \tag{1}$$

where $\sigma(\cdot)$ denotes a non-linear activation function, $\mathbf{W}_\tau \in \mathbb{R}^{d_{\phi(v_i)} \times d_c}$ and $\mathbf{b}_\tau \in \mathbb{R}^{1 \times d_c}$ denote the mapping matrix and the bias vector of type $\phi(v_i)$, respectively. Then, for a relation $r$, we perform metric learning on the common features and obtain the learned feature similarity graph $\mathbf{S}_r^{FS} \in \mathbb{R}^{|V_{\phi_h(r)}| \times |V_{\phi_t(r)}|}$, where the edge between nodes $v_i$ and $v_j$ is obtained by:

$$\mathbf{S}_r^{FS}[i, j] = \begin{cases} \Gamma_r^{FS}(\mathbf{f}'_i, \mathbf{f}'_j) & \Gamma_r^{FS}(\mathbf{f}'_i, \mathbf{f}'_j) \geq \epsilon^{FS} \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where $\epsilon^{FS} \in [0, 1]$ is the threshold that controls the sparsity of feature similarity graph, and larger $\epsilon^{FS}$ implies a more

sparse feature similarity graph. $\Gamma_r^{FS}$ is a K-head weighted cosine similarity function defined as:

$$\Gamma_r^{FS}(\mathbf{f}_i', \mathbf{f}_j') = \frac{1}{K}\sum_{k}^{K} \cos\left(\mathbf{w}_{k,r}^{FS} \odot \mathbf{f}_i', \mathbf{w}_{k,r}^{FS} \odot \mathbf{f}_j'\right), \quad (3)$$

where $\odot$ denotes the Hadamard product, and $\mathbf{W}_r^{FS} = [\mathbf{w}_{k,r}^{FS}]$ is the learnable parameter matrix of $\Gamma_r^{FS}$ that weights the importance of different dimensions of the feature vectors. By performing metric learning as in Equation 3 and ruling out edges with little feature similarity by threshold $\epsilon^{FS}$, HGSL learns the candidate feature similarity graph $\mathbf{S}_r^{FS}$.

**Feature Propagation Graph** The feature propagation graph is the underlying graph structure generated by the interaction between node features and topology structure. The key insight is that two nodes with similar features may have similar neighbors. Therefore, the process of generating feature propagation graph is two-fold: Firstly, generate the feature similarity graphs, i.e. find the similar nodes; secondly, propagate the feature similarity graph by topological structure to generate new edges, i.e. find the neighbors of the nodes with similar features.

Specifically, for each relation $r$, assume that we have two types of nodes $V_{\phi_h(r)}$ and $V_{\phi_t(r)}$ and the topology structure between them is $\mathbf{A}_r \in \mathbb{R}^{|V_{\phi_h(r)}|\times|V_{\phi_t(r)}|}$ . For the nodes $v_i, v_j \in V_{\phi_h(r)}$ with the same type $\phi_h(r)$ , we can obtain the feature similarity:

$$\mathbf{S}_r^{FH}[i,j] = \begin{cases} \Gamma_r^{FH}(\mathbf{f}_i, \mathbf{f}_j) & \Gamma_r^{FH}(\mathbf{f}_i, \mathbf{f}_j) \geq \epsilon^{FP} \\ 0 & \text{otherwise,} \end{cases} \quad (4)$$

where the threshold $\epsilon^{FP}$ controls the sparsity of feature similarity graph $\mathbf{S}_r^{FH}$. $\Gamma_r^{FH}$ is the metric learning function in the framework of Equation 3 with different parameters $\mathbf{W}_r^{FH}$. Then we can model the head feature propagation graph $\mathbf{S}_r^{FPH} \in \mathbb{R}^{|V_{\phi_h(r)}|\times|V_{\phi_t(r)}|}$ using $\mathbf{S}_r^{FH}$ and $\mathbf{A}_r$ as follows:

$$\mathbf{S}_r^{FPH} = \mathbf{S}_r^{FH}\mathbf{A}_r. \quad (5)$$

As we can see, the feature similarity is propagated through the original graph topological structure and further generates the potential feature propagation graph structure. As for the nodes $V_{\phi_t(r)}$ with the same type $\phi_t(r)$ , similar to Eq. 4, we can obtain the corresponding feature similarity graph $\mathbf{S}_r^{FT}$ with parameters $\mathbf{W}_r^{FT}$. Therefore, the corresponding feature propagation graph $\mathbf{S}_r^{FPT}$ can be obtained as follows:

$$\mathbf{S}_r^{FPT} = \mathbf{A}_r\mathbf{S}_r^{FT}. \quad (6)$$

Now, we have generated one feature similarity graph $\mathbf{S}_r^{FS}$ and two feature propagation graphs $\mathbf{S}_r^{FPH}$ and $\mathbf{S}_r^{FPT}$. The overall feature graph for relation $r$, denoted as $\mathbf{S}_r^{Feat} \in \mathbb{R}^{|V_{\phi_h(r)}|\times|V_{\phi_t(r)}|}$, can be obtained by fusing these graphs through a channel attention layer (Yun et al. 2019):

$$\mathbf{S}_r^{Feat} = \Psi_r^{Feat}([\mathbf{S}_r^{FS}, \mathbf{S}_r^{FPH}, \mathbf{S}_r^{FPT}]), \quad (7)$$

where $[\mathbf{S}_r^{FS}, \mathbf{S}_r^{FPH}, \mathbf{S}_r^{FPT}] \in \mathbb{R}^{|V_{\phi_h(r)}|\times|V_{\phi_t(r)}|\times 3}$ is the stacked matrix of the feature candidate graphs, and $\Psi_r^{Feat}$ denotes a channel attention layer with parameters $\mathbf{W}_{\Psi,r}^{Feat} \in \mathbb{R}^{1\times1\times3}$ which performs $1 \times 1$ convolution on the input using $\mathrm{softmax}(\mathbf{W}_{\Psi,r}^{Feat})$. In this way, HGSL balances the importance of each candidate feature graph for each relation $r$ by learning different weights respectively.

## Semantic Graph Generator

The semantic graph is generated depending on the high-order topology structure in HIN, describing the multi-hop structural interactions between two nodes. Notably, in heterogeneous graphs, these high-order relationships differ from each other with different semantics determined by metapaths. In light of this, we propose to learn semantic graph structures from different semantics.

Given a metapath $P$ with the corresponding relations $r_1 \circ r_2 \circ \cdots \circ r_l$, a straightforward way to generate semantic graph would be fusing the adjacency matrices, i.e. $\mathbf{A}_{r_1} \cdot \mathbf{A}_{r_2} \cdots \cdot \mathbf{A}_{r_l}$ (Yun et al. 2019). However, this method not only costs large memory with the computation of stacking multiple layers of adjacency matrices, but also discards the intermediate nodes which leads to information loss (Fu et al. 2020).

Alternatively, we propose a semantic graph generator shown in Figure 1 (c). The semantic graph generator generates the potential semantic graph structure by metric learning on trained metapath-based node embeddings. Specifically, for an interested metapath set $\mathcal{P} = \{P_1, P_2, ..., P_M\}$ with $M$ metapaths, HGSL uses trained MP2Vec (Dong, Chawla, and Swami 2017) embeddings, denoted as $\mathcal{Z} = \{\mathbf{Z}_{P_1}, \mathbf{Z}_{P_2}, \cdots, \mathbf{Z}_{P_M} \in \mathbb{R}^{|V|\times d}\}$ , to generate semantic graphs. Since the training process of semantic embeddings is off-line, the computation cost and model complexity is largely reduced. Moreover, thanks to the mechanism of heterogeneous skip-gram, the information of intermediate nodes is well preserved.

After obtaining the semantic embeddings $\mathcal{Z}$, for each metapath $P_m$ , we generate a candidate semantic subgraph adjacency matrix $\mathbf{S}_{r,m}^{MP} \in \mathbb{R}^{|V_{\phi_h(r)}|\times|V_{\phi_t(r)}|}$, where each edge is calculated by:

$$\mathbf{S}_{r,m}^{MP}[i,j] = \begin{cases} \Gamma_{r,m}^{MP}(\mathbf{z}_i^m, \mathbf{z}_j^m) & \Gamma_{r,m}^{MP}(\mathbf{z}_i^m, \mathbf{z}_j^m) \geq \epsilon^{MP} \\ 0 & \text{otherwise,} \end{cases} \quad (8)$$

where $\mathbf{z}_i^m$ stands for the $i$ th row of $\mathbf{Z}_{P_m}$, and $\Gamma_{r,m}^{MP}$ is the metric learning function with parameters $\mathbf{W}_{r,m}^{MP}$. We can see that a relation $r$ will generate $M$ candidate semantic subgraphs, so the overall semantic subgraph for relation $r$, denoted as $\mathbf{S}_r^{Sem}$, can be obtained by aggregating them:

$$\mathbf{S}_r^{Sem} = \Psi_r^{MP}([\mathbf{S}_{r,1}^{MP}, \mathbf{S}_{r,2}^{MP}, ..., \mathbf{S}_{r,M}^{MP}]) \quad (9)$$

where $[\mathbf{S}_{r,1}^{MP}, \mathbf{S}_{r,2}^{MP}, ..., \mathbf{S}_{r,M}^{MP}]$ is the stacked matrix of $M$ candidate semantic graphs. $\Psi_r^{MP}$ denotes a channel attention layer whose weight matrix $\mathbf{W}_{\Psi,r}^{MP} \in \mathbb{R}^{1\times1\times M}$ represents the importance of different metapath-based candidate graphs. After we obtain the aggregated semantic graph $\mathbf{S}_r^{Sem}$, the overall generated graph structure $\mathbf{A}_r'$ for relation $r$ can be obtained by aggregating the learned feature graph and semantic graph along with the original graph structure:

$$\mathbf{A}'_r = \Psi_r([\mathbf{S}_r^{Feat}, \mathbf{S}_r^{Sem}, \mathbf{A}_r]), \tag{10}$$

where $[\mathbf{S}_r^{Feat}, \mathbf{S}_r^{Sem}, \mathbf{A}_r] \in \mathbb{R}^{|V_{\phi_h(r)}|\times|V_{\phi_t(r)}|\times 3}$ is the stacked matrix of the candidate graphs. $\Psi_r$ is the channel attention layer whose weight matrix $\mathbf{W}_{\Psi,r} \in \mathbb{R}^{1\times1\times3}$ denotes the importance of candidate graphs in fusing the overall relation subgraph $\mathbf{A}'_r$. With a new relation adjacency matrix $\mathbf{A}'_r$ for each relation $r$, a new heterogeneous graph structure is generated, i.e. $\mathcal{A}' = \{\mathbf{A}'_r, r \in \mathcal{R}\}$.

## Optimization

In this section, we show how HGSL jointly optimizes the graph structure $\mathcal{A}'$ and the GNN parameters for downstream task. Here we focus on GCN (Kipf and Welling 2017) and node classification. Please note that, with the learned graph structure $\mathcal{A}'$, our model can be applied to other homogeneous or heterogeneous GNN methods and other tasks. A two layer GCN with parameters $\theta = (\mathbf{W}_1, \mathbf{W}_2)$ on the learned graph structure $\mathcal{A}'$, can be described as:

$$f_\theta(\mathbf{X}, \mathbf{A}') = \mathrm{softmax}\left(\hat{\mathbf{A}}\sigma\left(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}_1\right)\mathbf{W}_2\right), \tag{11}$$

where $\mathbf{X}$ is the original node feature matrix, i.e. $\mathbf{X}[i,:] = \mathbf{f}_i^T$ if the dimensions of all features are identical; otherwise, we use the common feature to construct $\mathbf{X}$, i.e. $\mathbf{X}[i,:] = \mathbf{f}_i'^T$. The adjacency matrix $\mathbf{A}'$ is constructed from the learned heterogeneous graph $\mathcal{A}'$ by considering all nodes as one type. $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}(\mathbf{A}'+\mathbf{I})\tilde{\mathbf{D}}^{-1/2}$, where $\tilde{\mathbf{D}}_{ii} = 1+\sum_j \mathbf{A}'_{ij}$. Thus, the classification loss of GNN, i.e. $\mathcal{L}_{GNN}$, on the learned graph can be obtained by:

$$\mathcal{L}_{GNN} = \sum_{v_i \in V_L} \ell\left(f_\theta(\mathbf{X}, \mathbf{A}')_i, y_i\right), \tag{12}$$

where $f_\theta(\mathbf{X}, \mathbf{A}')_i$ is the predicted label of node $v_i \in V_L$ and $\ell(\cdot,\cdot)$ measures the difference between prediction and the true label $y_i$ such as cross entropy.

Since graph structure learning methods enable the original GNN with stronger ability to fit the downstream task, it would be easier for them to over-fit. Thus, we apply regularization term $\mathcal{L}_{reg}$ to the learned graph as follows:

$$\mathcal{L}_{reg} = \alpha\|\mathbf{A}'\|_1. \tag{13}$$

This term encourages the learned graph to be sparse. The overall loss $\mathcal{L}$ can be obtained by:

$$\mathcal{L} = \mathcal{L}_{GNN} + \mathcal{L}_{reg}. \tag{14}$$

By minimizing $\mathcal{L}$, HGSL optimizes heterogeneous graph structure and the GNN parameters $\theta$ jointly towards better downstream task performance.

# Experiments

## Datasets

We employ the following real-world datasets to evaluate our proposed model. The statistics of these datasets are shown in Table 1:

- **DBLP** (Yun et al. 2019): This is a subset of DBLP which contains 4,328 papers (P), 2,957 authors (A), and 20 conferences (C). The authors are divided into four areas: database, data mining, machine learning, and information retrieval. The node features are the terms related to papers, authors and conferences respectively.
- **ACM** (Yun et al. 2019): We use the identical datasets and the experimental setting of GTN's. This dataset contains 3,025 papers (P), 5,912 authors (A), and 57 conference subjects (S). Papers are labeled according to their conferences. Node features are constructed by the keywords.
- **Yelp** (Lu et al. 2019): The Yelp dataset contains 2,614 businesses (B), 1,286 users (U), 4 services (S), and 9 rating levels (L). The business nodes are labeled by their category. The node features are constructed by the bag-of-words representation of the related keywords.

## Baselines

We compare HGSL with eleven state-of-the-art embedding methods including four homogeneous graph embedding methods, i.e., DeepWalk (Perozzi, Al-Rfou, and Skiena 2014), GCN (Kipf and Welling 2017), GAT (Velickovic et al. 2018), and GraphSAGE (Hamilton, Ying, and Leskovec 2017), four heterogeneous graph embedding methods, i.e., MP2Vec (Dong, Chawla, and Swami 2017), HAN (Wang et al. 2019b), HeGAN (Hu, Fang, and Shi 2019), and GTN (Yun et al. 2019), and three graph structure learning related methods, i.e. LDS (Franceschi et al. 2019), Pro-GNN (Jin et al. 2020), and Geom-GCN (Pei et al. 2020).

## Experimental Settings

For all GNN-related models, the number of layers are set as 2 for a fair comparison. The feature dimension in common space $d_c$ and the embedding dimension $d$ for all methods are set as 16 and 64 respectively. We choose the popular metapaths adopted in previous methods (Wang et al. 2019b; Dong, Chawla, and Swami 2017; Lu et al. 2019) for metapath based models and report the best result. For our proposed model, we use 2-head cosine similarity function defined in Equation 3, i.e. K=2. We set learning rate and weight decay as 0.01 and 0.0005 respectively. Other hyperparameters, namely $\varepsilon^{FS}$, $\varepsilon^{FP}$, $\varepsilon^{MP}$, and $\alpha$, are tuned by grid search. The code and datasets are publicly available on Github[1].

## Node Classification

In this section, we evaluate the performance of HGSL on node classification task. Macro-F1 and Micro-F1 are selected as the metrics for evaluation. The mean and standard deviation of percentage of the metric values are shown in Table 2, from which we have following observations: (1) With the capability to adaptively learn the heterogeneous graph structure, HGSL consistently outperforms all the baselines. It demonstrates the effectiveness of our proposed model. (2) Graph structure learning methods generally outperform the original GCN since it enables GCN to aggregate feature

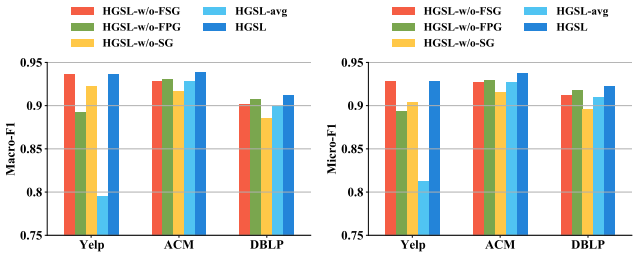
[1] https://github.com/Andy-Border/HGSL

| Dataset | # Nodes | # Edges | # Edge Type | # Features | # Training | # Validation | # Test |
|---|---|---|---|---|---|---|---|
| DBLP | 7305 | 19816 | 4 | 334 | 600 | 300 | 2057 |
| ACM | 8994 | 25922 | 4 | 1902 | 600 | 300 | 2125 |
| Yelp | 3913 | 77176 | 6 | 82 | 300 | 300 | 2014 |

Table 1: Statistics of the datasets.

| | DBLP | | ACM | | Yelp | |
|---|---|---|---|---|---|---|
| | Macro-F1 | Micro-F1 | Macro-F1 | Micro-F1 | Macro-F1 | Micro-F1 |
| DeepWalk | 88.00 ± 0.47 | 89.13 ± 0.41 | 80.65 ± 0.60 | 80.32 ± 0.61 | 68.68 ± 0.83 | 73.16 ± 0.96 |
| GCN | 83.38 ± 0.67 | 84.40 ± 0.64 | 91.32 ± 0.61 | 91.22 ± 0.64 | 82.95 ± 0.43 | 85.22 ± 0.55 |
| GAT | 77.59 ± 0.72 | 78.63 ± 0.72 | 92.96 ± 0.28 | 92.86 ± 0.29 | 84.35 ± 0.74 | 86.22 ± 0.56 |
| GraphSage | 78.37 ± 1.17 | 79.39 ± 1.17 | 91.19 ± 0.36 | 91.12 ± 0.36 | 93.06 ± 0.35 | 92.08 ± 0.31 |
| MP2Vec | 88.86 ± 0.19 | 89.98 ± 0.17 | 78.63 ± 1.11 | 78.27 ± 1.14 | 59.47 ± 0.57 | 65.11 ± 0.53 |
| HAN | 90.53 ± 0.24 | 91.47 ± 0.22 | 91.67 ± 0.39 | 91.57 ± 0.38 | 88.49 ± 1.73 | 88.78 ± 1.40 |
| HeGAN | 87.02 ± 0.37 | 88.34 ± 0.38 | 82.04 ± 0.77 | 81.80 ± 0.79 | 62.41 ± 0.76 | 68.17 ± 0.79 |
| GTN | 90.42 ± 1.29 | 91.41 ± 1.09 | 91.91 ± 0.58 | 91.78 ± 0.59 | 92.84 ± 0.28 | 92.19 ± 0.29 |
| LDS | 75.65 ± 0.20 | 76.63 ± 0.18 | 92.14 ± 0.16 | 92.07 ± 0.15 | 85.05 ± 0.16 | 86.05 ± 0.50 |
| Pro-GNN | 89.20 ± 0.15 | 90.28 ± 0.16 | 91.62 ± 1.28 | 91.55 ± 1.31 | 74.12 ± 2.03 | 77.45 ± 2.12 |
| Geom-GCN | 79.43 ± 1.01 | 80.94 ± 1.06 | 70.20 ± 1.23 | 70.00 ± 1.06 | 84.28 ± 0.70 | 85.36 ± 0.60 |
| HGSL | **91.92 ± 0.11** | **92.77 ± 0.11** | **93.48 ± 0.59** | **93.37 ± 0.59** | **93.55 ± 0.52** | **92.76 ± 0.60** |

Table 2: Performance evaluation of node classification (mean in percentage ± standard deviation).




(a) Macro-F1 (b) Micro-F1

Figure 2: Performance evaluation of variants of HGSL.

from the learned structure. (3) HGNN methods, i.e. HAN, GTN, and HGSL achieve better performance compared to GNNs since the heterogeneity is addressed. (4) GNN-based methods mostly outperform random walk-based graph embedding methods since the node features are utilized. This phenomenon becomes more obvious when it comes to Yelp dataset, since the node features, i.e. keywords, are helpful in classifying business categories.

## Ablation Study

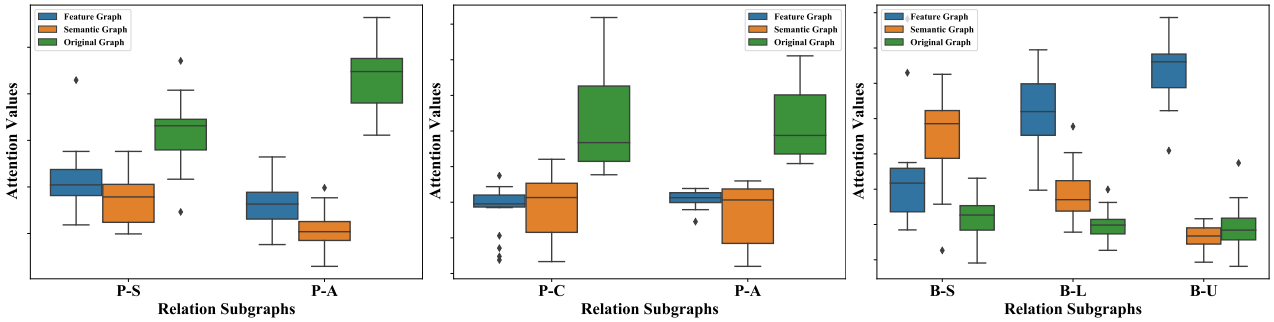
In order to verify the effectiveness of different parts of HGSL, we design four variants of HGSL and compare their classification performance against HGSL. The results in terms of Macro-F1 and Micro-F1 are shown in Figure 2 (a) and Figure 2 (b) respectively.

**Effectiveness of Candidate Graphs** HGSL generates new graph structure via fusing three kinds of candidate graphs, i.e., feature similarity graphs, feature propagation graphs, and semantic graphs. To understand their impact, we design three variants by removing each of these type of graphs from HGSL, denoted as HGSL-w/o-FSG, HGSL-w/o-FPG, and HGSL-w/o-SG, respectively. We can observe that HGSL outperforms these variants, indicating that it is necessary to consider all these candidate graphs. Moreover, compared to HGSL, the performance drop of these three variants are different on different datasets, showing that the importance of these candidate graphs differs in different cases and should be carefully balanced.

**Effectiveness of Weight Learning** To evaluate whether HGSL can effectively learn the importance of different graphs, we replace each channel attention layer in HGSL with an average aggregation layer, i.e. the graphs are obtained by averaging all candidate graphs, denoted as HGSL-avg. It is obvious that HGSL outperforms HGSL-avg significantly, which demonstrates the effectiveness of weight learning through channel attention layers. Notably, the performance of HGSL-avg on Yelp drops significantly compared to HGSL. This is because that the node features of Yelp are very important, however, HGSL-avg equally fuses three candidate graphs, where two graphs (the semantic graph and the original graph) are both generated from topology. Therefore, the impact of node features in Yelp is largely weakened, hindering the performance.

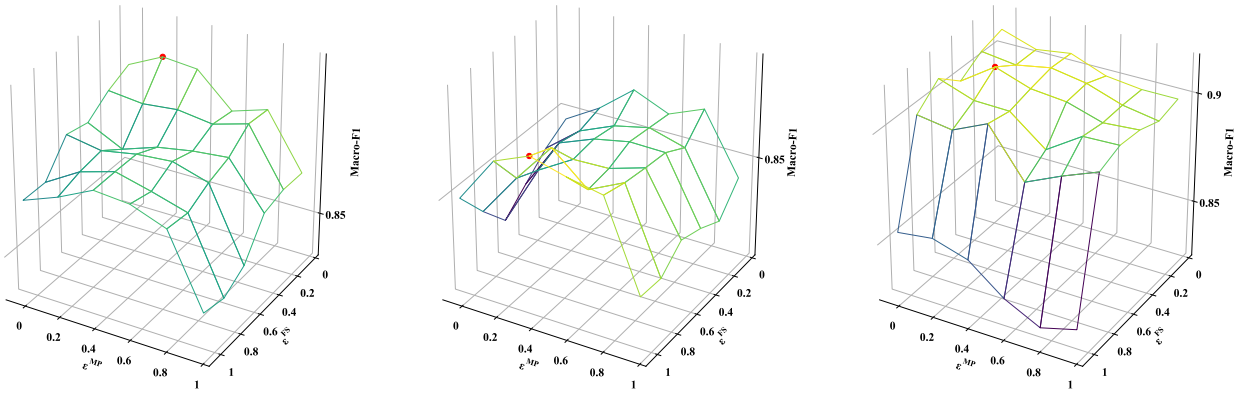
## Importance Analysis of Candidate Graphs

In order to investigate whether HGSL can distinguish the importance of candidate graphs, we analyze the weight distribution of the channel attention layer for fusing each relation subgraphs, i.e. the weights of $\Psi_r$ in Equation 10, on three datasets. We train HGSL 20 times and set all the thresholds of HGSL as 0.2. The attention distributions are shown in Figure 3. As we can observe, for relation sub-




(a) ACM (b) DBLP (c) Yelp

Figure 3: Channel attention distributions of relation subgraphs.




(a) ACM (b) DBLP (c) Yelp

Figure 4: Parameter sensitivity of different thresholds.

graphs in ACM and DBLP, the original graph structure is the most important structure for GNN-based classification. However, as for Yelp, the channel attention values of different relation subgraphs differ from each other. Specifically, for B-U (business-user) and B-L (business-rating level) relation subgraphs, the feature graphs are assigned with large channel attention value in graph structure learning. This phenomenon implies that the information in node features plays a more important role than that of semantic embeddings which agrees with the the previously discussed experiments and further demonstrates the capability of HGSL in adaptively learning a larger channel attention value for more important information.

## Parameter Analysis

In this section, we investigate the sensitivity of the important parameters. The main parameters of HGSL are the similarity thresholds, namely $\varepsilon^{FS}$, $\varepsilon^{FP}$, and $\varepsilon^{MP}$, defined in Equation 2, Equation 4, and Equation 8 respectively. These thresholds control the sparsity of the generated graphs.

For better visualization, we set $\varepsilon^{FS} = \varepsilon^{FP}$ and plot the Macro-F1 value with respect to different $\varepsilon^{FS}$ and $\varepsilon^{MP}$. The results are shown in Figure 4, from which we can observe that: For all datasets, while feature threshold ($\varepsilon^{FS} = \varepsilon^{FP}$) and semantic threshold ($\varepsilon^{MP}$) are set as 1, the performance of HGSL drops sharply. This is because in this case, HGSL utilizes the original graph structure only and the model is degenerated into a vanilla GNN model. The significant performance drop clearly demonstrates the effectiveness of graph structure learning. What's more, the performance trends for different thresholds differ greatly, which indicates that the importance of features and semantics varies in different graphs and need to be carefully evaluated.

## Conclusion

In this paper, we make the first attempt to study heterogeneous graph structure learning for GNNs and propose a framework named HGSL which jointly learns the heterogeneous graph structure and the GNN parameters towards classification objective. Particularly, by utilizing the complex interactions inside heterogeneous graphs, feature similarity graphs, feature propagation graphs and semantic graphs are generated and fused to learn an optimal heterogeneous graph structure for classification. Extensive experiments including node classification, ablation study, and model analysis are conducted, which well demonstrate the effectiveness of the proposed framework.

## Acknowledgments

J. Zhao, C. Shi, and X. Wang's work is supported by the National Natural Science Foundation of China (No. U1936220, 61772082, 61702296, 62002029, U1936104, 61972442). Y. Ye's work is partially supported by the NSF under grants IIS-2027127, IIS-2040144, IIS-1951504, CNS-2034470, CNS-1940859, CNS-1946327, CNS-1814825, OAC-1940855 and ECCS-2026612, the DoJ/NIJ under grant NIJ 2018-75-CX-0032. This work is also supported by Ant Group.

## References

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *ICLR*.

Chen, J.; Ma, T.; and Xiao, C. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*.

Chen, Y.; Wu, L.; and Zaki, M. J. 2019. Deep iterative and adaptive learning for graph neural networks. *arXiv preprint arXiv:1912.07832* .

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS*, 3837–3845.

Dong, Y.; Chawla, N. V.; and Swami, A. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*, 135–144.

Duvenaud, D.; Maclaurin, D.; Aguilera-Iparraguirre, J.; Gómez-Bombarelli, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. P. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *NIPS*, 2224–2232.

Fan, S.; Zhu, J.; Han, X.; Shi, C.; Hu, L.; Ma, B.; and Li, Y. 2019a. Metapath-guided Heterogeneous Graph Neural Network for Intent Recommendation. In *KDD*, 2478–2486. ACM.

Fan, W.; Ma, Y.; Li, Q.; He, Y.; Zhao, Y. E.; Tang, J.; and Yin, D. 2019b. Graph Neural Networks for Social Recommendation. In *WWW*, 417–426.

Franceschi, L.; Niepert, M.; Pontil, M.; and He, X. 2019. Learning Discrete Structures for Graph Neural Networks. In *ICML*, 1972–1982.

Fu, X.; Zhang, J.; Meng, Z.; and King, I. 2020. MAGNN: Metapath Aggregated Graph Neural Network for Heterogeneous Graph Embedding. In *WWW*, 2331–2341.

Hamilton, W. L.; Ying, Z.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*, 1024–1034.

Hong, H.; Guo, H.; Lin, Y.; Yang, X.; Li, Z.; and Ye, J. 2020. An Attention-Based Graph Neural Network for Heterogeneous Structural Learning. In *AAAI*, 4132–4139.

Hu, B.; Fang, Y.; and Shi, C. 2019. Adversarial Learning on Heterogeneous Information Networks. In *KDD*, 120–129.

Hu, B.; Zhang, Z.; Shi, C.; Zhou, J.; Li, X.; and Qi, Y. 2019a. Cash-Out User Detection Based on Attributed Heterogeneous Information Network with a Hierarchical Attention Mechanism. In *AAAI*, 946–953.

Hu, L.; Yang, T.; Shi, C.; Ji, H.; and Li, X. 2019b. Heterogeneous Graph Attention Networks for Semi-supervised Short Text Classification. In *EMNLP-IJCNLP*, 4820–4829.

Hu, Z.; Dong, Y.; Wang, K.; and Sun, Y. 2020. Heterogeneous Graph Transformer. In *WWW*, 2704–2710.

Jiang, B.; Zhang, Z.; Lin, D.; Tang, J.; and Luo, B. 2019. Semi-Supervised Learning With Graph Learning-Convolutional Networks. In *CVPR*, 11313–11320.

Jin, W.; Ma, Y.; Liu, X.; Tang, X.; Wang, S.; and Tang, J. 2020. Graph Structure Learning for Robust Graph Neural Networks. In *KDD*, 66–74.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

Lee, J.; Lee, I.; and Kang, J. 2019. Self-Attention Graph Pooling. In *ICML*, 3734–3743.

Li, A.; Qin, Z.; Liu, R.; Yang, Y.; and Li, D. 2019. Spam Review Detection with Graph Convolutional Networks. In *CIKM*, 2703–2711.

Lu, Y.; Shi, C.; Hu, L.; and Liu, Z. 2019. Relation Structure-Aware Heterogeneous Information Network Embedding. In *AAAI*, 4456–4463.

Pei, H.; Wei, B.; Chang, K. C.; Lei, Y.; and Yang, B. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *ICLR*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *KDD*, 701–710.

Shi, C.; Li, Y.; Zhang, J.; Sun, Y.; and Yu, P. S. 2017. A Survey of Heterogeneous Information Network Analysis. *IEEE Trans. Knowl. Data Eng.* 29(1): 17–37.

Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *ICLR*.

Wang, X.; Bo, D.; Shi, C.; Fan, S.; Ye, Y.; and Yu, P. S. 2020a. A Survey on Heterogeneous Graph Embedding: Methods, Techniques, Applications and Sources.

Wang, X.; He, X.; Wang, M.; Feng, F.; and Chua, T. 2019a. Neural Graph Collaborative Filtering. In *SIGIR*, 165–174.

Wang, X.; Ji, H.; Shi, C.; Wang, B.; Ye, Y.; Cui, P.; and Yu, P. S. 2019b. Heterogeneous Graph Attention Network. In *WWW*, 2022–2032.

Wang, X.; Lu, Y.; Shi, C.; Wang, R.; Cui, P.; and Mou, S. 2020b. Dynamic Heterogeneous Information Network Embedding with Meta-path based Proximity. *IEEE Transactions on Knowledge and Data Engineering* .

Wang, X.; Zhu, M.; Bo, D.; Cui, P.; Shi, C.; and Pei, J. 2020c. AM-GCN: Adaptive Multi-channel Graph Convolutional Networks. In Gupta, R.; Liu, Y.; Tang, J.; and Prakash, B. A., eds., *KDD*.

Wu, F.; Jr., A. H. S.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. Q. 2019a. Simplifying Graph Convolutional Networks. In *ICML*, 6861–6871.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2019b. A Comprehensive Survey on Graph Neural Networks. *CoRR* abs/1901.00596.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How Powerful are Graph Neural Networks? In *ICLR*.

Ye, Y.; and Ji, S. 2019. Sparse Graph Attention Networks. *CoRR* abs/1912.00552.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*, 974–983.

Yun, S.; Jeong, M.; Kim, R.; Kang, J.; and Kim, H. J. 2019. Graph Transformer Networks. In *NIPS*, 11960–11970.

Zhang, C.; Song, D.; Huang, C.; Swami, A.; and Chawla, N. V. 2019a. Heterogeneous Graph Neural Network. In *KDD*, 793–803.

Zhang, C.; Swami, A.; and Chawla, N. V. 2019. SHNE: Representation Learning for Semantic-Associated Heterogeneous Networks. In *WSDM*, 690–698.

Zhang, Y.; Pal, S.; Coates, M.; and Üstebay, D. 2019b. Bayesian Graph Convolutional Neural Networks for Semi-Supervised Classification. In *AAAI*, 5829–5836.

Zhang, Y.; Xiong, Y.; Kong, X.; Li, S.; Mi, J.; and Zhu, Y. 2018. Deep Collective Classification in Heterogeneous Information Networks. In *WWW*, 399–408.

Zhang, Z.; Cui, P.; and Zhu, W. 2018. Deep Learning on Graphs: A Survey. *CoRR* abs/1812.04202.

Zhao, J.; Wang, X.; Shi, C.; Liu, Z.; and Ye, Y. 2020a. Network Schema Preserving Heterogeneous Information Network Embedding. In *IJCAI*, 1366–1372.

Zhao, K.; Bai, T.; Wu, B.; Wang, B.; Zhang, Y.; Yang, Y.; and Nie, J. 2020b. Deep Adversarial Completion for Sparse Heterogeneous Information Network Embedding. In *WWW*.

Zheng, C.; Zong, B.; Cheng, W.; Song, D.; Ni, J.; Yu, W.; Chen, H.; and Wang, W. 2020. Robust Graph Representation Learning via Neural Sparsification. In *ICML*.