A Novel Composite Graph Neural Network

Zhaogeng Liu, Jielong Yang, Xionghu Zhong, Wenwu Wang, *Senior Member, IEEE,* Hechang Chen, and Yi Chang, *Senior Member, IEEE*

***Abstract*—Graph Neural Networks (GNNs) have achieved great success in many fields due to their powerful capabilities of processing graph-structured data. However, most GNNs can only apply to scenarios where graphs are known, but real-world data are often noisy or even do not have available graph structures. Recently, graph learning has attracted increasing attention in dealing with these problems. In this paper, we develop a novel approach to improving the robustness of the GNNs, called Composite Graph Neural Network. Different from existing methods, our method uses composite graphs to characterize both sample and feature relations. The composite graph is a unified graph that unifies these two kinds of relations, where edges between samples represent sample similarities, and each sample has a tree-based feature graph to model feature importance and combination preference. By jointly learning multi-aspect composite graphs and neural network parameters, our method improves the performance of semi-supervised node classification and ensures robustness. We conduct a series of experiments to evaluate the performance of our method and the variants of our method that only learn sample relations or feature relations. Extensive experimental results on nine benchmark datasets demonstrate that our proposed method achieves the best performance on almost all the datasets and is robust to feature noises.**

***Index Terms*—Composite graph, Tree-based feature graph, Sample graph, Graph neural networks.**

I. Introduction

In recent years, Graph Neural Networks (GNNs) have been successfully used to deal with graph-structured data [1], [2], and have been widely applied in many domains such as computer vision systems [3], [4], natural language processing [5], [6], neural-symbolic computing [7], [8], and COVID-19 prevention and control [9], [10]. The good performance of GNNs depends on their capabilities of utilizing data graph structure [11]. Many GNNs are developed to aggregate node information along edges of given sample graphs (the graphs whose nodes are samples and the edges are the relations between samples), such as Graph Convolutional Network (GCN) [12] and Graph Attention Network (GAT) [13]. These information aggregation methods are also further improved in recently published works [14], [15]. However, most of these methods do not learn the graph structure of samples (i.e., the sample graph) and thus can be hardly applied to real scenarios where graphs are noisy or even unavailable. Many link prediction methods proposed in the literature can learn sample graphs [16], [17], but these methods either are difficult to be directly incorporated into GNNs or can not jointly learn neural network parameters and sample graphs.

Jointly learning the graph structure of samples and the parameters of a neural network has shown to be important to the robustness of GNNs [18], [19]. To this end, some GNN methods capable of simultaneously learning or refining the graph structure of samples and neural network parameters are developed [19], [20]. However, none of them consider learning feature graphs (the graphs whose nodes are features and edges are feature relationships), thereby incapable of capturing the structural information between features. Similar to samples, features can also exhibit multifarious structures, such as tree structures [21], [22] and graph structures [23], [24]. The structures of features have been proved to be very important in many real-world applications [25], [26].

Recently, some studies noticed that learning the unknown graph structure of features can help improve the performance of GNNs, such as Fi-GNN [27] and CatGCN [28]. These methods consider learning the feature graphs, but the sample graphs are required to be known, which hinders their applications in scenarios where the sample graphs are inaccessible. High-quality sample relations and feature relations depend on each other, and thus should be jointly learned instead of being learned at two stages. To this end, we need to use a graph to characterize both sample relations and feature relations. Both feature and sample relations can be learned from the Euclidean distance or the cosine similarity of each pair of samples and features. However, pairwise similarities (distances) can neither select important features, nor obtain important feature combinations, which have been shown to be important for feature relationship learning [28], [29]. For example, men aged 20-25 are more likely to be digital enthusiasts [28]. This gender-age feature combination is more discriminating than either gender or age alone for judging digital enthusiasts.

This work was supported in part by the Young Scientists Fund of the National Natural Science Foundation of China under Grant 62106082, in part by the National Natural Science Foundation of China under Grant 61976102, Grant U19A2065, and Grant 61971186, in part by the International Cooperation Project of Jilin Province under Grant 20220402009GH. *(Corresponding authors: Jielong Yang and Hechang Chen.)*

Zhaogeng Liu and Jielong Yang are with the School of Artificial Intelligence, Jilin University, Changchun 130012, China, and also with the Engineering Research Center of Knowledge-Driven Human-Machine Intelligence, Ministry of Education, Jilin University, Changchun 130012, China (e-mail: zgliu20@mails.jlu.edu.cn; JYANG022@e.ntu.edu.sg).

Xionghu Zhong is with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China (e-mail: xzhong@hnu.edu.cn).

Wenwu Wang is with the Department of Electrical and Electronic Engineering, University of Surrey, GU2 7XH Guildford, U.K. (e-mail: w.wang@surrey.ac.uk).

Hechang Chen and Yi Chang are with the School of Artificial Intelligence, Jilin University, Changchun 130012, China, also with the Engineering Research Center of Knowledge-Driven Human-Machine Intelligence, Ministry of Education, Jilin University, Changchun 130012, China, and also with the Key Laboratory of Symbolic Computation and Knowledge Engineering, Ministry of Education, Jilin University, Changchun 130012, China (e-mail: chenhc@jlu.edu.cn; yichang@jlu.edu.cn).

0000–0000/00$00.00 © 2021 IEEE

Sample relations usually represent the pairwise similarity between samples, while feature relations need to show their combination preference. Hence, it is a natural way to model sample relations with a sample graph, and on each node, feature relations should be modeled with trees.

Thus, a natural question arises: *How to model a graph of samples and trees of features with a unified graph and jointly learn this unified graph and the neural network parameters?*

We propose a Composite Graph Neural Network (CGNN) to learn and utilize composite graphs (C-graphs) for node embeddings. First, we propose a new type of feature graph called the tree-based feature graph. When using features to build a tree in CGNN, the important features are selected as nodes, and the combination of two connected features is preferred. The reasons are because (i) a feature tree's hierarchy corresponds to how informative each feature is, with more informative features closer to the tree's root node [30], and (ii) the features used to build a tree are more closely related to one another than the ones not used [31]. Then, we propose the composite graph, which uses a sample graph to model the sample similarities and a tree-based feature graph to model the feature relations of each sample. Finally, we propose a C-graph Transformation Operation to aggregate features using the C-graph. In our method, both sample relations and feature relations can be jointly updated by minimizing the same loss function, which enables us to utilize their dependencies and obtain a better composite graph.

To the best of our knowledge, no previous studies discuss the unified modeling of the tree structure of features and the graph structure of samples. We highlight the contributions and the advantages of our work as follows:

a **Unifying the tree-based feature graph and the sample graph into a composite graph (C-graph).** The composite graph reveals both sample similarities and feature combination preferences.

b **Proposing a C-graph Transformation Operation to aggregate features using the C-graph.** This operation jointly uses sample and feature relations to update features.

c **Proposing a Composite Graph Neural Network (CGNN) for semi-supervised node classification problems.** We develop a C-graph based neural network method, which can jointly learn C-graphs from different aspects and neural network parameters and thus can be directly used to handle *semi-supervised problems without knowing graphs a priori.*

In addition, extensive validations on different datasets demonstrate that CGNN achieves better performance than state-of-the-art methods in semi-supervised classification accuracy, regardless of whether the sample graph is known or not. Moreover, additional experiments show that CGNN is robust to feature noises even when the sample graph is unknown.

The remainder of this paper is organized as follows. Section II reviews the related work. In Section III, we propose our framework CGNN that uses C-graphs to deal with semi-supervised problems and give the details of jointly learning the graph structure of samples and the tree structure of features to obtain C-graph. Comprehensive evaluation and experimental result analysis are conducted in Section IV. Finally, conclusion is given in Section V.

Notations: In this paper, we use boldfaced characters to represent vectors and matrices. Suppose that $\mathbf{M}$ is a matrix, then $\mathbf{M}[n,\cdot]$, $\mathbf{M}[\cdot,f]$, and $\mathbf{M}[n,f]$ denote its n-th row, f-th column, and (n,f)-th element, respectively. A vector $\mathbf{x}=(x_1,x_2,\ldots,x_N)$ is abbreviated as $(x[i])_{i=1}^{N}$ if the index set that i runs over is clear from the context, and the i-th element of $\mathbf{x}$ is $\mathbf{x}[i]$. We summarize the important notations used throughout the paper in Table I, where $\mathbf{X}$ is a matrix of node feature vectors, and the feature vector of each node is called a sample. We use each row in $\mathbf{X}$ to denote a sample, each column in $\mathbf{X}$ to denote a feature, and $\boldsymbol{\alpha}$, $\tilde{\boldsymbol{\alpha}}$, $\boldsymbol{\Theta}_1$, and $\boldsymbol{\Theta}_2$ to denote the neural network parameters.

TABLE I
SUMMARY OF IMPORTANT NOTATIONS.

Symbol	Meaning
$\mathbb{R}$	the set of real numbers
N	the number of samples
F	the number of features
C	the number of classes
R	the number of submodules
$\mathbf{0}$	the C-dimensional zero vector
$\mathbf{1}^\top$	the N-dimensional all-one column vector
$\mathbf{I}\in\mathbb{R}^{N\times N}$	the identity matrix
$\mathbf{X}\in\mathbb{R}^{N\times F}$	the input dataset
$\mathbf{Y}\in\mathbb{R}^{N\times C}$	the one-hot labels
$\mathring{\mathbf{Y}}\in\mathbb{R}^{N\times C}$	the pseudo labels
$\mathbf{A_S}\in\mathbb{R}^{N\times N}$	the adjacency matrix of the sample graph
$\mathbf{A_F}\in\mathbb{R}^{F\times F}$	the adjacency matrix of the tree-based feature graph
$\mathcal{G}$	the composite graph (C-graph)
$\mathbf{A}_\mathcal{G}=\{\mathbf{A_S},\mathbf{A_F}\}$	the adjacency matrix set of $\mathcal{G}$
$\boldsymbol{\alpha}\in\mathbb{R}^{F+C}$, $\tilde{\boldsymbol{\alpha}}\in\mathbb{R}^{F}$	the trainable vectors
$\boldsymbol{\Theta}_1$, $\boldsymbol{\Theta}_2$	the trainable weight matrices
$\odot$	element-wise product
ℓ	a differentiable error function
σ	a non-linear activation function

II. Related Work

Many graph-based methods have been proposed for relational data with known graph structures, such as semi-supervised embedding (SemiEmb) [32], DeepWalk [33], and LINE [34]. Among these methods, graph embedding based method receives considerable attention. In [35], a method named Planetoid is proposed, which trains an embedding for each sample by jointly predicting the class label and capturing structural information in the given sample graph. However, Planetoid can not fully embed all the graph structure and label information limited by its sampling strategy [36]. Recently, GNNs closely related to graph embedding have attracted increasing interest in the field of machine learning and data mining [37]. In [12] and [13], the well-known GCN and GAT are proposed, which directly aggregate neighbor information using graph convolution and graph attention, respectively. Inspired by GCN and GAT, many GNNs with better performances are proposed, such as GCNII [14], AdaGCN [38], and BGAT-T [39].

A foundational assumption of the previous methods is that the given graph structure of samples is of high quality, limiting

their application in scenarios with unknown or noisy graph structures. Recently, graph learning has drawn significant attention in dealing with these problems [40], [41]. Some link prediction methods and GNN methods can extend a sample graph based on some prior known structural relationships between nodes [42], [43], but these methods cannot learn the sample graph and neural network parameters simultaneously. Recently, some methods that can jointly learn the sample graph and neural network parameters have been proposed. kNN-LDS gives a bilevel programming approach to jointly learn the sample graph and a neural network [19]. DIAL-GNN iteratively learns the parameters of a neural network and augments the given graph structure [44]. Graphite proposed in literature [45] uses a deep latent variable generative model to learn a sample graph and the sample graph can be iteratively refined via a message passing operation. Pro-GNN simultaneously learns the parameters of a neural network and a robust graph structure of samples by exploring graph properties of sparsity, low rank, and feature smoothness [18]. A GCN method proposed in literature [46] learns the graph and graph representation from the low-dimensional space of the original data simultaneously by using a new multigraph fusion method. Peng *et al.* [47] proposed a GNN method that learns the graph structure from the given data points by using reverse graph learning and out-of-sample extension strategies.

Although the aforementioned methods jointly learn sample graph and neural network parameters, they do not consider the unknown underlying relations between features. However, in many applications, the hierarchical structure of features and the combination of related features are essential. In our method, we jointly learn the graph structure of samples and the tree structure of features, which further compose the composite graph. Hence, our proposed composite graph can characterize the similarity between samples, relative importance between features, and combination preference of features. However, it is still an unresolved problem in the literature to learn and utilize the C-graph. Next, we will provide the details of our method in Section III.

III. Composite Graph Neural Network

In this section, we propose graph neural network method CGNN to address the challenging problem where data samples influence each other through edges of multiple unknown graphs and these edges are closely related to the features of samples. CGNN is a unified framework to learn a generalized graph, C-graph, capable of demonstrating both sample relationships and feature relationships. Specifically, CGNN is proposed to

a Learn a C-graph of features and samples, which characterizes the similarity between samples, the hierarchical structure of features and the combination preference of related features.
b Learn multiple C-graphs from different aspects using different submodules.
c Jointly learn the C-graphs and neural network parameters for semi-supervised tasks. The joint learning of sample relations and neural network parameters is shown to be essential to the robustness of GNNs.

A toy example for illustrating CGNN is shown in Fig. 1. The pseudocode of CGNN is given in Algorithm 2 in Section IV.

A. Main Definitions

Consider a dataset with N samples $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^\top$. For any $n \in \{1, 2, \ldots, N\}$, $\mathbf{x}_n \in \mathbb{R}^F$ has F features. The one-hot labels of these samples are $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N]^\top$, where each element $\mathbf{y}_n \in \mathbb{R}^C$ with C being the number of classes. In this paper, we focus on the semi-supervised classification problem where $\mathbf{X}$ and a subset of $\mathbf{Y}$ are used to learn a model to predict the remaining unknown subset of $\mathbf{Y}$. Incorporating the relationship between samples (i.e., the sample graph) has shown great potential in semi-supervised classification problems [12]. Most GNNs can only deal with the case where graphs are known and some other GNNs capable of learning graphs only consider the sample relations. In this paper, we consider a composite graph characterizing both the sample relations and feature relations.

Definition 1 (Composite Graph)**.** *In this paper, the composite graph (C-graph) is defined as* $\mathcal{G} \triangleq (\mathbf{S}, \mathbf{E_S}, \mathbf{E_F})$*, where* $\mathbf{S}$*,* $\mathbf{E_S}$ *and* $\mathbf{E_F}$ *are the sample node set, the sample edge set, and the feature edge set, respectively. We define the adjacency matrix set of* $\mathcal{G}$ *as* $\mathbf{A}_\mathcal{G} \triangleq \{\mathbf{A_S}, \mathbf{A_F}\}$*, where* $\mathbf{A_S}$ *and* $\mathbf{A_F}$ *are the adjacency matrices of the sample graph and the tree-based feature graph, respectively. We say* $\mathbf{A_S}$ *and* $\mathbf{A_F}$ *are unified into a composite graph since they are simultaneously used to update features in Algorithm 1, jointly learned by optimizing the same loss function* $\ell(\mathbf{Y}, \widehat{\mathbf{Y}})$*, and depend on each other in the update process (see Section III-C3).*

Given a dataset $\mathbf{X}$ and a known subset of $\mathbf{Y}$, we aim at learning the composite graph $\mathbf{A}_\mathcal{G} = \{\mathbf{A_S}, \mathbf{A_F}\}$ and use the learned graph to obtain node embeddings and predicted labels. It is worth noting that $\mathbf{A_S}$ and $\mathbf{A_F}$ are closely related to each other and are jointly learned in a unified framework. In the following sections, $\mathbf{D_S}$ denotes an $N \times N$ diagonal matrix with $\mathbf{D_S}[n, n] \triangleq 1 + \sum_{p=1}^{n} \mathbf{A_S}[n, p]$ for each $n \in \{1, 2, \ldots, N\}$. $\sigma(\cdot)$ denotes an activation function. $\ell(\mathbf{Y}, \widehat{\mathbf{Y}})$ denotes a differentiable function that measures the difference between the target $\mathbf{Y}$ and the prediction $\widehat{\mathbf{Y}}$, and we use the cross-entropy error function in our experiments.

B. General Framework

In this section, we present the general framework of CGNN. Our framework can jointly learn C-graphs from different aspects and neural network parameters. The learned C-graphs can reveal both the underline structure information of sample similarities and feature combination preferences for helping obtain better node embeddings and deal with semi-supervised problems without knowing graphs a priori. Let R be the number of submodules in our method. In Fig. 1, each blue dotted box denotes a submodule, and there are R submodules in total. The following formulas (1), (2), and (3) are repeated for each submodule $r \in \{1, 2, \ldots, R\}$, and thus in these formulas, we omit submodule index r. Firstly, we use a C-graph transformation operation to embed $\mathbf{X}$ with the C-graph,

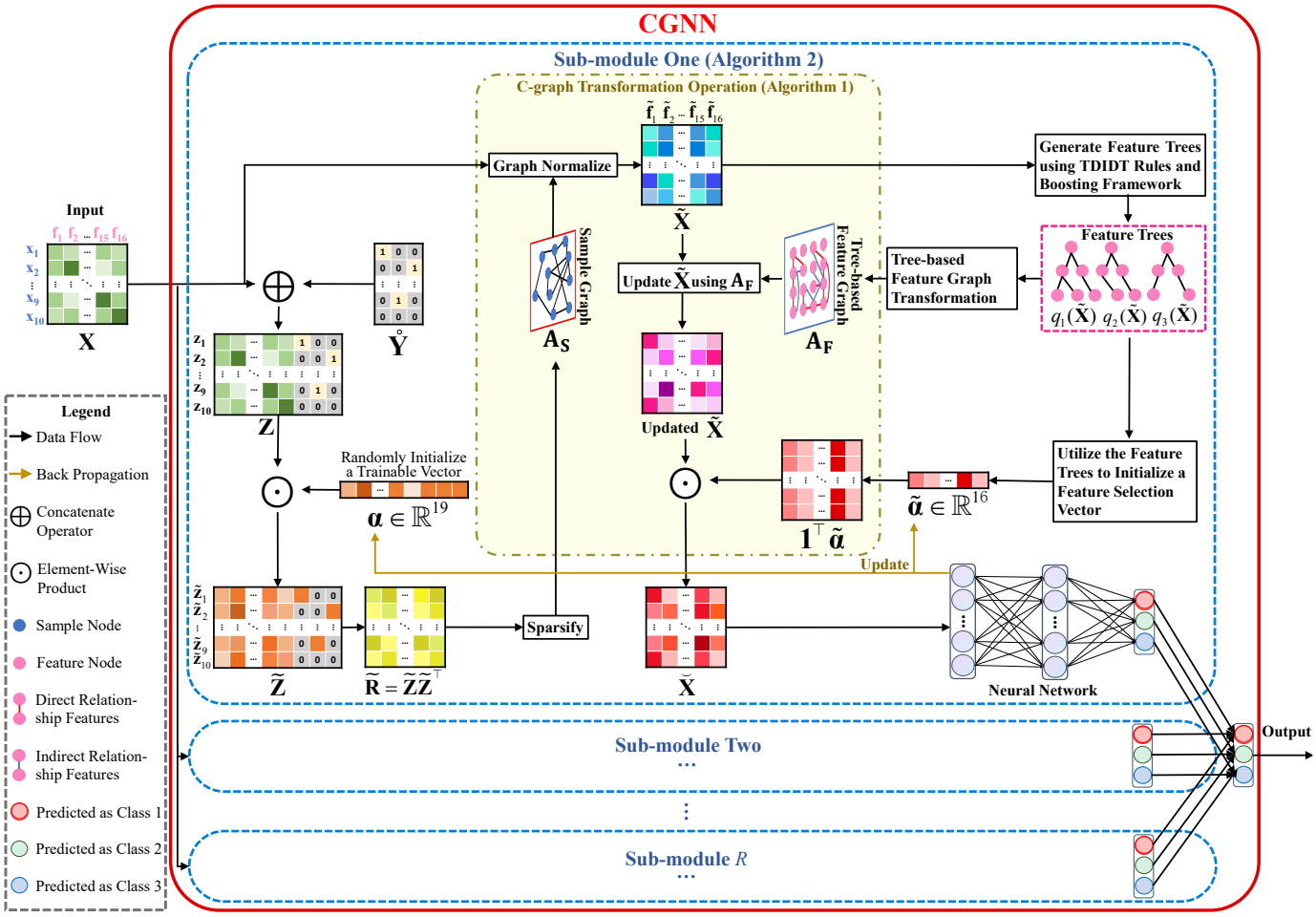


Fig. 1. A toy example of CGNN with R submodules for a three-class classification of input $\mathbf{X} \in \mathbb{R}^{10\times 16}$. Each blue dotted box denotes a submodule, and there are R blue dotted boxes in total. $\mathring{\mathbf{Y}} \in \mathbb{R}^{10\times 3}$ denotes the pseudo labels generated from the training labels. $\breve{\mathbf{X}}$ denotes the node embeddings obtained by learning the composite graph (C-graph) defined in Definition 1. In our method, the C-graph $\mathbf{A}_{\mathcal{G}} = \{\mathbf{A_S}, \mathbf{A_F}\}$ and the parameters of the neural network are jointly learned. In the C-graph $\mathbf{A}_{\mathcal{G}}$, $\mathbf{A_S}$ and $\mathbf{A_F}$ depend on each other in the update process. The proposed C-graph Transformation Operation takes $\mathbf{X}$, $\mathbf{A}_{\mathcal{G}} = \{\mathbf{A_S}, \mathbf{A_F}\}$ and $\tilde{\boldsymbol{\alpha}}$ as input, and outputs $\breve{\mathbf{X}}$. Best viewed in color.

and the obtained node embeddings are further input to a fully connected hidden layer. This process is given by

$$\mathbf{X}^{(1)} = \sigma\left(\mathrm{Cgto}\left(\mathbf{X}, \mathbf{A}_{\mathcal{G}}, \tilde{\boldsymbol{\alpha}}\right)\boldsymbol{\Theta}_1\right), \tag{1}$$

where $\mathrm{Cgto}(\cdot,\cdot,\cdot)$ is the C-graph transformation operation over $\mathbf{A}_{\mathcal{G}}$, $\boldsymbol{\Theta}_1$ is a trainable $F \times H$ matrix where H denotes the number of features of the first hidden layer, and $\mathbf{X}^{(1)}$ is the output of the first hidden layer. The Cgto can not only enhance the feature weight by extracting information between neighbors but also can highlight the important features. The pseudocode of Cgto is shown in Algorithm 1. In line 6 of Algorithm 1, $\mathbf{X}$ is normalized to $\tilde{\mathbf{X}}$ by $\mathbf{A_S}$ and $\mathbf{D_S}$. From line 7 to line 16, $\tilde{\mathbf{X}}$ is further updated using $\mathbf{A_F}$. It is worth noting that in line 17, the important features will be assigned larger weights by $\tilde{\boldsymbol{\alpha}}$ in the training process, which will be illustrated in Section III-C2 in details.

Next, we use a generalized GCN layer to update $\mathbf{X}^{(1)}$ and obtain

$$\mathbf{X}^{(2)} = \sigma\left(\mathbf{D_S}^{-\frac{1}{2}}(\mathbf{A_S}+\mathbf{I})\mathbf{D_S}^{-\frac{1}{2}}\mathbf{X}^{(1)}\boldsymbol{\Theta}_2\right), \tag{2}$$

where $\boldsymbol{\Theta}_2$ is a trainable $H \times C$ matrix. Different from the GCN [12] that uses a fixed graph in the training process, in formula (2), the adjacency matrix of the samples $\mathbf{A_S}$ are trained together with $\boldsymbol{\Theta}_2$.

Finally, we use a softmax function to process $\mathbf{X}^{(2)}$ and obtain

$$\mathbf{X}^{(3)} = \mathrm{Softmax}\left(\mathbf{X}^{(2)}\right). \tag{3}$$

In our method, (1), (2), and (3) are jointly regarded as a submodule of CGNN. CGNN uses multiple submodules to learn multiple C-graphs from different aspects and thus can adapt to the case where the relations of samples and features are influenced by multiple factors.

We use R submodules and for each $r \in \{1, 2, \ldots, R\}$, the loss function of submodule r is given by

$$L_r = \ell(\mathbf{Y}, \mathbf{X}^{(3)}). \tag{4}$$

Note that the submodule index r is omitted in $\mathbf{X}^{(3)}$.

Then the loss function of CGNN is

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{R}\sum_{r=1}^{R} L_r + \Omega(\boldsymbol{\theta}), \tag{5}$$

where R is the number of submodules, Ω is a regularizer and $\boldsymbol{\theta}$ denotes the set of trainable parameters.

Algorithm 1: C-graph Transformation Operation

Input : $\mathbf{X}$, $\mathbf{A}_{\mathcal{G}} = \{\mathbf{A_S}, \mathbf{A_F}\}$, $\tilde{\boldsymbol{\alpha}}$
Output: the transformed $\breve{\mathbf{X}}$

Initialize the three hyperparameters w_C, w_D, and w_I;
Initialize $\mathbf{D_S} \in \mathbb{R}^{N\times N}$ as a zero matrix;
for *each* $n \in \{1, 2, \ldots, N\}$ **do**
  $\mathbf{D_S}[n,n] = 1 + \sum_{p=1}^{n} \mathbf{A_S}[n,p]$;
end
$\tilde{\mathbf{X}} = \mathbf{D_S}^{-\frac{1}{2}}(\mathbf{A_S} + \mathbf{I})\mathbf{D_S}^{-\frac{1}{2}}\mathbf{X}$;
for *each feature* $\tilde{\mathbf{f}}_i$ *in* $\tilde{\mathbf{X}}$ **do**
  $\tilde{\mathbf{f}}_i = w_C\tilde{\mathbf{f}}_i$;
  for *each* $f \in \{1, 2, \ldots, F\}$ **do**
    if $\mathbf{A_F}[i,f] = 2$ **then**
      $\tilde{\mathbf{f}}_i = \tilde{\mathbf{f}}_i + w_D\tilde{\mathbf{f}}_f$;
    else if $\mathbf{A_F}[i,f] = 1$ **then**
      $\tilde{\mathbf{f}}_i = \tilde{\mathbf{f}}_i + w_I\tilde{\mathbf{f}}_f$;
    end
  end
end
$\breve{\mathbf{X}} = \mathbf{1}^\top\tilde{\boldsymbol{\alpha}} \odot \tilde{\mathbf{X}}$;
return $\breve{\mathbf{X}}$;

C. Learning the C-graph

In our model, C-graphs are used for node embeddings and the proposed C-graph framework considers both the relationship between samples and the relationship between features. In this section, we first introduce how to learn the sample graph $\mathbf{A_S}$ and the tree-based feature graph $\mathbf{A_F}$. Moreover, we introduce how $\mathbf{A_S}$ and $\mathbf{A_F}$ depend on each other in the learning process.

1) Learning Sample Relations: In this section, we propose the method to learn the adjacency matrix of the sample graph $\mathbf{A_S}$. We consider the known class label as an important feature to learn the sample graph. We first construct pseudo labels $\mathring{\mathbf{Y}} = [\mathring{\mathbf{y}}_1, \mathring{\mathbf{y}}_2, \ldots, \mathring{\mathbf{y}}_N]^\top$ using the training labels by

$$\mathring{\mathbf{y}}_n = \begin{cases} \mathbf{y}_n & \text{if } \mathbf{y}_n \text{ is a training label} \\ \mathbf{0} & \text{otherwise} \end{cases}, \tag{6}$$

where $n \in \{1, 2, \ldots, N\}$, and $\mathbf{0}$ denotes the C-dimensional zero vector. Then, we concatenate $\mathbf{x}_n$ and $\mathring{\mathbf{y}}_n$, and obtain

$$\mathbf{z}_n = \mathrm{Concatenate}\left(\mathbf{x}_n, \mathring{\mathbf{y}}_n\right), \tag{7}$$

where $\mathrm{Concatenate}(\cdot,\cdot)$ denotes the concatenation operation. It should be noted that, the test labels are all-zero vectors in the train and test process. Besides, as pointed out in the literature [48], noisy features can hinder the model training process. Hence, a trainable vector $\boldsymbol{\alpha}$ is utilized to diminish the noisy features, which is given by

$$\tilde{\mathbf{z}}_n = \mathbf{z}_n \odot \boldsymbol{\alpha}. \tag{8}$$

We regard $\boldsymbol{\alpha}$ as a feature improvement operator. The effectiveness of this simple operator is also shown in [49]. Different from most feature selection methods, $\boldsymbol{\alpha}$ is trained mainly to assign different weights to different features rather than reduce the feature dimension. In other words, at the end of the training process, the smaller values of $\boldsymbol{\alpha}$ will indicate the noisy dimensions of features. Given that $\boldsymbol{\alpha}$ is initialized randomly, different $\boldsymbol{\alpha}$ can have different values and focus on different features in different submodules. From (8), a new matrix $\tilde{\mathbf{Z}} = [\tilde{\mathbf{z}}_1, \tilde{\mathbf{z}}_2, \ldots, \tilde{\mathbf{z}}_N]^\top$ can be obtained and then we compute the relationship matrix $\tilde{\mathbf{R}}$ using $\tilde{\mathbf{Z}}$:

$$\tilde{\mathbf{R}} = \tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^\top. \tag{9}$$

Next, we find the top η values in each row of $\tilde{\mathbf{R}}$, and set the others to 0 to sparsify $\tilde{\mathbf{R}}$. The sparsified $\tilde{\mathbf{R}}$ is regarded as the $\mathbf{A_S}$.

2) Learning Feature Relations: In this section, the method to learn the tree-based feature graph (i.e., $\mathbf{A_F}$) is presented. We decompose the learning of a tree-based feature graph into the learning of multiple binary decision trees. There are three main advantages of using binary decision trees: (i) features used to construct a tree are more closely related to each other than the unused features [31]; (ii) the hierarchy of features in a tree reflects their informativeness and the features closer to the root are more important [30]; and (iii) the binary decision tree structure is easy to implement and has high computational efficiency [50]. Take Fig. 2 as an example. We first learn three binary trees (see Fig. 2 (a)) and then turn them into the tree-based feature graph (see Fig. 2 (b)). The specific tree-based feature graph transformation method is as follows. First, we define two kinds of relationships: the direct relationship and the indirect relationship. The direct relationship is between each pair of directly connected nodes (e.g., the relationship between $\mathbf{f}_{11}$ and $\mathbf{f}_{16}$ in Fig. 2 (a)) and the indirect relationship is between each pair of nodes that belong to the same branch of the tree but are not directly connected (e.g., the relationship between $\mathbf{f}_{11}$ and $\mathbf{f}_{10}$ in Fig. 2 (a)).

Second, we assume that direct relationships represent stronger influence than indirect relationships and use 0, 1, and 2 to denote no relationship, an indirect relationship, and a direct relationship between each pair of features, respectively. Then we connect all the related features by finding all the direct and indirect relationships in the tree (see the transformation from Fig. 3 (a) to Fig. 3 (b)). Similarly, for multiple binary trees in Fig. 2, the same nodes and edges generated by each tree are merged. The adjacency matrix of the merged graph with values from $\{0, 1, 2\}$ is the tree-based feature graph $\mathbf{A_F}$ that we want to learn.

In this paper, multiple trees are employed to learn the tree-based feature graph since a very deep tree will be generated when the number of features is becoming larger. Thus, in our learning process, ensemble learning methods are considered [51] to generate multiple binary trees to express the relationship between many features with the purpose of

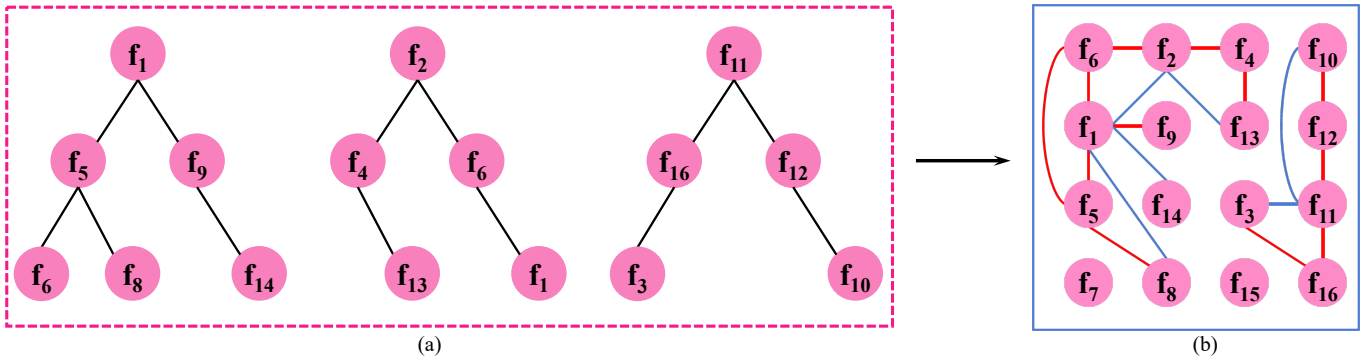


Fig. 2. An example of constructing a tree-based feature graph of 16 features from multiple binary trees. The red lines and the blue lines indicate the direct relationships and the indirect relationships, respectively.

constructing the tree-based feature graph. The main advantage of ensemble learning is that it can combine many weak models into a strong model, so each generated tree can be relatively simple. We use the boosting framework in ensemble learning to generate multiple trees and these trees are all finally merged into one graph. Here the boosting framework is used due to the following reason: In the boosting framework, the generation of the latter tree depends on the information of the former tree [52], and thus the trees generated in the framework of boosting have stronger correlation with each other than other frameworks in ensemble learning.

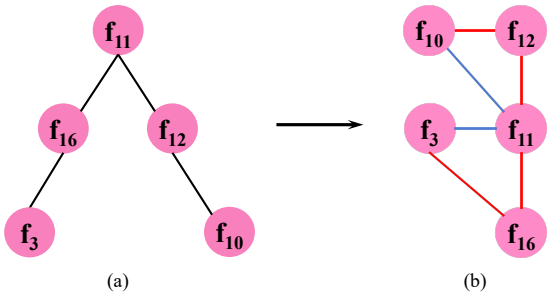


Fig. 3. An example of constructing a tree-based feature graph from a binary tree. The red lines and the blue lines indicate the direct relationships and the indirect relationships, respectively.

Specifically, we build trees according to the Top-Down Induction of Decision Trees (TDIDT) method [53], which splits the features of $\tilde{\mathbf{X}} = \mathbf{D}_{\mathbf{S}}^{-\frac{1}{2}}(\mathbf{A}_{\mathbf{S}} + \mathbf{I})\mathbf{D}_{\mathbf{S}}^{-\frac{1}{2}}\mathbf{X}$ recursively, creates the successor children by searching the greatest gain value to the predictions, and stops when the gain value does not increase. For $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_N]^\top$, our method builds trees by minimizing $\ell(\mathbf{Y}, \tilde{\mathbf{Y}})$, where $\tilde{\mathbf{Y}} = [\tilde{\mathbf{y}}_1, \tilde{\mathbf{y}}_2, \ldots, \tilde{\mathbf{y}}_N]^\top$ is the prediction results of the feature trees, and each element $\tilde{\mathbf{y}}_n \in \mathbb{R}^C$. We train the trees in an additive manner. Suppose that T is the total number of iterations for training trees; then, we build CT trees in total because C trees are generated in each iteration by minimizing $\ell(\mathbf{Y}, \tilde{\mathbf{Y}})$. Thus, minimizing $\ell(\mathbf{Y}, \tilde{\mathbf{Y}})$ is equivalent to minimize

$$\begin{aligned}\ell^{(T)}\left(\mathbf{Y}, \tilde{\mathbf{Y}}^{(T)}\right) &= \sum_{n=1}^{N} l\left(\mathbf{y}_n, \tilde{\mathbf{y}}_n^{(T)}\right) \\ &= \sum_{n=1}^{N} l\left(\mathbf{y}_n, \left(\tilde{\mathbf{y}}_n^{(T)}[c]\right)_{c=1}^{C}\right),\end{aligned} \tag{10}$$

where $\ell^{(T)}(\mathbf{Y}, \tilde{\mathbf{Y}}^{(T)})$ and $\tilde{\mathbf{Y}}^{(T)} = [\tilde{\mathbf{y}}_1^{(T)}, \tilde{\mathbf{y}}_2^{(T)}, \ldots, \tilde{\mathbf{y}}_N^{(T)}]^\top$ are the loss function value and the prediction results, respectively, and l denotes a differentiable function that measures the difference between the target $\mathbf{y}_n$ and the prediction $\tilde{\mathbf{y}}_n^{(T)}$.

Let $B^{(T)}[c]$ be the c-th tree ($c \in \{1, 2, \ldots, C\}$) generated at the finial iteration. Next, we take the process of generating $B^{(T)}[c]$ as an example to introduce our approach to building feature trees. At the T-th iteration, for each $n \in \{1, 2, \ldots, N\}$, $\tilde{\mathbf{y}}_n^{(T)}[c]$ is computed by

$$\begin{aligned}\tilde{\mathbf{y}}_n^{(T)}[c] &= \tilde{\mathbf{y}}_n^{(T-1)}[c] + B^{(T)}[c]\left(\tilde{\mathbf{x}}_n\right) \\ &= \tilde{\mathbf{y}}_n^{(T-1)}[c] + \sum_{m=1}^{M^{(T)}[c]} b_m^{(T)}[c]\mathbb{1}\left(\tilde{\mathbf{x}}_n \in B_m^{(T)}[c]\right),\end{aligned} \tag{11}$$

where $B^{(T)}[c](\tilde{\mathbf{x}}_n)$ denotes the value of using $B^{(T)}[c]$ to predict $\tilde{\mathbf{x}}_n$ as the c-th class, $M^{(T)}[c]$ denotes the total number of leaves in $B^{(T)}[c]$, $B_m^{(T)}[c]$ and $b_m^{(T)}[c]$ denote the m-th leaf of $B^{(T)}[c]$ and the value of $B_m^{(T)}[c]$, respectively, and $\mathbb{1}(\cdot)$ denotes the indicator function. Notably, $\tilde{\mathbf{y}}_n^{(0)}[c] = 0$ for $n \in \{1, 2, \ldots, N\}$ and $c \in \{1, 2, \ldots, C\}$. We use second–order approximation [54] to optimize (11) and compute the optimal $b_m^{(T)}[c]$. Then we further rewrite (10) as the following regularized objective.

$$\begin{aligned}&\ell^{(T)}\left(\mathbf{Y}, \tilde{\mathbf{Y}}^{(T)}\right) + \sum_{c=1}^{C}\Omega(B^{(T)}[c]) \\ &= \sum_{n=1}^{N} l\left(\mathbf{y}_n, \tilde{\mathbf{y}}_n^{(T-1)} + \mathbf{B}^{(T)}\left(\tilde{\mathbf{x}}_n\right)\right) + \sum_{c=1}^{C}\Omega(B^{(T)}[c]) \\ &\simeq \sum_{n=1}^{N}\left[l\left(\mathbf{y}_n, \tilde{\mathbf{y}}_n^{(T-1)}\right) + \sum_{c=1}^{C}\frac{\partial l\left(\mathbf{y}_n, \tilde{\mathbf{y}}_n^{(T-1)}\right)}{\partial \tilde{\mathbf{y}}_n^{(T-1)}[c]} B^{(T)}[c]\right.\end{aligned} \tag{12}$$

$$+\frac{1}{2}\sum_{c_1=1}^{C}\sum_{c_2=1}^{C}\frac{\partial^2 l\left(\mathbf{y}_n,\tilde{\mathbf{y}}_n^{(T-1)}\right)}{\partial\tilde{\mathbf{y}}_n^{(T-1)}[c_1]\tilde{\mathbf{y}}_n^{(T-1)}[c_2]}B^{(T)}[c_1]B^{(T)}[c_2]\Bigg]+\sum_{c=1}^{C}\Omega(B^{(T)}[c]),$$

where $\Omega(B^{(T)}[c])$ denotes the regularizer on the leaves of $B^{(T)}[c]$ and $\mathbf{B}^{(T)}(\tilde{\mathbf{x}}_n)$ denotes a C-dimension vector consisting of $B^{(T)}[1](\tilde{\mathbf{x}}_n)$, $B^{(T)}[2](\tilde{\mathbf{x}}_n)$, …, $B^{(T)}[C](\tilde{\mathbf{x}}_n)$. Since $B^{(T)}[c](\tilde{\mathbf{x}}_n)=\sum_{m=1}^{M^{(T)}[c]}b_m^{(T)}[c]\mathbb{1}\left(\tilde{\mathbf{x}}_n\in B_m^{(T)}[c]\right)$, we then substitute $\sum_{m=1}^{M^{(T)}[c]}b_m^{(T)}[c]\mathbb{1}\left(\tilde{\mathbf{x}}_n\in B_m^{(T)}[c]\right)$ into (12) and transform it into an optimization problem with respect to $b_m^{(T)}[c]$. Therefore, we can train $B^{(T)}[c]$ by optimizing $b_m^{(T)}[c]$. As for how to set and split the nodes of $B^{(T)}[c]$, there are many good algorithms proposed in past literatures, such as greedy top-down approach [55], multiple classification rank method [56] and candidate node split finding approximate algorithm [57]. Since the main contribution of this paper is not to improve the node splitting of decision trees, we choose loss reduction as gain value and use the candidate node split finding approximate algorithm to conduct node selection and splitting of $B^{(T)}[c]$ after considering a trade-off between the computational efficiency and practical effect.

3) Dependencies between $\mathbf{A_S}$ and $\mathbf{A_F}$ in the Learning Process: We have presented the methods to learn $\mathbf{A_S}$ and $\mathbf{A_F}$. In the following, we will show how $\mathbf{A_S}$ and $\mathbf{A_F}$ depend on each other in the learning process. The feature enhancement in learning $\mathbf{A_S}$ can help the learning of binary trees when learning $\mathbf{A_F}$. The operator $\boldsymbol{\alpha}$ in $\mathbf{A_S}$ can enhance the important features through continuous optimization. The enhanced features can further help $\mathbf{A_F}$ construct the ensemble trees with nodes being features. Recall that in training $\mathbf{A_S}$, $\boldsymbol{\alpha}$ is randomly initialized with different values to ensure learning different $\mathbf{A_S}$ in different submodules, which can increase the diversity of trees in $\mathbf{A_F}$.

Sample graph and feature graph are jointly used to generate the node embeddings. Firstly, $\tilde{\mathbf{X}}$ is computed using $\mathbf{A_S}$ (see Section III-C2). Then for each $i\in\{1,2,\ldots,F\}$, we further update each feature (i.e., column vector) $\tilde{\mathbf{f}}_i$ in $\tilde{\mathbf{X}}$ using the learned $\mathbf{A_F}$, which is given by

$$\tilde{\mathbf{f}}_i=\sum_{j\in\mathbf{J}}w_D\tilde{\mathbf{f}}_j+\sum_{k\in\mathbf{K}}w_I\tilde{\mathbf{f}}_k+w_C\tilde{\mathbf{f}}_i, \tag{13}$$

where $\mathbf{J}$ and $\mathbf{K}$ represent the index sets of features having indirect and direct relations with $\tilde{\mathbf{f}}_i$, respectively, and w_D, w_I, w_C are three hyperparameters denoting the weight of the direct relationship, the weight of the indirect relationship, and the weight of the self-relations, respectively. Next, we perform feature selection on the updated $\tilde{\mathbf{X}}=[\tilde{\mathbf{f}}_1,\tilde{\mathbf{f}}_2,\ldots,\tilde{\mathbf{f}}_F]$ by using a new trainable row vector $\tilde{\boldsymbol{\alpha}}$.

$$\breve{\mathbf{X}}=\mathbf{1}^\top\tilde{\boldsymbol{\alpha}}\odot\tilde{\mathbf{X}}, \tag{14}$$

where $\mathbf{1}^\top$ is the N-dimensional all-one column vector and $\tilde{\boldsymbol{\alpha}}$ is initialized with feature importance values, namely $\tilde{\boldsymbol{\alpha}}[i]=\exp(v_i)$. Here $\exp(\cdot)$ denotes the exponential function, and $v_i\in[0,1]$ is the importance value of $\tilde{\mathbf{f}}_i$. Notably, $\{v_i\}_{i=1}^F$ are obtained in the process of building trees in learning $\mathbf{A_F}$.

The obtained $\breve{\mathbf{X}}$ is further updated in (2) using $\mathbf{A_S}$.

D. Model Training and Pseudocode of CGNN

It is worth noting that the C-graph transformation operation in (1) is non-differentiable for $\mathbf{A_F}$. Thus, the three variables $\mathbf{A_F}$, $\boldsymbol{\theta}$ and $\mathbf{A_S}$ share the optimization objective, but have different update method. Specifically, we use ensemble tree model to update $\mathbf{A_F}$, as shown in (11)-(12). It can be seen that in formulas (1)-(5) the loss function is differentiable with respect to $\boldsymbol{\theta}$ and $\mathbf{A_S}$, and thus we use error back-propagation to update these two variables.

The pseudocode of CGNN with one submodule is given in Algorithm 2.

Algorithm 2: CGNN (one submodule)

Input: $\mathbf{X}$, $\mathbf{Y}$;
Output: Best learned C-graph $\mathbf{A}_{\mathcal{G}}^*$ and the values of parameters $\boldsymbol{\alpha}^*$, $\tilde{\boldsymbol{\alpha}}^*$ and $\boldsymbol{\theta}^*$;
Initialize parameters: $\boldsymbol{\alpha}$, $\tilde{\boldsymbol{\alpha}}$, $\boldsymbol{\theta}=\{\boldsymbol{\Theta}_1,\boldsymbol{\Theta}_2\}$;
Initialize hyperparameters;
while *Stopping condition is not met* **do**
  Substitute $\mathbf{X}$, $\mathbf{Y}$ and $\boldsymbol{\alpha}$ into (6)-(9) to compute the sample graph $\mathbf{A_S}$;
  Use $\mathbf{A_S}$ to obtain $\mathbf{D_S}$;
  $\tilde{\mathbf{X}}\leftarrow\mathbf{D_S}^{-\frac{1}{2}}(\mathbf{A_S}+\mathbf{I})\mathbf{D_S}^{-\frac{1}{2}}\mathbf{X}$;
  Use $\tilde{\mathbf{X}}$ and (11)-(12) to compute the trees of features and obtain the tree-based feature graph $\mathbf{A_F}$;
  Initialize $\tilde{\boldsymbol{\alpha}}$ using $\mathbf{A_F}$;
  Generate $\mathbf{A}_{\mathcal{G}}$ using $\mathbf{A_S}$ and $\mathbf{A_F}$;
  while *Inner objective decreases* **do**
    Substitute $\mathbf{X}$, $\mathbf{A}_{\mathcal{G}}$, $\tilde{\boldsymbol{\alpha}}$, $\boldsymbol{\theta}$, $\mathbf{Y}$ into (1)-(5) to compute $\mathcal{L}(\boldsymbol{\theta})$;
    Update $\boldsymbol{\alpha}$, $\tilde{\boldsymbol{\alpha}}$, $\boldsymbol{\theta}$ by using $\mathcal{L}(\boldsymbol{\theta})$;
  end
end
Denote the last iteration values of $\boldsymbol{\alpha}$, $\tilde{\boldsymbol{\alpha}}$, and $\boldsymbol{\theta}$ as $\boldsymbol{\alpha}^*$, $\tilde{\boldsymbol{\alpha}}^*$ and $\boldsymbol{\theta}^*$, respectively;
Substitute $\mathbf{X}$, $\mathbf{Y}$ and $\boldsymbol{\alpha}^*$ into (6)-(9) to compute the best sample graph $\mathbf{A_S^*}$;
Use $\mathbf{A_S^*}$ to obtain $\mathbf{D_S^*}$;
$\tilde{\mathbf{X}}^*\leftarrow\mathbf{D_S^*}^{-\frac{1}{2}}(\mathbf{A_S^*}+\mathbf{I})\mathbf{D_S^*}^{-\frac{1}{2}}\mathbf{X}$;
Use $\tilde{\mathbf{X}}^*$ and (11)-(12) to compute the trees of features and obtain the best tree-based feature graph $\mathbf{A_F^*}$;
Generate $\mathbf{A}_{\mathcal{G}}^*$ using $\mathbf{A_S^*}$ and $\mathbf{A_F^*}$;
return $\mathbf{A}_{\mathcal{G}}^*$, $\boldsymbol{\alpha}^*$, $\tilde{\boldsymbol{\alpha}}^*$ and $\boldsymbol{\theta}^*$.

IV. Experiments

In the experiments, we use 28 baseline methods and eight datasets to evaluate the performance of our model. We evaluate CGNN on node classification problems. Our experiments are conducted mainly to answer the following questions:

- Q_1 Does CGNN have better performance than other baseline methods in semi-supervised node classification problems when graphs are unknown?

TABLE II
DETAILS OF THE EXPERIMENTAL DATASETS. "NO" INDICATES THAT THE SAMPLE GRAPH IS UNKNOWN IN THIS DATASET.

Dataset	# Samples	# Features	# Sample Graph Edges	# Classes	Train / Val / Test	Type
Wine	178	13	No	3	10 / 20 / 158	Chemical
Cancer	569	30	No	2	10 / 20 / 539	Medical
Digits	1,797	64	No	10	50 / 100 / 1,647	Image
20news	9,607	236	No	10	100 / 200 / 9,307	News
FMA	7,994	140	No	8	160 / 320 / 7,514	Music
Citeseer	3,327	3,703	4,732	6	120 / 500 / 1,000	Citation
Cora	2,708	1,433	5,429	7	140 / 500 / 1,000	Citation
Pubmed	19,717	500	44,338	3	60 / 500 / 1,000	Citation
ogbn-arxiv	169,343	128	1,166,243	40	90,941 / 29,799 / 48,603	Citation

Q_2 Will CGNN perform worse if the C-graph is replaced by a sample graph or a feature graph?

Q_3 Is the proposed tree-based feature graph better than the feature graph based on pairwise similarity?

Q_4 Does CGNN have stable performance for the three proposed hyperparameters W_C, W_D, and W_I?

Q_5 CGNN can learn multiple C-graphs using multiple submodules, but do multiple C-graphs really improve performance?

Q_6 What is the relationship between the number of submodules and the time and space consumption of CGNN?

Q_7 Does jointly learning graph structures and parameters improve the classification performance of our proposed CGNN when the structure of sample graph is known?

Q_8 Does CGNN have the scalability to be applied in large datasets?

A. Details of Experiments

Datasets Settings. In the experiments, we use eight benchmark datasets to evaluate the performance of CGNN. The datasets include three frequently utilized citation network datasets in GNNs such as Citeseer [58], Cora [59], and Pubmed [60], a music genre classification dataset FMA [61], and four classic datasets from scikit-learn [62]: Wine, Cancer, Digits and 20news. These datasets are commonly used to evaluate graph learning methods [19]. In addition, ogbn-arxiv [63] is used in our experiments to evaluate the scalability and performance of CGNN in large graph applications. We use the same dataset split of previous work [19], [64]. Details of the above datasets are given in Table II.

Experimental Settings. For the sake of fair comparisons, the hyperparameters of baselines are selected according to the original implementations of the authors. Specifically, we tune hyperparameters using validation datasets and select their values through the validation loss and accuracy. The values of main hyperparameters are given in Table III. In our method, the activation function σ is set as the ReLU function, the function ℓ is set as the cross-entropy error function, and the optimizer is set as AdaMax [65].

Experimental Environment. We use Python 3.7, TensorFlow 2.0, and XGBoost [57] to implement our method[1], and we run all the codes on a Hewlett-Packard server with 4 $\times$ NVIDIA Quadro RTX 6000 24GB GPUs, 4 $\times$ Intel Xeon Gold 5210 CPUs, 16 $\times$ 32GB DDR4 RAM, and 5 $\times$ 2TB hard disks.

[1]Codes is available at https://github.com/Peter7777777/CGNN.

B. The Performance of CGNN without Knowing Graphs in Advance (Q_1)

CGNN tries to solve the semi-supervised node classification problem when graph structures are completely unknown. In this case, we compare CGNN with 14 baseline methods including:

a Four classical machine learning algorithms: logistic regression (LogReg), feed-forward neural networks (FFNN), and support vector machines (Linear SVM and RBF SVM).

b Two ensemble learning methods: random forests (RF) [66], and eXtreme Gradient Boosting (XGB) [57] which is the basis of constructing ensemble trees in our model.

c Three popular semi-supervised learning methods: label propagation (LP) [67], manifold regularization (ManiReg) [68], and semi-supervised embedding (SemiEmb) [32]. ManiReg and SemiEmb are provided with a kNN sample graph for graph Laplacian regularization.

d Seven GNN methods: Sparse-GCN, Dense-GCN, RBF-GCN, kNN-GCN, kNN-GAT, kNN-AdaGCN and kNN-LDS. Sparse-GCN creates a sparse Erdös-Rényi random sample graph, Dense-GCN creates a dense sample graph with equal edge probabilities, and RBF-GCN creates a dense RBF kernel sample graph. kNN-GCN, kNN-GAT, kNN-AdaGCN and kNN-LDS create a kNN sample graph.

To ensure the fair comparison with baseline methods, we adopt the same dataset preprocessing and splitting way as that in kNN-LDS [19]. Table IV shows the comparative experiment results of CGNN and the baseline methods. In terms of classification accuracy, we can observe that CGNN achieves the highest node classification accuracies than other methods on all datasets except Digits. CGNN underperforms kNN-GAT on the Digits dataset because every neighbor in the C-graph has the same importance, while kNN-GAT considers the importance of each neighbor utilizing an attention mechanism to assign different weights to nodes, which helps kNN-GAT learn better embeddings than CGNN on Digits. Comparing CGNN with kNN-LDS, we can find that CGNN can achieve

TABLE III
VALUES OF MAIN HYPERPARAMETERS.

Hyperparameters	Wine	Cancer	Digits	Citeseer	Cora	20news	FMA	Pubmed	ogbn-arxiv
Learning rate	0.01	0.01	0.1	0.01	0.01	0.01	0.01	0.1	0.01
Number of epochs	800	700	800	800	800	800	1000	700	800
η (in Section III-C1)	50	85	30	12	12	12	12	12	10
w_C (in Section III-C3)	1.2	1	1.2	1.3	1.3	1.3	1.2	1.5	1.5
w_D (in Section III-C3)	1.1	0.8	0.02	1.1	1.1	1.1	0.2	0.2	0.02
w_I (in Section III-C3)	1	0.5	0.01	0.9	0.9	0.9	0.1	0.1	0.01
Number of trees for each class	100								
Max depth of each tree	5								
Default number of submodules	3								
Features of the first hidden layer	64								
L_2 Regularizer parameter	5.00E-05								
Dropout rate	0.5								

TABLE IV
THE PERCENTAGE OF THE AVERAGE ACCURACY ($\pm$ STANDARD DEVIATION) WITHOUT GRAPH STRUCTURE AS INPUT, AND WE FUSE THREE SUBMODULES IN CGNN FOR REPORT ITS RESULTS. EACH RESULT OF CGNN IS REPORTED BASED ON 30 MONTE CARLO EXPERIMENTS. THE TOP THREE RESULTS ARE IN BOLD, AND CGNN ACHIEVES THE BEST RESULTS ON ALL THE DATASETS.

Method	Wine	Cancer	Digits	Citeseer	Cora	20news	FMA
LogReg	92.1 (1.3)	93.3 (0.5)	85.5 (1.5)	62.2 (0.0)	60.8 (0.0)	42.7 (1.7)	37.3 (0.7)
Linear SVM	93.9 (1.6)	90.6 (4.5)	87.1 (1.8)	58.3 (0.0)	58.9 (0.0)	40.3 (1.4)	35.7 (1.5)
RBF SVM	94.1 (2.9)	91.7 (3.1)	86.9 (3.2)	60.2 (0.0)	59.7 (0.0)	41.0 (1.1)	38.3 (1.0)
FFNN	89.7 (1.9)	92.9 (1.2)	36.3 (10.3)	56.7 (1.7)	56.1 (1.6)	38.6 (1.4)	33.2 (1.3)
RF	93.7 (1.6)	92.1 (1.7)	83.1 (2.6)	60.7 (0.7)	58.7 (0.4)	40.0 (1.1)	37.9 (0.6)
XGB	85.1 (0.0)	85.7 (0.0)	64.8 (0.0)	56.6 (0.0)	57.6 (0.0)	38.4 (0.0)	25.9 (0.0)
LP	89.8 (3.7)	76.6 (0.5)	91.9 (3.1)	23.2 (6.7)	37.8 (0.2)	35.3 (0.9)	14.1 (2.1)
ManiReg	90.5 (0.1)	81.8 (0.1)	83.9 (0.1)	67.7 (1.6)	62.3 (0.9)	46.6 (1.5)	34.2 (1.1)
SemiEmb	91.9 (0.1)	89.7 (0.1)	90.9 (0.1)	68.1 (0.1)	63.1 (0.1)	**46.9** (0.1)	34.1 (1.9)
Sparse-GCN	63.5 (6.6)	72.5 (2.9)	13.4 (1.5)	33.1 (0.9)	30.6 (2.1)	24.7 (1.2)	23.4 (1.4)
Dense-GCN	90.6 (2.8)	90.5 (2.7)	35.6 (21.8)	58.4 (1.1)	59.1 (0.6)	40.1 (1.5)	34.5 (0.9)
RBF-GCN	90.6 (2.3)	92.6 (2.2)	70.8 (5.5)	58.1 (1.2)	57.1 (1.9)	39.3 (1.4)	33.7 (1.4)
kNN-GCN	93.2 (3.1)	93.8 (1.4)	91.3 (0.5)	68.3 (1.3)	66.5 (0.4)	41.3 (0.6)	37.8 (0.9)
kNN-GAT	97.3 (0.1)	93.1 (0.1)	**93.4** (0.1)	**69.3** (1.1)	65.7 (0.4)	41.3 (0.7)	38.8 (0.2)
kNN-AdaGCN	**97.8** (0.5)	**94.1** (2.7)	91.9 (3.6)	59.0 (0.6)	58.1 (2.0)	**48.5** (1.4)	38.6 (0.5)
kNN-LDS	97.3 (0.4)	**94.4** (1.9)	92.5 (0.7)	**71.5** (1.1)	**71.5** (0.8)	46.4 (1.6)	**39.7** (1.4)
SCRL	93.9 (2.1)	92.2 (0.3)	89.5 (1.3)	66.2 (2.7)	65.7 (2.2)	40.3 (1.9)	**38.9** (1.7)
SPGRL	**97.7** (0.3)	94.0 (0.2)	**92.6** (0.4)	69.0 (1.3)	**68.9** (0.9)	42.4 (0.6)	37.4 (0.4)
CGNN	**98.0** (0.1)	**94.5** (0.1)	**92.7** (0.4)	**73.6** (0.3)	**73.4** (0.4)	**50.2** (0.6)	**40.5** (0.2)

2% higher accuracies than kNN-LDS on Citeseer and Cora, and about 4% higher than kNN-LDS on 20news. CGNN also outperforms kNN-AdaGCN by a significant margin on Citeseer and Cora. CGNN outperforms baseline methods since it captures and utilizes the relationships between features by learning C-graphs. The comparison between CGNN and the ensemble learning algorithms RF and XGB also shows that although CGNN uses the same scheme as XGB to implement the ensemble tree models, the performance of CGNN is much higher than that of XGB. This result further highlights the importance of learning C-graphs. As for the standard deviation, the experimental results of CGNN on all the datasets are also significantly lower than those of other GNN baseline methods. We also give a visualization result of the learned C-graph in Fig. 4 and visualization results of $\boldsymbol{\alpha}$ and $\tilde{\boldsymbol{\alpha}}$ in Fig. 5.

We conduct some experiments to show the robustness of CGNN and the results are shown in Table V. We add random noises on the original dataset. The noises are randomly sampled from a uniform distribution over the interval $[0, 0.2]$, and the proportion of features that are added noises is given in the first column of Table V. From Table V, we can find that when the noise ratio reaches 10%, the performance of our algorithm is still better than that of kNN-LDS. When the ratio reaches 20%, the performance of our algorithm is still close to the performance of kNN-LDS on clean datasets. When the noise ratio is 50%, the performance of our algorithm decreases on Cora, Citeseer and 20news, but in general, are still comparable to the performance of other baselines on clean datasets. The average node classification results on several datasets (e.g., Wine, Cancer, and Digits) do not change because these datasets are relatively simple, within a certain degree of noise ratio. After reducing the impact of noises through the proposed trainable vector $\boldsymbol{\alpha}$, the sample graph learned after adding noises is high-quality. Moreover, the features used to generate trees are first filtered by $\boldsymbol{\alpha}$, which helps find valuable features in noisy data. Furthermore, the trees are constructed by minimizing the classification loss, which ensures the features selected for data splitting are essential for the target classification task and high-quality feature trees. Consequently, our model gets high-quality C-graphs, equivalent to the C-graphs we are learning from these datasets when we do not add noises.

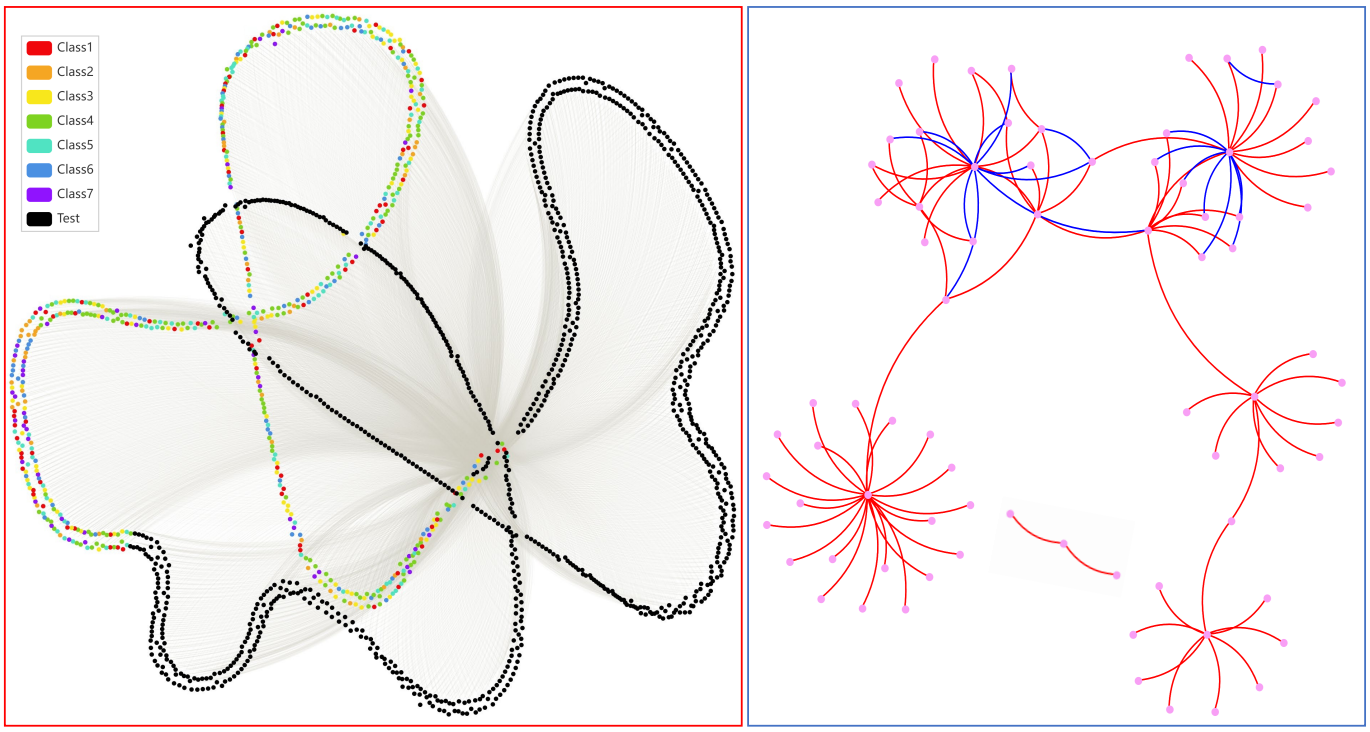


Fig. 4. The C-graph learned on the Cora dataset when our method has one submodule. The graph in the red box is the sample graph. In the sample graph, the classes of the black nodes are unknown, the other nodes use different colors to represent their classes, and each grey line is an edge between two samples. The graph in the blue box is a tree-based feature graph. For the convenience of observation, we do not show the feature nodes without direct or indirect relations to other feature nodes. The pink nodes are features in the tree-based feature graph, and the red and blue edges are the direct and indirect relations between features, respectively.

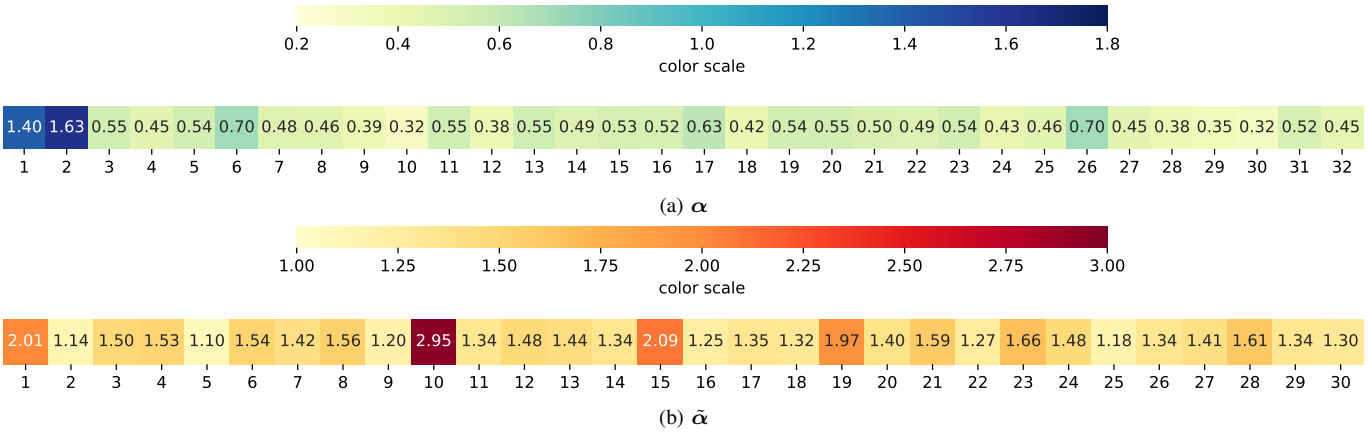


Fig. 5. Heatmap visualization results of $\boldsymbol{\alpha}$ and $\tilde{\boldsymbol{\alpha}}$ on the Cancer dataset when our method uses one submodule. The scale value in the horizontal direction represents the feature number. Recall that $\boldsymbol{\alpha} \in \mathbb{R}^{F+C}$ and $\tilde{\boldsymbol{\alpha}} \in \mathbb{R}^{F}$, so $\boldsymbol{\alpha}$ is longer than $\tilde{\boldsymbol{\alpha}}$.

TABLE V
AVERAGE NODE CLASSIFICATION ACCURACY (%) OF CGNN IN THE ROBUSTNESS EXPERIMENTS. THE FIRST COLUMN INDICATES THE PROPORTION OF FEATURES THAT ARE ADDED NOISES.

Noise	Wine	Cancer	Digits	Citeseer	Cora	20news	FMA
1%	**98.0**	**94.5**	**92.6**	**73.2**	**73.1**	**49.3**	**40.5**
2%	**98.0**	**94.5**	**92.6**	72.9	**73.1**	49.2	**40.5**
5%	**98.0**	**94.5**	**92.6**	72.5	72.9	48.2	**40.5**
10%	**98.0**	**94.5**	**92.6**	72.5	72.5	46.5	40.4
20%	**98.0**	94.2	**92.6**	71.7	71.1	44.3	40.4
50%	**98.0**	94.1	92.4	70.6	70.1	38.7	40.3

C. Ablation Study (Q_2, Q_3)

Our proposed CGNN has two key components to learning C-graphs: learning sample graphs and learning tree-based feature graphs. In this section, we validate their impacts on the performance via ablation experiments. We detail the variants of CGNN as follows:

- SGNN: SGNN (sample graph neural network) only learns the sample graphs and uses them to replace the C-graphs in CGNN. In other words, SGNN is equivalent to CGNN without learning the tree-based feature graphs.
- FGNN: FGNN (tree-based feature graph neural network)

is the method that only learns the feature graphs. Notably, FGNN uses the tree-based feature graphs generated in the final iteration and the sample graphs generated in the first iteration to conduct node classification. FGNN requires sample graphs since learning feature graphs and solving node classification depend on sample relations in our approach. Using the sample graphs generated at the initial iteration facilitates us to compare the difference between learning C-graphs and only learning tree-based feature graphs without learning the sample graphs.

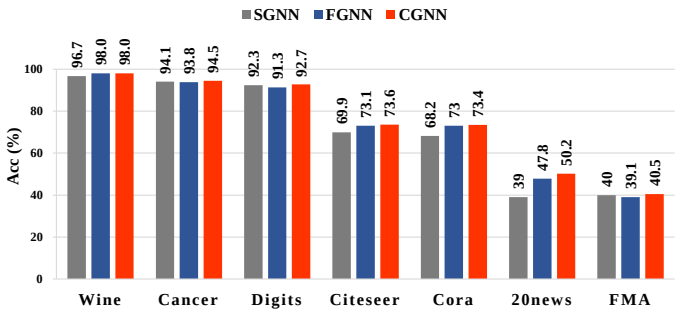


Fig. 6. Comparison of the percentage of the average node classification accuracy (Acc) of CGNN and its two variants, SGNN and FGNN. SGNN denotes only learning the sample graphs in CGNN. FGNN denotes only learning the feature graphs in CGNN.

The results of the ablation experiments are given in Fig. 6. From the node classification accuracies in Fig. 6, we see that CGNN outperforms SGNN and FGNN. In general, the results imply that both learning sample graphs and tree-based feature graphs are conducive to the model performance. It is worth noting that although the results of SGNN or FGNN are close to those of CGNN sometimes on some datasets, the performance of CGNN is much better than its two variants on 20news. The main reason is that only learning sample relations or feature relations are unable to get satisfactory embeddings for the downstream neural network to classify nodes nicely on the dataset. Therefore, both the sample graph and the tree-based feature graph are important in graph node embedding.

We also compare CGNN with its two another variants:

- CGNN-S: CGNN-S (composite graph neural network with a simple fully connected layer) only uses a simple fully connected layer to learn the embeddings for classification.
- CGNN-M: CGNN-M (composite graph neural network with multiple layers) uses multiple hidden layers to learn the embeddings for classification.

The main difference between CGNN, CGNN-S, and CGNN-M is the number of network layers. The neural network in CGNN has two hidden layers. Fig. 7 gives the comparison results between the three methods, and we set the hidden layer number as five to report the results of CGNN-M. From Fig. 7, we observe that CGNN has the best performance, which means two hidden layers are most appropriate for the current CGNN. However, since the results of CGNN-M on several datasets (e.g., Citeseer, Cora, and 20news) are inferior to that of CGNN, this indicates that CGNN also exists the limitation of over-smoothing, i.e., with the number of layers increasing, the representations of nodes in CGNN become indistinguishable. Thus, we think it's significant future work to design an improved CGNN that not only jointly learns the C-graph and neural network parameters but also deals with the problem of over-smoothing.

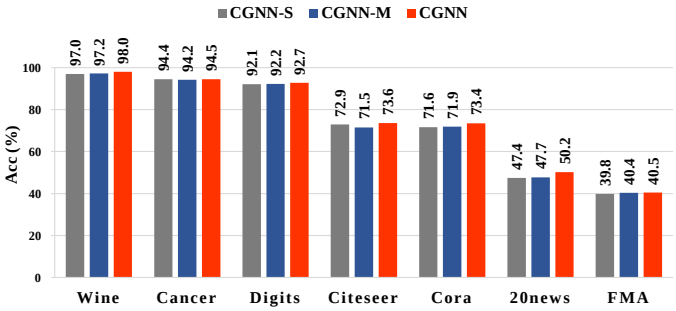


Fig. 7. Comparison of the percentage of the average node classification accuracy (Acc) of CGNN and its another two variants, CGNN-S and CGNN-M. Difference from CGNN, which uses two hidden layers to learn the embeddings for classification, CGNN-S uses a simple fully connected layer, and CGNN-M uses multiple hidden layers to learn the embeddings for classification. The hidden layers number of CGNN-M is set as five to report the results.

To further compare the difference between our proposed tree-based feature graph and pairwise similarity-based feature graph, we propose T-EmbNN (tree-based feature graph embedding neural network). T-EmbNN is an extremely simplified variant method of CGNN, which only uses tree-based feature graphs to conduct node classification. Specifically, we use $\tilde{\mathbf{X}} = \mathbf{X}$ to replace $\tilde{\mathbf{X}} = \mathbf{D}_{\mathbf{S}}^{-\frac{1}{2}}(\mathbf{A}_{\mathbf{S}}+\mathbf{I})\mathbf{D}_{\mathbf{S}}^{-\frac{1}{2}}\mathbf{X}$ in (11)-(12) and Algorithm 1, and we use $\mathbf{X}^{(2)} = \sigma\left(\mathbf{X}^{(1)}\mathbf{\Theta}_2\right)$ to replace $\mathbf{X}^{(2)} = \sigma\left(\mathbf{D}_{\mathbf{S}}^{-\frac{1}{2}}(\mathbf{A}_{\mathbf{S}}+\mathbf{I})\mathbf{D}_{\mathbf{S}}^{-\frac{1}{2}}\mathbf{X}^{(1)}\mathbf{\Theta}_2\right)$ in (2). In order to control variables, the number of submodules of T-EmbNN is set to one. Then we compare T-EmbNN with two methods that use pairwise similarity of features to generate a feature graph. These two methods are denoted as E-EmbNN and C-EmbNN, and use the popular Euclidean distance and cosine similarity to measure the similarity between paired features, respectively. Both E-EmbNN and C-EmbNN use a kNN graph, $k \in \{2, 3, \ldots, 20\}$, to construct the sparsified feature graph. In addition to E-EmbNN and C-EmbNN, we also compare T-EmbNN with a neural network approach that does not use any graph structures, denoted as NN. NN takes the simple form: $\mathrm{Softmax}\left(\sigma\left(\sigma\left(\mathbf{X}\mathbf{\Theta}_1\right)\mathbf{\Theta}_2\right)\right)$. In other words, the model of NN only has an input layer, two hidden fully connected layers, and an output layer. The results of the comparative experiments are given in Fig. 8. From Fig. 8, we can observe that T-EmbNN achieves the best performance in most cases. It is worth noting that T-EmbNN is significantly superior to other methods on Citeseer, Cora, and 20news. It can also be observed that E-EmbNN is not always better than C-EmbNN, and vice versa. This implies that it is difficult to find a pairwise feature similarity metric that is appropriate to all these datasets. Moreover, on some datasets, both the performance of E-EmbNN and C-EmbNN are even inferior to NN. In this experiment, T-EmbNN achieves better performance on almost all the datasets, which demonstrates the importance of learning feature importance and proper feature combination.

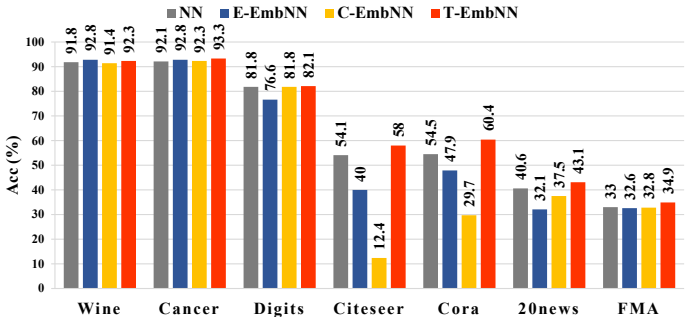


Fig. 8. Comparative experiments on the average node classification accuracy (Acc) of learning tree-based feature graphs. T-EmbNN only learns a tree-based feature graph, which is an extremely simplified variant method of CGNN. Both E-EmbNN and C-EmbNN learn a kNN feature graph, and they use the Euclidean distance and cosine similarity to measure the pairwise feature similarity, respectively. NN denotes a neural network approach that does not use any graph structures.

D. Sensitivity Analysis of Hyperparameters (Q_4)

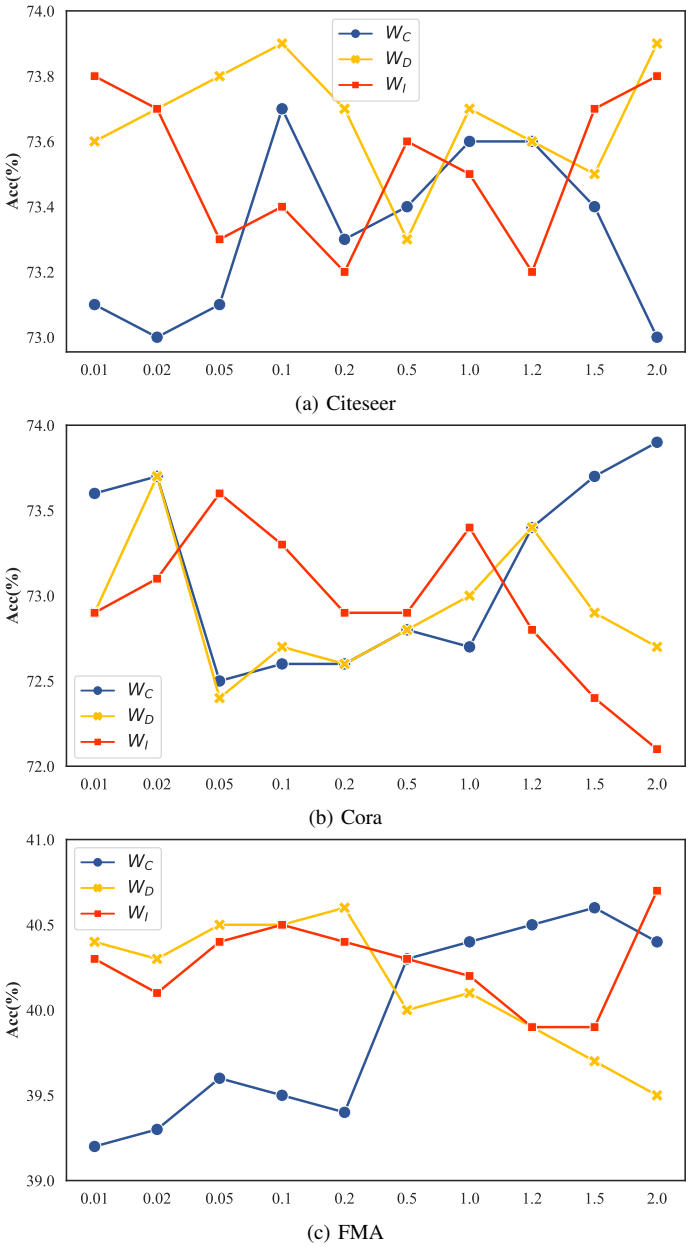
In this section, we analyze the three proposed hyperparameters, W_C, W_D, and W_I. First, we test them by setting each of their values via a grid search in $\{0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0, 1.2, 1.5, 2.0\}$, and fix others values as given in Table III. We report the average classification accuracy of ten runs, and the results are shown in Fig. 9. We find that our model mostly performs better than the baselines in Table IV. Even if we only look at the worst results on each dataset, the performance of CGNN still stays comparable. Furthermore, in order to analyze the sensitivity of the three hyperparameters more comprehensively, we conduct an experiment by setting all of their values in $\{0.01, 0.1, 0.5, 1.0, 2.0\}$ through grid search. Since there are a total of 125 combinations and three variables are involved, we choose parallel coordinates to show the experimental results. We report the average classification accuracy of five runs on Cancer, and the results are shown in Fig. 10. We observe that our model has stable performance as the results mostly reach the fine-tuned accuracy in Table IV.

E. Space and Time Consumption (Q_5, Q_6)

We consider that CGNN can improve the classification accuracy by fusing different C-graphs learned in multiple submodules. We perform experiments to study the impact of the number of submodules on the classification accuracy of CGNN. Table VI shows the classification accuracy of CGNN with varying numbers of submodules when graphs are unknown. In Table VI, the number in the first column of

TABLE VI
AVERAGE NODE CLASSIFICATION ACCURACY (%) OF CGNN WITH VARYING NUMBERS OF SUBMODULES WHEN NO GRAPH STRUCTURE IS PROVIDED AS INPUT. HERE R DENOTES THE NUMBER OF SUBMODULES.

R	Wine	Cancer	Digits	Citeseer	Cora	20news	FMA
1	**98.0**	94.4	92.5	73.0	73.1	48.5	40.4
2	**98.0**	**94.5**	92.6	73.1	73.2	49.9	40.4
3	**98.0**	**94.5**	92.7	73.6	73.4	50.2	**40.5**
4	**98.0**	**94.5**	**92.8**	73.1	73.7	50.3	40.2
5	**98.0**	94.4	92.6	73.6	73.8	50.7	40.4



(a) Citeseer

(b) Cora

(c) FMA

Fig. 9. The influence of the hyperparameters, W_C, W_D and W_I, on Citeseer, Cora and FMA.

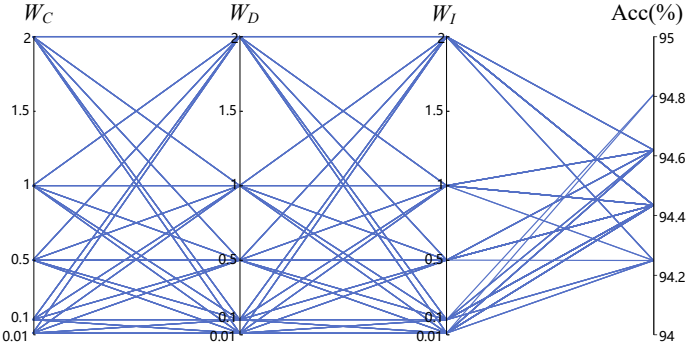


Fig. 10. The influence of the hyperparameters, W_C, W_D and W_I, on Cancer.

TABLE VII
BRIEF DESCRIPTIONS OF THE BASELINE METHODS FOR COMPARING WITH CGNN WHEN THE SAMPLE GRAPH IS KNOWN A PRIORI.

Algorithm	Description
GIN [69]	A GNN method under the neighborhood aggregation framework.
GCN [12]	A method aggregates neighbor information with spectral convolutions on graphs.
GAT [13]	A method aggregates neighbor information by attention mechanism.
Graphite [45]	A latent variable generative model based on variational autoencoding.
BGCN-T [39]	A method improves GCN with bilinear interactions between neighbor nodes.
BGAT-T [39]	A method improves GAT with bilinear interactions between neighbor nodes.
SGC [70]	A simple and efficient GCN-based method.
Shoestring [71]	A graph-based method with severely limited labeled data.
APPNP [72]	A method utilizes the relationship between GCN and personalized PageRank.
DAGNN [15]	A deep adaptive GNN method can capture sample relations from large and adaptive receptive fields.
SelfSAGCN [73]	A GCN-based simple and effective self-supervised semantic alignment method.
kNN-LDS [19]	A method jointly learns the graph structure of samples and parameters of a neural network.
GCNII [14]	A simple and deep GCN-based method via initial residual and identity mapping.
(A+kNN)-GCN [64]	A GCN-based method uses the given sample graph and a constructed kNN feature graph.
SimP-GCN [64]	A node similarity preserving GCN-based method.
SCRL [74]	A self-supervised consensus graph representation learning method.
SPGRL [75]	A structure-preserving graph representation learning method.
Ortho-GCN [76]	A method improves GCN with orthogonal graph convolutions.
Ortho-GCNII [76]	A method improves GCNII with orthogonal graph convolutions.

each row indicates the number of submodules in CGNN. From Table VI, we see that properly increasing the number of submodules, especially from 1 to 3, can improve the performance of our method. Thus, after considering the tradeoff between computation complexity and performance, we choose $R = 3$ as the default number of submodules in our method. Fig. 11 and Fig. 12 give the results of average running time (second) and memory (MB) of CGNN under different submodules, respectively. From Fig. 11 and Fig. 12, we observe that the time and space consumption of CGNN increase linearly with the number of submodules in general.

Complexity. The time complexity of learning sample graph and tree-based feature graph are $O(N^2)$ and $O(NFTD)$, respectively. N and F are the number of samples and features of the input dataset, respectively; T and D denote the number of trees and the max depth of each tree, respectively. Thus, the time complexity of our method is $O(N^2 + NFTD + |\mathcal{E}_S||\mathcal{E}_F|U)$, where U is the number of hidden units in the network; $\mathcal{E}_S$ and $\mathcal{E}_F$ denote the edge number of the learned sample graph and tree-based feature graph, respectively. The time complexity of the baseline method kNN-GCN is $O(N^2 + |\mathcal{E}_S|U)$, and other GCN-based baselines have similar time complexity with kNN-GCN. Our method has higher time complexity than these methods since it learns and captures feature relations while others do not.

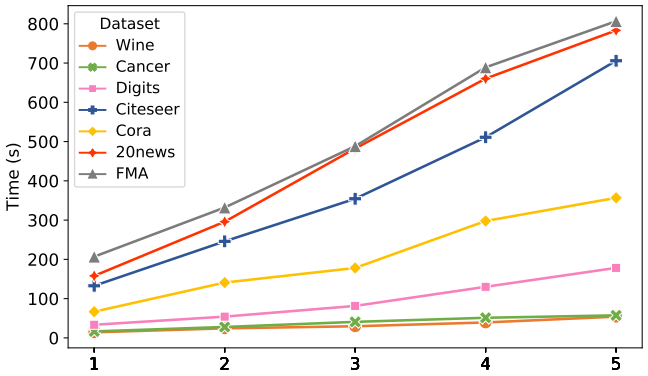


Fig. 11. Average training time (second) of CGNN when using different submodules. The horizontal coordinate indicates the number of submodules.

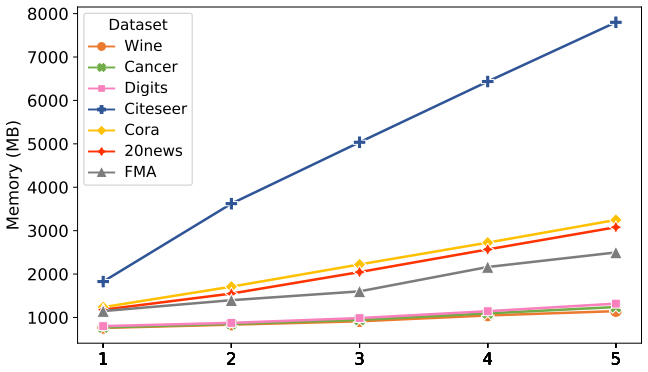


Fig. 12. Average memory (MB) of CGNN when using different submodules. The horizontal coordinate indicates the number of submodules.

F. The Performance of CGNN When the Sample Graph Is Known (Q_7)

Our method is proposed mainly for the scenarios where graph structures are completely inaccessible and noisy. In this experiment, we will show our method is better than baselines when graph structures are known a priori. To this end, we compare our method with many GNNs on datasets with known graphs. Specifically, we learn the tree-based feature graph using the known sample graph, and then obtain the C-graph. We select some recently proposed and state-of-the-art algorithms for comparative experiments. Brief descriptions of the baseline methods are given in Table VII.

Table VIII shows the results of CGNN and the baseline methods. We provide a comparison on each dataset by following the same dataset splitting setup as that in previous

works [39], [64], and some reported results of the baseline methods also come from these publications. From Table VIII, we observe CGNN achieves the best accuracy on all the three datasets. The main reason why CGNN performs better than other baselines is that CGNN learns the tree-based feature graphs and thus captures and utilizes the beneficial feature relationships.

TABLE VIII
THE PERCENTAGE OF THE AVERAGE ACCURACY (± STANDARD DEVIATION) WHEN GRAPH IS PROVIDED AS INPUT. EACH RESULT OF CGNN IS REPORTED BASED ON 30 MONTE CARLO EXPERIMENTS. THE TOP THREE RESULTS ARE IN BOLD, AND CGNN ACHIEVES THE BEST RESULTS ON ALL THE DATASETS.

Method	Cora	Citeseer	Pubmed
GIN	79.7 (0.8)	69.4 (0.6)	78.5 (0.2)
GCN	81.2 (0.4)	71.1 (0.7)	78.5 (1.0)
GAT	83.1 (0.7)	72.5 (0.7)	79.0 (0.3)
Graphite	82.1 (0.1)	71.0 (0.1)	79.3 (0.1)
BGCN-T	82.0 (0.1)	71.9 (0.1)	79.4 (0.1)
BGAT-T	84.2 (0.4)	74.0 (0.3)	79.8 (0.3)
SGC	81.7 (0.6)	71.3 (1.1)	78.9 (1.3)
Shoestring	81.9 (2.1)	69.5 (2.4)	79.7 (4.5)
APPNP	83.3 (0.4)	71.8 (0.9)	80.1 (1.3)
DAGNN	84.4 (0.4)	73.3 (0.7)	80.5 (0.9)
SelfSAGCN	83.8 (0.5)	73.5 (1.2)	80.7 (1.5)
kNN-LDS	84.1 (0.4)	**75.0** (0.4)	80.0 (0.5)
GCNII	**85.5** (0.5)	73.4 (0.6)	80.2 (0.4)
(A+kNN)-GCN	79.1 (0.7)	71.1 (0.7)	80.8 (0.9)
SimP-GCN	82.8 (0.5)	72.6 (0.7)	**81.1** (0.6)
SCRL	72.5 (1.0)	73.6 (1.4)	79.6 (1.1)
SPGRL	81.3 (0.4)	**75.1** (1.1)	77.6 (1.0)
Ortho-GCN	82.8 (1.0)	72.3 (1.2)	80.7 (0.5)
Ortho-GCNII	**85.6** (0.7)	74.1 (1.1)	**81.2** (0.6)
CGNN	**86.3** (0.5)	**75.7** (0.7)	**81.6** (0.7)

G. Scalability of CGNN in Large Dataset Applications (Q_8)

We evaluate the performance of our method on eight datasets with less than 20,000 nodes in both scenarios where the graph structure is unknown or known. In this section, we conduct experiments on ogbn-arxiv with 169,343 nodes to further evaluate the scalability and performance of CGNN on large datasets. We compare CGNN with baseline methods on ogbn-arxiv under the same dataset splitting setup as that in previous works [63], [76]. In Table IX, we give the results of CGNN and baselines, and the number of hidden layers is set to two for a fair comparison. From Table IX, we know that CGNN is applicable to large datasets in scenarios where graph structure is unknown or known a priori, and CGNN achieves the best performance in both scenarios. The results demonstrate that learning the C-graph is vital for graph learning, especially when graph structure is unknown a priori.

TABLE IX
THE PERCENTAGE OF THE AVERAGE ACCURACY (± STANDARD DEVIATION) IN THE LARGE GRAPH APPLICATION. EACH RESULT OF CGNN IS REPORTED BASED ON 30 MONTE CARLO EXPERIMENTS.

Dataset	Scenario	Method	Result
ogbn-arxiv	graph unknown a priori	kNN-GCN	53.5 (0.3)
		kNN-GAT	54.2 (0.2)
		CGNN	**54.9** (0.3)
	graph known a priori	GCN	71.3 (0.3)
		GAT	71.3 (0.2)
		GCNII	71.2 (0.2)
		Ortho-GCN	71.3 (0.3)
		Ortho-GCNII	71.4 (0.2)
		CGNN	**72.2** (0.3)

V. CONCLUSION

In this work, we have presented a new Graph Neural Network method named Composite Graph Neural Network (CGNN) to solve semi-supervised classification problems. Unlike other GNN methods, our composite graphs not only characterize sample similarities but also imply feature importance and combination preference. Furthermore, CGNN can learn multiple composite graphs (C-graphs) from different aspects. Each C-graph is obtained by simultaneously optimizing the graph structure of samples and the tree structure of features, which ensures robustness. Experiments on multiple datasets show our model achieves the best performance on almost all the datasets and is robust to feature noises.

In terms of limitations, just like many GNN methods face the problem of over-smoothing, there is also an over-smoothing phenomenon in CGNN. Although the over-smoothing problem has aroused many concerns recently, the existing methods are not good enough to help CGNN. The key reason is that existing approaches do not have optimization schemes for the C-graph. Thus, employing existing techniques in our framework can not ensure a high-quality C-graph is obtained by jointly learning the C-graph and neural network parameters. How to improve CGNN to avoid over-smoothing deserves further study. Another limitation is that our method has more time complexity than most baselines since it learns and captures feature relations while others do not. Two solutions may be investigated in the future to improve the computation efficiency: (i) our method learns the sample graph of the C-graph using all the samples, a node cluster algorithm proposed in [77] may help our model use a part of samples to infer the sample graph of all samples quickly; and (ii) since our model includes many submodules to learn C-graphs, an approach that can split the training process into optimizing many subtasks [78] may be involved in our model to speed up the training process.

REFERENCES

[1] G. Li, M. Müller, B. Ghanem, and V. Koltun, "Training graph neural networks with 1000 layers," in *Proceedings of the International Conference on Machine Learning*, 2021, pp. 6437–6449.

[2] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[3] Y. Yang, Z. Ren, H. Li, C. Zhou, X. Wang, and G. Hua, "Learning dynamics via graph neural networks for human pose estimation and tracking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021, pp. 8074–8084.

[4] D. Gao, K. Li, R. Wang, S. Shan, and X. Chen, "Multi-modal graph neural network for joint reasoning on vision and scene text," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 743–12 753.

[5] Y. Zhang, X. Yu, Z. Cui, S. Wu, Z. Wen, and L. Wang, "Every document owns its structure: Inductive text classification via graph neural networks," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 334–339.

[6] L. Chen, Y. Zhao, B. Lyu, L. Jin, Z. Chen, S. Zhu, and K. Yu, "Neural graph matching networks for chinese short text matching," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 6152–6158.

[7] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *Proceedings of the International Conference on Learning Representations*, 2019, pp. 1–25.

[8] L. C. Lamb, A. S. d'Avila Garcez, M. Gori, M. O. R. Prates, P. H. C. Avelar, and M. Y. Vardi, "Graph neural networks meet neural-symbolic computing: A survey and perspective," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2020, pp. 4877–4884.

[9] G. Panagopoulos, G. Nikolentzos, and M. Vazirgiannis, "Transfer graph neural networks for pandemic forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021, pp. 4838–4845.

[10] V. L. Gatta, V. Moscato, M. Postiglione, and G. Sperlì, "An epidemiological neural network exploiting dynamic graph structured data applied to the COVID-19 outbreak," *IEEE Transactions on Big Data*, vol. 7, no. 1, pp. 45–55, 2021.

[11] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 3558–3565.

[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the International Conference on Learning Representations*, 2017, pp. 1–14.

[13] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proceedings of the International Conference on Learning Representations*, 2018, pp. 1–12.

[14] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proceedings of the International Conference on Machine Learning*, vol. 119, 2020, pp. 1725–1735.

[15] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2020, pp. 338–348.

[16] X. Wei, L. Xu, B. Cao, and P. S. Yu, "Cross view link prediction by learning noise-resilient representation consensus," in *Proceedings of the International Conference of World Wide Web*, 2017, pp. 1611–1619.

[17] M. Zhang and Y. Chen, "Weisfeiler-lehman neural machine for link prediction," in *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2017, pp. 575–583.

[18] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang, "Graph structure learning for robust graph neural networks," in *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2020, pp. 66–74.

[19] L. Franceschi, M. Niepert, M. Pontil, and X. He, "Learning discrete structures for graph neural networks," in *Proceedings of the International Conference on Machine Learning*, vol. 97, 2019, pp. 1972–1982.

[20] T. Wanyan, A. Vaid, J. K. D. Freitas, S. Somani, R. Miotto, G. N. Nadkarni, A. Azad, Y. Ding, and B. S. Glicksberg, "Relational learning improves prediction of mortality in COVID-19 in the intensive care unit," *IEEE Transactions on Big Data*, vol. 7, no. 1, pp. 38–44, 2021.

[21] S. Kim and E. P. Xing, "Tree-guided group lasso for multi-task regression with structured sparsity," in *Proceedings of the International Conference on Machine Learning*, 2010, pp. 543–550.

[22] J. Wang and J. Ye, "Multi-layer feature reduction for tree structured group lasso via hierarchical projection," in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2015, pp. 1279–1287.

[23] T. Sandler, J. Blitzer, P. P. Talukdar, and L. H. Ungar, "Regularized learning with networks of features," in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2008, pp. 1401–1408.

[24] S. Yang, L. Yuan, Y. Lai, X. Shen, P. Wonka, and J. Ye, "Feature grouping and selection over an undirected graph," in *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2012, pp. 922–930.

[25] S. Kim and E. P. Xing, "Statistical estimation of correlated genome associations to a quantitative trait network," *PLoS Genetics*, vol. 5, no. 8, p. e1000587, 2009.

[26] K. Jia, T. Chan, and Y. Ma, "Robust and practical face recognition via structured sparsity," in *Proceedings of the European Conference on Computer Vision*, vol. 7575, 2012, pp. 331–344.

[27] Z. Li, Z. Cui, S. Wu, X. Zhang, and L. Wang, "Fi-GNN: Modeling feature interactions via graph neural networks for CTR prediction," in *Proceedings of the ACM International Conference on Information and Knowledge Management*, 2019, pp. 539–548.

[28] W. Chen, F. Feng, Q. Wang, X. He, C. Song, G. Ling, and Y. Zhang, "CatGCN: Graph convolutional networks with categorical node features," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 3500–3511, 2023.

[29] S. Yun, Z. Guo-ying, and Y. Yong, "A road detection algorithm by boosting using feature combination," *IEEE Intelligent Vehicles Symposium*, pp. 364–368, 2007.

[30] X. M. Zhou and T. S. Dillon, "A statistical-heuristic feature selection criterion for decision tree induction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 8, pp. 834–841, 1991.

[31] F. Provost and T. Fawcett, *Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking*. O'Reilly Media, Inc., 2013.

[32] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Proceedings of the International Conference on Machine Learning*, 2008, pp. 1168–1175.

[33] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2014, pp. 701–710.

[34] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale information network embedding," in *Proceedings of the International Conference of World Wide Web*, 2015, pp. 1067–1077.

[35] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *Proceedings of the International Conference on Machine Learning*, 2016, pp. 40–48.

[36] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *Proceedings of the Web Conference*, 2018, pp. 499–508.

[37] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021.

[38] K. Sun, Z. Zhu, and Z. Lin, "AdaGCN: Adaboosting graph convolutional networks into deep models," in *Proceedings of the International Conference on Learning Representations*, 2021, pp. 1–12.

[39] H. Zhu, F. Feng, X. He, X. Wang, Y. Li, K. Zheng, and Y. Zhang, "Bilinear graph neural network with neighbor interactions," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2020, pp. 1452–1458.

[40] Z. Kang, H. Pan, S. C. H. Hoi, and Z. Xu, "Robust graph learning from noisy data," *IEEE Transactions on Cybernetics*, vol. 50, no. 5, pp. 1833–1843, 2020.

[41] X. Ai, C. Sun, Z. Zhang, and E. R. Hancock, "Two-level graph neural network," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14, 2022.

[42] S. Guan, X. Jin, Y. Wang, and X. Cheng, "Link prediction on n-ary relational data," in *Proceedings of the Web Conference*, 2019, pp. 583–593.

[43] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2017, pp. 1024–1034.

[44] Y. Chen, L. Wu, and M. J. Zaki, "Deep iterative and adaptive learning for graph neural networks," *arXiv:1912.07832*, 2019.

[45] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," in *Proceedings of the International Conference on Machine Learning*, vol. 97, 2019, pp. 2434–2444.

[46] J. Gan, R. Hu, Y. Mo, Z. Kang, L. Peng, Y. Zhu, and X. Zhu, "Multigraph fusion for dynamic graph convolutional network," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2022.

[47] L. Peng, R. Hu, F. Kong, J. Gan, Y. Mo, X. Shi, and X. Zhu, "Reverse graph learning for graph neural network," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2022.

[48] A. H. Sung and S. Mukkamala, "Identifying important features for intrusion detection using support vector machines and neural networks," in *Proceedings of the Symposium on Applications and the Internet*, 2003, pp. 209–217.

[49] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Computing Surveys*, vol. 50, no. 6, pp. 94:1–94:45, 2017.

[50] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth, 1984.

[51] Z. Zhou, *Ensemble Methods: Foundations and Algorithms*. CRC press, 2012.

[52] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[53] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[54] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *The Annals of Statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[55] R. Bekkerman, M. Bilenko, and J. Langford, *Scaling Up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press, 2011.

[56] P. Li, C. J. C. Burges, and Q. Wu, "McRank: Learning to rank using multiple classification and gradient boosting," in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2007, pp. 897–904.

[57] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[58] C. L. Giles, K. D. Bollacker, and S. Lawrence, "Citeseer: An automatic citation indexing system," in *Proceedings of the ACM International Conference on Digital Libraries*, 1998, pp. 89–98.

[59] A. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Information Retrieval*, vol. 3, no. 2, pp. 127–163, 2000.

[60] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.

[61] M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson, "FMA: A dataset for music analysis," in *Proceedings of the International Society for Music Information Retrieval Conference*, 2017, pp. 316–323.

[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[63] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec, "OGB-LSC: A large-scale challenge for machine learning on graphs," *arXiv:2103.09430*, 2021.

[64] W. Jin, T. Derr, Y. Wang, Y. Ma, Z. Liu, and J. Tang, "Node similarity preserving graph convolutional networks," in *Proceedings of the ACM International Conference on Web Search and Data Mining*, 2021, pp. 148–156.

[65] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations*, 2015, pp. 1–15.

[66] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[67] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proceedings of the International Conference on Machine Learning*, 2003, pp. 912–919.

[68] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of Machine Learning Research*, vol. 7, no. 11, pp. 2399–2434, 2006.

[69] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proceedings of the International Conference on Learning Representations*, 2019, pp. 1–17.

[70] F. Wu, A. H. S. Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *Proceedings of the International Conference on Machine Learning*, 2019, pp. 6861–6871.

[71] W. Lin, Z. Gao, and B. Li, "Shoestring: Graph-based semi-supervised classification with severely limited labeled data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 4173–4181.

[72] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *Proceedings of the International Conference on Learning Representations*, 2019, pp. 1–15.

[73] X. Yang, C. Deng, Z. Dang, K. Wei, and J. Yan, "SelfSAGCN: Self-supervised semantic alignment for graph convolution network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 775–16 784.

[74] C. Liu, L. Wen, Z. Kang, G. Luo, and L. Tian, "Self-supervised consensus representation learning for attributed graph," in *Proceedings of the ACM International Conference on Multimedia*, 2021, pp. 2654–2662.

[75] R. Fang, L. Wen, Z. Kang, and J. Liu, "Structure-preserving graph representation learning," *arXiv:2209.00793*, 2022.

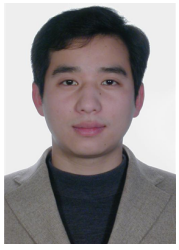
[76] K. Guo, K. Zhou, X. Hu, Y. Li, Y. Chang, and X. Wang, "Orthogonal graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022, pp. 3996–4004.

[77] L. Hu, S. Yang, X. Luo, and M. Zhou, "An algorithm of inductively identifying clusters from attributed graphs," *IEEE Transactions on Big Data*, vol. 8, no. 2, pp. 523–534, 2022.

[78] X. Luo, H. Wu, Z. Wang, J. Wang, and D. Meng, "A novel approach to large-scale dynamically weighted directed network representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 12, pp. 9756–9773, 2022.

Zhaogeng Liu is currently pursuing his Ph.D. degree in the School of Artificial Intelligence at Jilin University, supervised by Prof. Yi Chang. His current research interests include graph neural networks and feature selection.

Jielong Yang received the Ph.D. degree in Electrical and Electronic Engineering from Nanyang Technological University, Singapore, in 2020. He is currently an Assistant Professor in the School of Artificial Intelligence at Jilin University, P. R. China. He received the B.Eng. and M.Sc. degrees in Mechanical Engineering from Xi'an Jiaotong University in 2012 and 2014, respectively. His research interests include semi-supervised and unsupervised learning with bayesian networks and graph neural networks.



Xionghu Zhong received the B.Eng. and M.Sc. degrees from Northwestern Polytechnical University, China, in 2003 and 2006, respectively, and the Ph.D. degree from the Institute for Digital Communications, The University of Edinburgh, U.K., in 2010. He was a Research Fellow with the School of Computer Engineering and a Senior Research Fellow with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. He was with Xylem Inc., as a Data Scientist from 2017 to 2018. He is currently a Professor with the College of Computer Science and Electronic Engineering, Hunan University, China. His research interests include statistical signal processing, target localization and tracking, and machine learning methods, and their applications to distant speech enhancement and recognition, V2X communications, and water distribution network monitoring.

Wenwu Wang received the B.Sc. degree in 1997, the M.E. degree in 2000, and the Ph.D. degree in 2002, all from Harbin Engineering University, China. He then worked in King's College London, Cardiff University, Tao Group Ltd. (now Antix Labs Ltd.), and Creative Labs, before joining University of Surrey, UK, in May 2007, where he is currently a Professor in Signal Processing and Machine Learning, and a Co-Director of the Machine Audition Lab within the Centre for Vision Speech and Signal Processing. He is also an AI Fellow within the Surrey Institute for People Centred Artificial Intelligence. His current research interests include signal processing, machine learning and perception, artificial intelligence, machine audition (listening), and statistical anomaly detection. He has (co)-authored over 300 publications. His work has been funded by EPSRC, EU, Dstl, MoD, DoD, Home Office, Royal Academy of Engineering, National Physical Laboratory, BBC, and industry (including Samsung, Tencent, Huawei, Atlas, Saab, and Kaon). He is a Senior Area Editor (2019-2023) for IEEE Transactions on Signal Processing, an Associate Editor (2020-) for IEEE/ACM Transactions on Audio Speech and Language Processing, and an Associate Editor of Nature Scientific Report (2022-) and Senior Area Editor of Digital Signal Processing (2021-).

Hechang Chen is currently an associate professor of the School of Artificial Intelligence, Jilin University (JLU), China. He received his Ph.D. degree from the College of Computer Science and Technology, Jilin University (JLU), in December 2018. He was enrolled in the University of Illinois at Chicago (UIC) as a joint training Ph.D. student from November 2015 to December 2016 and enrolled in HongKong Baptist University (HKBU) as a visiting student from July 2017 to January 2018. He has published more than 40 articles in international journals and conferences, including IEEE TPAMI, TNNLS, KBS, EAAI, TKDD, IJCAI, SIGIR, ICDE, WWW, EMNLP, WSDM, ICDM, etc. His current research interests lie in the areas of machine learning, data mining, complex network analysis, deep reinforcement learning, and knowledge graph. He has served as a peer reviewer for several international journals and conferences, including IEEE TKDE, IEEE TCYB, IoT, IEEE TITS, STOTEN, IEEE TKDD, IODP, FCS, KDD, ICML, AAAI, IJCAI, WWW, WSDM, CIKM, ECML, KSEM, etc.

Yi Chang is the dean of the School of Artificial Intelligence, Jilin University (JLU). He became a Senior Member (SM) of IEEE in 2010, a Chinese National Distinguished Professor in 2017, and an ACM Distinguished Scientist in 2018. Before joining academia, he was a Technical Vice President at Huawei Research America, in charge of knowledge graph, question answering, and vertical search projects. Before that, he was at Yahoo Labs/Research from 2006 to 2016 as a research director and was in charge of search relevance of Yahoo's web search engine and vertical search engines. His research interests include information retrieval, data mining, machine learning, natural language processing, and artificial intelligence. He is the author of two books and more than 100 papers in top conferences or journals, and the associate editor of IEEE TKDE. He won the Best Paper Award on ACM KDD'2016 and ACM WSDM'2016. He served as one of the conference General Chairs for ACM WSDM'2018 and ACM SIGIR'2020.