Supervised Contrastive Learning With Structure Inference for Graph Classification

Junzhong Ji, Hao Jia, Yating Ren, and Minglong Lei

Abstract—Advanced graph neural networks have shown great potentials in graph classification tasks recently. Different from node classification where node embeddings aggregated from local neighbors can be directly used to learn node labels, graph classification requires a hierarchical accumulation of different levels of topological information to generate discriminative graph embeddings. Still, how to fully explore graph structures and formulate an effective graph classification pipeline remains rudimentary. In this paper, we propose a novel graph neural network based on supervised contrastive learning with structure inference for graph classification. First, we propose a data-driven graph augmentation strategy to enhance the existing connections. Concretely, we resort to a structure inference stage based on diffusion cascades to recover possible connections with high node similarities. Second, to improve the contrastive power of graph neural networks, we propose a supervised contrastive loss for graph classification. With the integration of label information, the one-vs-many contrastive learning is extended to a many-vs-many setting. The supervised contrastive loss and structure inference can be naturally incorporated within the hierarchical graph neural networks where the topological patterns can be fully explored to produce discriminative graph embeddings. Experiment results show the effectiveness of the proposed method compared with recent state-of-the-art methods.

Index Terms—Graph classification, graph neural networks, network inference, supervised contrastive learning.

I. INTRODUCTION

GRAPHS are universal data structures that can model many real-world systems with complex interactions, e.g., social networks, biological networks, and traffic networks. Graph classification is a fundamental machine learning task that can largely improve the understanding of the functions of those systems [1]. The goal of graph classification is to summarize the patterns in topological structures and node attributes to train a competitive model that can classify each graph into a category [2]. Successfully resolving this problem provides important support for many downstream applications, e.g., molecular property prediction [3], [4], brain disease classification [5], [6], point clouds classification [7], [8], and malware detection [9].

Early methods that resolve graph classification tasks are mainly based on graph kernels that define functions to measure the similarities between different graphs [10], [11]. With the structural similarities captured by graph kernels, the kernelized learning method (e.g., Support Vector Machines, SVMs) can be leveraged to perform the classification [12]. General options for graph kernels include random walk kernels [13], shortest-path kernels [14], and Weisfeiler-Lehman kernels [11]. However, these methods highly depend on hand-crafted kernels to extract structural patterns in graphs without explicit graph representations in most cases, which limits their integration with general classification models [15].

Recently, graph neural networks (GNNs) have been widely used in graph classification tasks. In general, GNNs use a local aggregation function to update node features by iteratively aggregating information from neighbors, and then employ a global readout function to summarize the node features as graph embeddings. The classification results can be easily obtained by applying an additional classification layer. From the local perspective, the information aggregations through edges are correlated with the feature forward process in different types of neural networks, such as, MLP [16], attention networks [17], and jumping connections [18]. From the global perspective, the acquisition of graph embeddings depends on pooling methods which progressively generate coarser graphs to capture hierarchical structures [19], [20], [21]. With carefully designed aggregation and readout functions, GNNs are as powerful as the Weisfeiler-Lehman isomorphism test [22].

Despite much progress of GNNs, thoroughly exploring the hierarchical structures and learning generalizable graph-level features for classification remains a challenge. To go beyond their limitations, recent methods use contrastive self-supervised learning to improve the feature extraction ability of GNNs [23], [24], [25], [26]. A common approach is using the graph augmentation to develop different views for graphs so that the contrastive objective can be easily constructed [27], [28]. However, on the one hand, the augmentation strategies in these methods are usually defined from underlying priors, e.g., node dropping and attribute masking. Such a formulation may bring about model biases toward specific graph structures or tasks. Besides, some of those methods are designed for node-level classifications, which may encounter difficulties when they are applied to graph classifications. On the other hand, the widely used contrastive learning frameworks for
graphs are usually established in a self-supervised manner. The contrastive losses in those methods are based on one-positive versus one-negative or many-negatives paradigms, where the contrastive power is limited.

In this paper, we address above problems by proposing a novel graph classification method that incorporates supervised contrastive learning with structure inference into GNNs (Sup-Cosine). The goal is to fully utilize the graph structures and label information to formulate an effective graph contrastive learning framework. We first develop a novel graph augmentation strategy to explore the global structures in graphs. The key idea is to simulate the diffusion processes to sample ordered node sequences named cascades, and then use structure inference to recover the augmented graphs from the cascades. The co-occurred nodes in the same cascade are assumed to have higher similarities, which will be endowed with higher probabilities of connections in the augmented graphs, and vice versa. The benefits of our augmentation strategy are two-fold: First, the inference process considers intra-graph contrastive information during the optimization, which equips the learned graph structures with more discriminative ability in a global view. Second, compared with previous methods, we build an optimization problem to construct the augmented graphs in a data-driven fashion, where no priors are required.

With the introduction of the graph augmentation strategy for structural information enhancement, we proceed to further leverage the label information to improve the graph contrastive learning. To improve the contrastive power, we propose to use a supervised contrastive loss to replace the original self-supervised loss. The label information is injected into the contrastive learning pipeline where the one-vs-one or one-vs-many contrastive paradigms are extended to many-vs-many ones. The introduction of labels can also discover the graph patterns that can hardly be distinguished by classifiers. The supervised contrastive loss with structure inference can be easily integrated into hierarchical GNNs to accomplish the graph classification task, where the supervised contrastive loss can be jointly optimized with the general classification loss over augmented graphs. The contributions of this paper can be summarized as follows:

- We propose a novel graph classification method that integrates supervised contrastive learning with structure inference into hierarchical GNNs, which improves the contrastive power from the structure and label sides.
- We propose a data-driven graph augmentation strategy based on structure inference, where the intra-graph contrastive information is explored to facilitate the graph classification.
- We extend the self-supervised graph contrastive learning framework to a supervised setting where the label information is introduced to further improve the contrastive power.
- Experiments on graph classification over several benchmark datasets show the effectiveness of the proposed model.

II. RELATED WORKS

A. Graph Neural Networks

GNNs are firstly proposed as a form of recursive networks [31]. Later methods mainly follow a message passing scheme where the node features can be updated by iteratively aggregating features from neighbor nodes [3], [32]. Many representative methods can be formulated within the framework, including vanilla graph convolutional networks (GCNs) [33], GraphSAGE [16], and graph attention networks (GATs) [17]. Limited by the over-smoothing issue, GNNs usually suffer from performance degradation as the number of neural layers increases. To improve the feature extraction abilities of GNNs, later methods focus on extending the local receptive field to encode high-order structures. For example, APPNP [34] utilizes Personalized Page Rank to obtain a better locality for a target node.

Another line of methods focuses on global graph representation learning, which can be directly used for graph classification tasks [2], [35], [36]. The key notion is developing a multi-scale feature extraction pipeline to leverage different levels of topological information in graphs [37], [38], [39]. The graph isomorphism network (GIN) is proposed as a graph classification method that is as powerful as the Weisfeiler-Lehman isomorphism test [22]. To accomplish graph-level classifications, the readout functions that produce graph embeddings should be permutation invariant and insensitive to matrix conversions. Graph pooling is a commonly-used method in GNNs to generate the permutation invariant embedding for a whole graph. A pooling method usually generates coarser graphs with new node sets by summarizing the node features and topology information [20]. A simple strategy is to select a subset of nodes from the original graph where the selected nodes have larger weights according to certain criteria [21], [40]. For example, the attention mechanism is usually used to determine the subset of nodes [37], [41], [42].

B. Contrastive Learning

Self-supervised learning is a promising paradigm that designs pre-text tasks to learn generalizable features, where the labels are constructed from the data itself [43]. Contrastive learning is a branch of self-supervised learning, which learns to compare the similarities of samples. Maximizing mutual information has been widely used in contrastive learning. Deep InfoMax [44] attempts to maximize the mutual information between the local patches and their global contexts in images. The main assumption is that the global features of images are supposed to have high similarities with their own local patch features, but have low similarities with local patches from other images. Data augmentations are essential in contrastive learning, which can generate different views from data. AutoAugment [45] automatically discovers data augmentation methods to improve the classification performance.

Contrastive learning has also been successfully applied to GNNs. Similar to images, it is natural to build local and global contrastive learning objectives in graphs. DGI [23] first
introduces the mutual information maximization into GNNs, which maximizes the mutual information between nodes and the whole graph. By utilizing different scales of structures in graphs, contrastive learning can equip GNNs with different properties. On the sub-graph level, GCC [46] introduces the InfoCE loss to implement contrastive learning. SAGE [37] separately leverages graph-instance level and hierarchical graph level information to accomplish the graph classification task. Other scales for contrastive learning include graph-graph level [27], graph-patch level [47], graph-node level [25], and node-node level [48]. GXN [21] develops a cross-scale layer to enable feature exchanges. It can improve the performance when serving as a graph pooling method.

III. OUR METHOD

A. Notations and Preliminaries

We begin our discussion by giving the notations used in this paper. A graph can be represented as \( G = (V, E) \), where \( V \) is a node set and \( E \) is an edge set. The adjacency and attribute matrices of the graph are denoted as \( A \in \mathbb{R}^{n \times n} \) and \( X \in \mathbb{R}^{n \times d} \), where \( n \) is the number of nodes and \( d \) is the dimension of node attributes. The neighbor set for node \( v_i \) is denoted as \( N(v_i) \). Given a dataset containing \( m \) graphs with labels \( D = \{(G_i, y_i)\}_{i=1}^{m} \), the goal of graph classification is to learn a prediction function \( F \) from the graph space \( \mathcal{G} \) to the label space \( \mathcal{Y} \).

GNNs usually follow a message propagation mechanism, where the node features are updated by aggregating attribute information of neighbors through graph structures. The layer-wise updating rule for a GNN model can be denoted as

\[
\begin{align*}
\mathbf{h}_i^{(0)} &= \mathbf{x}_i, \\
\mathbf{a}_{ij}^{(l)} &= \text{AGGREGATE}((\mathbf{h}_j^{(l-1)} | v_j \in \mathcal{S}(v_i)), \\
\mathbf{h}_i^{(l)} &= \text{UPDATE}(\mathbf{h}_i^{(l-1)}, \mathbf{a}_{ij}^{(l)}),
\end{align*}
\]

where \( \mathbf{x}_i \) is the attributed vector of node \( v_i \), \( \mathbf{h} \) is the embedding vector, \( l \) and \( l-1 \) are layer indices, AGGREGATE(·) is the aggregation function that accumulates the features of neighbors, UPDATE(·) is the updating function that combines the aggregated features and the feature of the node itself, and \( \mathcal{S}(v_i) \subset N(v_i) \) is a node set that contains all the neighbors that participate the aggregation.

The schematic illustration of the proposed method is presented in Fig. 1. In detail, our model contains three main components: a hierarchical GNN is used to learn graph embeddings, a supervised graph contrastive learning module is used to improve the contrastive power of the GNN for graph classification, and the structure inference is designed for graph augmentation. The three components will be introduced in the following subsections.

B. Hierarchical Graph Neural Networks

The skeleton of the proposed model is realized by a hierarchical GNN. Generally, the graph classification process can be broken down into a local aggregation step that updates the node features, a graph pooling step that hierarchically summarizes the node features into a graph embedding, and a classification step that maps the graph embedding to the label space.

1) Learning Node Features: The feature learning process for nodes is usually composed of standard local aggregation functions (see (1)) in GNNs. For graph classification, it is essential to combine structural information from different scales (e.g., neighbors and subgraphs) to learn node features. However, the standard local aggregation functions only consider structural information from direct neighbors, where the learned features are not necessarily suitable for graph classification. To this end, we design a mix-aggregation scheme that combines region-wise aggregation and layer-wise aggregation functions to learn node features. The region-wise aggregation directly leverages multi-hops of neighbor information from a subgraph so that the high-order structure information can be utilized. The layer-wise aggregation is similar to standard local aggregations that utilize directly connected neighbors layer by layer. Concretely, we use a region-wise aggregation layer followed by multiple layer-wise aggregation layers to learn node features.

The region-wise aggregation process defines a region of nodes that are most related to the target node. We can denote such a region of nodes as a subgraph. Notice that the constructed subgraph contains direct and multi-hop neighbors, which equips the region-wise aggregation with the ability to capture high-order information. To construct qualified subgraphs for target nodes, we resort to Breath-First-Search (BFS) following [26] to build subgraphs. For each node \( v_i \), we can obtain its corresponding subgraph \( g_i \) with the node set \( S_{\text{BFS}}(v_i) \). To facilitate the computation, the size of \( S_{\text{BFS}}(\cdot) \) is controlled by a hyper-parameter \( \beta \) which denotes the maximal nodes BFS can generate. We put the region-wise aggregation based on constructed subgraphs in the first layer. The propagation rule can be denoted as:

\[
\begin{align*}
\mathbf{a}_{i}^{(1)} &= \text{AGGREGATE}(\mathbf{x}_i | v_j \in S_{\text{BFS}}(v_i)), \\
\mathbf{h}_i^{(1)} &= \text{UPDATE}(\mathbf{x}_i, \mathbf{a}_{i}^{(1)}),
\end{align*}
\]

where \( \mathbf{h}_i^{(1)} \) is the updated node features for node \( v_i \), AGGREGATE is the aggregation function that accumulates neighbor features, and UPDATE combines the neighbor features and the feature of the node itself. We can then obtain the output of the first layer \( \mathbf{H}^{(1)} \). Notice that the number of nodes generated by BFS is relatively small, which ensures that the computation complexity remains low compared with a complete aggregation layer.

After the region-wise layer, we further stack several layer-wise aggregation layers to extract information from direct neighbors. To improve the discriminative ability of learned features, we follow [22] and leverage a powerful layer-wise aggregation function to obtain the node features:

\[
\mathbf{h}_v^{(l)} = \text{MLP}^{(l)} \left( 1 + \eta^{(l)} \mathbf{h}_v^{(l-1)} + \sum_{u \in N(v_i)} \mathbf{h}_u^{(l-1)} \right),
\]
where $l > 1$ is the layer index, $\gamma$ is a small number, $N(v_i)$ is the neighbor set of node $v_i$, and $\text{MLP}^{(l)}$ is the multi-layer perceptron used for feature transformation. After $L$ local aggregation layers, the node embeddings are updated as $Z = H^{(L)} \in \mathbb{R}^{n \times d_k}$, where $d_k$ is the node embedding dimension in the $L$-th layer.

2) Learning Graph Features: After obtaining the node features, we then summarize the node features to formulate the graph-level features. To guarantee that the graph features can capture hierarchical graph structures, we first apply a graph pooling layer to reduce the original graph $G$ as a coarser graph $\tilde{G}$. Consider the node set $V = \{v_i\}_{i=1}^n$, where we use a global ranking function with a linear projection operation to obtain the coarser graph:

$$idx = \text{TOP}_k \left( \frac{Zw}{\|w\|} \right),$$

where $idx$ denotes the indices of nodes for the coarsened graph, $w \in \mathbb{R}^{d_k \times 1}$ is a vector of learnable parameters that maps node features into 1 dimension, and $\text{TOP}_k(\cdot)$ is the top-$k$ function that selects $k$ largest values from the projected results. The results of the linear projection are normalized by the norm of the parameters. According to $idx$, we can construct the coarser graph $\tilde{G}$ with the node set $\tilde{V}$ and edge set $\tilde{E}$. To further leverage the subgraph information, we use the subgraphs generated by BFS to build the coarser graph, where each selected subgraph is treated as a super node. The edges of the coarser graph are determined by the number of overlapping nodes between two subgraphs.

$$\tilde{V} = \{ \tilde{v}_i; \tilde{v}_i = v_i, \forall i \in idx \},$$

$$\tilde{E} = \{ \tilde{e}_{ij}; |S_{BFS}(v_i) \cap S_{BFS}(v_j)| \geq \varepsilon \},$$

where $\tilde{v}$ and $\tilde{e}$ denote the node and edge in the coarser graph, and $\varepsilon$ is the threshold parameter. The representations of the super nodes are calculated by a linear transformation:

$$\tilde{z}_i = \sum_{-e_{ij} \in \tilde{E}} W_{e}z_j,$$

where $\tilde{z}_i$ is the feature vector of the super node $\tilde{v}_i$, $W_e \in \mathbb{R}^{d_s \times d_k}$ is the parameter matrix, and $d_s$ is the feature dimension for the super nodes.

We can then calculate the graph-level embeddings from the representations of the coarser graph via a readout function:

$$\mathbf{r} = \text{READOUT}(\{\tilde{z}_i\}_{i=1}^k),$$

where $\mathbf{r} \in \mathbb{R}^{d_s \times 1}$ is the embedding of the graph. In practice, the CONCAT operator is used as the readout function.

3) Graph Classification: The optimization objective for the graph classification is a cross-entropy loss which can be written as

$$\mathcal{L}_{gc} = \text{CrossEntropy}(\mathbf{Y}, \mathbf{P}),$$

where $\mathbf{Y}$ denotes the ground truth labels for graphs, and $\mathbf{P} \in \mathbb{P}$ denotes the predictions obtained from graph embeddings $\mathbf{r}$ via several classification layers.

C. Supervised Graph Contrastive Learning

In this subsection, we introduce a supervised graph contrastive loss to improve the performance of the graph classification. Generally, the graph embeddings learned by hierarchical GNNs only explore the interior information in each graph. However, to achieve high classification performance, it is necessary to further mine the external information between graphs. Graph contrastive learning is a powerful self-supervised learning paradigm, which learns to compare given different views of inputs. The incorporation of graph contrastive learning can improve the discriminative ability of graph embeddings and boost the classification performance.

The essential part of graph contrastive learning is to build contrastive pairs from the graphs. Previous methods sample from different scales of topological structures to formulate contrastive pairs. Based on the hierarchical GNNs, we build our contrastive loss from the global graph embeddings. As
indicated in Fig. 1, the positive and negative samples are separately drawn from the augmented graph \( G \), and are then fed into the hierarchical GNN to obtain graph embeddings. We first introduce the self-supervised contrastive loss based on graph embeddings:

\[
L_{self} = - \sum_{i \in \Omega} \log \frac{\exp(\mathbf{r}_i \cdot \mathbf{r}_{i(i)}/\tau)}{\sum_{k \in \Gamma(i)} \exp(\mathbf{r}_i \cdot \mathbf{r}_k/\tau)},
\]

where \( \Omega = \{1, \ldots, 2m\} \) represents the index set for all positive and negative samples in the data, \( \Gamma(i) = \Omega \setminus i \), \( \tau \) is a positive temperature parameter, \( \cdot \) is the dot product, \( i \) is the anchor, \( j(i) \) denotes the positive sample, and others are negative samples.

Since the label information is available in supervised learning scenarios, more positive samples can be included to improve the contrastive power. Under this scenario, the one-vs-many contrastive learning can be extended to a many-vs-many setting. In other words, compared with self-supervised contrastive learning, supervised contrastive learning has more positive and negative data for contrastive learning. To leverage label information, we extend the loss in (10) to a supervised graph contrastive (SupGCon) loss with the following form [30]:

\[
L_{self} = - \frac{1}{|\Phi(i)|} \sum_{i \in \Omega} \log \frac{\exp(\mathbf{r}_i \cdot \mathbf{r}_{p(i)}/\tau)}{\sum_{k \in \Gamma(i)} \exp(\mathbf{r}_i \cdot \mathbf{r}_k/\tau)},
\]

where \( \Phi(i) \) is the index set for all positive samples in a batch, \( |\Phi(i)| \) denotes the number of elements in the set. Based on (11), we can see that the gaps between different classes will be enlarged due to the integration of labels.

### D. Structure Inference

In this subsection, we mainly discuss the graph augmentation strategy based on structure inference. The goal of structure inference is to enhance the existing edges by discovering possible connections between nodes with high similarities. To capture node similarities, we resort to diffusion samplings [49] that simulate diffusion processes in graphs. It starts from a root node and then activates neighbor nodes at a certain rate. This process can generate ordered node sequences named cascades to capture node similarities. A cascade \( \mathbf{c} \) is represented as an \( n \)-dimensional vector \( (t_1, \ldots, t_n) \) where each element represents the time-stamp of the activated time for a given node. Only the time-stamps within a fixed time window \( T \) are considered. The time-stamps for inactivated nodes are denoted as \( \infty \). A set of cascades is represented as \( \mathbf{C} : (\mathbf{c}^1, \ldots, \mathbf{c}^l) \in \mathbb{R}^{q \times n} \) where \( q \) is the number of cascades.

To recover the possible connections, we formulate a learning objective that embeds the node similarities in cascades into connection weights [50]. Concretely, we use a bunch of observed cascades to infer a weighted connection matrix \( \mathbf{M} \) by maximizing the likelihood \( \phi(\mathbf{C}; \mathbf{M}) \). We begin by considering the inference problem of a single cascade \( \mathbf{c} \in \mathbf{C} \). Given the time window \([0, T]\), the inference problem for \( \mathbf{c} \) is \( \phi(\mathbf{c}; \mathbf{M}) \), which is a joint likelihood of activated nodes \( (t_i \leq T) \) and inactivated nodes \( (t_i > T) \). The pair-wise transmission from \( v_i \) to \( v_j \) can be modeled by a network model. For example, the exponential model can be denoted as

\[
f(t_j|t_i; \mathbf{M}_{ij}) = \begin{cases} M_{ij} \cdot e^{-M_{ij}(t_j-t_i)}, & \text{if } t_i < t_j, \\ 0, & \text{otherwise}. \end{cases}
\]

Then, the probability that \( v_j \) is activated by \( v_i \) can be represented as the joint probability that \( v_j \) is activated by \( v_i \) and is not activated by other already activated nodes in the cascade. \( \phi(t_j|t_i; \mathbf{M}) \) can be denoted as

\[
\phi(t_j|t_i; \mathbf{M}) = f(t_j|t_i; \mathbf{M}_{ij}) \prod_{t_k < t_j \atop t_k < t_i} (1 - f(t_j|t_k; \mathbf{M})).
\]

From (13), we can calculate \( \phi(t_j; \mathbf{M}) \) by summing up the likelihood \( \phi(t_j|t_i; \mathbf{M}) \) over \( t_i \):

\[
\phi(t_j; \mathbf{M}) = \sum_{t_i < t_j} \phi(t_j|t_i; \mathbf{M}) = \sum_{t_i < t_j} \left[ f(t_j|t_i; \mathbf{M}) \prod_{t_k < t_i \atop t_k < t_j} (1 - f(t_j|t_k; \mathbf{M})) \right].
\]

Then, the likelihood of observing all activations in a cascade \( \phi(\mathbf{c}^T; \mathbf{M}) \) is the joint probability of \( \phi(t_j; \mathbf{M}) \) for all \( t_j \leq T \):

\[
\phi(\mathbf{c}^T; \mathbf{M}) = \prod_{t_j \leq T} \phi(t_j; \mathbf{M}) = \prod_{t_j \leq T} \left( \sum_{t_i < t_j} \phi(t_j|t_i; \mathbf{M}) \right)
= \prod_{t_j \leq T} \left( \sum_{t_i < t_j} \left[ f(t_j|t_i; \mathbf{M}) \prod_{t_k < t_i \atop t_k < t_j} (1 - f(t_j|t_k; \mathbf{M})) \right] \right)
= \prod_{t_j \leq T} \left( \sum_{t_i < t_j} \frac{f(t_j|t_i; \mathbf{M})}{1 - f(t_j|t_i; \mathbf{M})} \prod_{t_k < t_j} (1 - f(t_j|t_k; \mathbf{M})) \right).
\]

The inactivated nodes \( (t_i > T) \) provide contrastive information when inferring the possible edges. Consequently, the likelihood function for a cascade \( \mathbf{c} \) can be represented as

\[
\phi(\mathbf{c}; \mathbf{M}) = \phi(\mathbf{c}^T; \mathbf{M}) \times \phi(\mathbf{c}^T; \mathbf{M})
= \prod_{t_j \leq T} \left( \sum_{t_i < t_j} \frac{f(t_j|t_i; \mathbf{M})}{1 - f(t_j|t_i; \mathbf{M})} \times \prod_{t_k < t_j} (1 - f(t_j|t_k; \mathbf{M})) \right)
\times \left( \prod_{t_j > T \atop t_j \leq T} \prod_{t_i < t_j} (1 - f(t_j|t_i; \mathbf{M})) \right).
\]

where \( \phi(\mathbf{c}^T; \mathbf{M}) \) denotes nodes \( v_i \) is not activated by already activated nodes \( v_j \) in the cascade.
Considering the independence of all cascades, the estimation over all observed cascades turns to $\phi(C; M) = \prod_{c \in C} \phi(c; M)$. The objective that required to be optimized is denoted as

$$\max_{M \geq 0} \sum_{c \in C} \log \phi(c; M). \quad (17)$$

Following the optimization process in [51], we can obtain the update rule on $M_{i,j}$:

$$M_{i,j} = \frac{1}{\Psi_c} \sum_{t_1 \leq T, t_2 < t_1} \frac{1}{t_1 - t_2} + \frac{1}{\Psi_c} \sum_{t_1 > T, t_1 < T} \frac{1}{t_2 - t_1}, \quad (18)$$

where $\Psi_c$ is a normalizer that calculates the total number of node pairs that satisfy the constraint. By solving (18), we can obtain the augmented affinity matrix.

The obtained connection matrix $M$ may contain noisy connections. As the elements in $M$ denote the connection weights, we simply add the edges with high probabilities to the adjacency matrix $A$, and remove the edges with low probabilities. To this end, we use a parameter $\xi$ to denote the number of edges that should be added or deleted in the original graph. We can formulate a new adjacent matrix by combining the original adjacent matrix and the learned connection matrix via $A' = A \oplus \phi_\xi(M)$, where $\oplus$ is the element-wise fusion operator, and $\phi_\xi$ is the function that selects augmented edges.

### E. Computational Complexity

The computational complexity of our model main comes from three parts. For network inference, the complexity is $O(n^2q)$, where $n$ is the number of nodes and $q$ is the number of cascades. For subgraph generation, the complexity is $O(n\beta)$ where $\beta$ is the upper limit of the BFS search. The complexity for GNNs is $O(Lnd^2)$, where $L$ is the number of convolution layers and $d$ is the feature dimension. In summary, the overall complexity is $O(n^2q + n\beta + Lnd^2)$.

### F. Algorithm

The overall optimization objective is the combination of the graph classification loss and supervised graph contrastive loss:

$$\mathcal{L} = \mathcal{L}_{gc} + \lambda \mathcal{L}_{sc}, \quad (19)$$

where $\lambda$ is a positive hyper-parameter. By jointly optimizing the graph classification and supervised graph contrastive losses, the learned graph embeddings are more discriminative for graph classification. The overall algorithm of the proposed model is listed in Algorithm 1.

### IV. EXPERIMENTS

To evaluate the effectiveness of the proposed model, we conduct experiments to show the improvements compared with previous graph classification methods. The code is available at https://github.com/drinks-single/SupCosine.

### A. Data Description

We use seven benchmark datasets, MUTAG, PTC, IMDB-BINARY, PROTEINS, ENZYMES, COLLAB, and REDDIT-MULTI-5 K, that are widely used in graph classification tasks to evaluate the proposed method. MUTAG [52] is a molecule graph dataset that contains 188 graphs representing nitro compounds, where each graph is classified as aromatic or heteroaromatic. PTC [53] is a chemical graph dataset that contains 344 chemical compounds, where each graph is labeled as toxic or atoxic. IMDB-BINARY [12] is a movie dataset where each node denotes an actor/actress, and the edges denote whether two actors/actresses are in the same movie. PROTEINS [54] is a dataset that consists of enzymes and non-enzymes. ENZYMES [54] contains 600 protein tertiary structures that can be classified into one of three categories. COLLAB [12] is a scientific collaboration dataset collected from three fields. The ego networks for researchers are constructed and the labels are determined by their research fields. REDDIT-MULTI-5K [12] is a social network dataset collected from five subreddits in Reddit. Each graph is generated from an online discussion thread and labeled by its subreddit. Details of the datasets are summarized in Table I.

### B. Baseline Methods

For comprehensive comparisons, we compare the proposed method with three types of graph methods:

1. Graph kernel methods: Weisfeiler-Lehman subtree kernel (WL) [11] runs the Weisfeiler-Lehman test on two graphs and counts the number of aggregated labels in each node.

---

**Algorithm 1:** The Training Process of SupCosine

**Input:** Graphs with labels $D = \{(G_i, y_i)\}_{i=1}^m$; coefficient of SupGCon loss $\lambda$; Up-limit of BFS $\beta$; Number of epochs $epochs$, batch size $B$

**Output:** Model parameters $W$

1. for $G \in D$ do
2.  Cascade Set $C \leftarrow$ simulation($A$);
3.  $M \leftarrow$ Eq. (17); // Structure Inference
4.  $A' = A \oplus \phi(M)$;
5. end
6. $B, \hat{B} \leftarrow$ sampling($D$);
7. for epoch $< epochs$ do
8.  for $G, \hat{G} \in B, \hat{B}$ do
9.    $H^i, H^j \leftarrow$ Eq. (2); // Region-wise agg.
10.   $Z, \hat{Z} \leftarrow$ Eq. (3); // Layer-wise agg.
11. end
12. $iidx \leftarrow$ Eq. (4);
13. $\bar{Z} \leftarrow$ Eq. (5), Eq. (6), and Eq. (7);
14. $r \leftarrow$ Eq. (8); // Top k nodes
15. end
16. $\mathcal{L}_{gc} \leftarrow$ Eq. (9); // Cross Entropy Loss
17. $\mathcal{L}_{sc} \leftarrow$ Eq. (11); // SupGConLoss
18. $\mathcal{L} \leftarrow$ Eq. (19);
19. Update $W \leftarrow$ Backward($\mathcal{L}$);
20. end
21. Return $W$
Such a process can be viewed as comparing the number of shared subtrees that are decomposed from graphs. Graphlet kernel (GK) [55] is a kernel method based on subgraphs with limited sizes. The frequencies of graphlets are formulated as normalized vectors to compute the kernel. Deep Graph Kernels (DGK) [12] is a kernel method based on return probabilities from random walks. Wasserstein embedding for graph learning (WEGL) [57] uses Wasserstein distance to calculate the distances of nodes between different graphs, and defines the graph similarities as node distribution similarities.

(2) Graph neural network methods: GraphSAGE [16] is a graph representation learning method that leverages sampling and aggregation to learn node embeddings. Different aggregation functions are proposed to accumulate the neighbor information. GIN [22] proposed a GNN model that is as powerful as the Weisfeiler-Lehman test. DAGCN [58] is a dual-attention GCN that uses the attention mechanism in both the aggregation and pooling steps. PPGN [59] designs a GNN based on the message-passing mechanism, which encodes unique node identities into node embeddings so that the permutation equivariance can be preserved. CapsGNN [60] applies the idea of capsule to GNNs, where the routing mechanism is used to extract important information from the graph perspective. SAGPool [24] proposed different orders of subgraph networks to consider different levels of structure information. SAGPool [42] is a graph pooling method constructed over GNNs, which uses a self-attention mechanism to formulate graph representations. NDP [61] is a node decimation pooling method for GNNs that keeps the overall topology of graphs when generating coarser graphs. Graph Multiset Transformer (GMT) [41] treats graph pooling as a multiset encoding operation. Multi-head attention is used in the graph pooling layer.

(3) Graph contrastive learning methods: GraphCL [27] is a self-supervised learning framework for graph representation learning. The contrastive loss between different views of subgraphs is optimized. InfoGraph [47] is a self-supervised learning method that develops an unsupervised graph-level feature learning framework based on deep InfOrMax. M-GCL [25] uses contrastive learning with different views to learn node and graph level embeddings. Graph Cross Network (GXN) [21] uses multiscale structure information in graphs for graph representation learning, and builds cross layers to enable information interchange between different scales. sGIN [48] improves the GIN model by extending the neighbors of each node. SUGAR [26] presents a GNN based on subgraphs. By maximizing the mutual information of subgraph features, more global information can be captured. Triplet [62] and N-pair loss [63] can be used to build contrastive learning frameworks, where the former builds the supervised contrastive loss with one positive sample and one negative sample, whereas the latter builds contrastive loss with one positive sample and many negative samples. We leverage GIN as the feature extraction backbone to formulate two graph contrastive learning models based on the above losses.

C. Implementation Details

We use the RMSprop optimizer with $\alpha = 0.9$ and weight decay $0.001$. We adopt 5 layer-wise aggregation layers with 2 MLP layers. The dimensions of all hidden layers are set to 16. The aggregation function for all node-level representation learning methods based on GNNs is set as the sum function. The temperature $\tau$ of the supervised graph contrastive loss is set to 0.07. For structure inference, the time window $\tau$ is set to 10, the type of diffusion is set as the exponential model, and the number of cascades is set to equal to the number of nodes for each graph. The coefficient $\lambda$ is selected from $[0.005, 0.01, 0.02, 0.03]$. For each dataset, the upper limit of BFS $\beta$ is selected from $[3, 5, 7]$. The number of augmented edges is selected from $[1, 2, 3]$ for MUTAG and PTC, and from $[3, 5, 7]$ for other datasets. The positive samples are selected from the samples that have the same labels with the target sample in a training batch, while the negative samples are randomly selected from the samples that have different labels.

D. Classification Results

In this subsection, we first compare the performance of our method with several strong baselines for binary classification.

TABLE I

| Dataset | #Classes | #Graphs | Avg. Nodes | Max. Nodes | Node Labels |
|---------|----------|---------|------------|------------|-------------|
| MUTAG   | 2        | 188     | 17.93      | 28         | 7           |
| PTC     | 2        | 344     | 25.56      | 109        | 19          |
| IMDB    | 2        | 1000    | 19.77      | 136        | –           |
| PROTEINS| 2        | 1113    | 39.06      | 620        | 3           |
| ENZYMES | 6        | 600     | 32.63      | 126        | 3           |
| COLLAB  | 3        | 5000    | 74.49      | 492        | –           |
| REDDIT  | 5        | 5000    | 508.52     | 3648       | –           |

TABLE II

| Method    | MUTAG | PTC | PROTEINS | IMDB |
|-----------|-------|-----|----------|------|
| WL [11]   | 83.8±1.5 | –   | 74.7±0.5 | 73.4±1.6 |
| GK [55]   | 83.5±0.6 | 59.7±0.3 | –       | –     |
| DGK [12]  | 87.4±2.7 | 60.1±2.6 | 75.7±0.5 | –     |
| RetGK [56] | 90.3±1.1 | 67.9±1.4 | 75.8±0.6 | 72.3±0.6 |
| WEGL [57] | –     | 67.5±7.7 | 76.5±4.2 | 75.4±5.0 |
| GraphSAGE [16] | 79.8±13.9 | –   | 65.9±2.7 | 72.4±3.6 |
| GIN [22]  | 89.4±5.6 | 64.6±7.0 | 76.2±2.8 | 75.1±5.1 |
| DAGCN [58] | 87.2±6.1 | 62.9±9.6 | 76.3±4.3 | –     |
| PPGN [59] | 90.6±8.7 | 66.2±6.5 | 77.2±4.7 | 73.0±5.8 |
| CapsGNN [60] | 86.7±6.9 | –   | 76.3±3.6 | 73.1±4.8 |
| SGN [38]  | 89.5±7.4 | 64.1±3.7 | 76.3±4.1 | 76.5±5.7 |
| SAGPool [42] | 76.8±2.1 | –   | 72.0±1.1 | 72.2±0.9 |
| NDP [61]  | 87.9±5.7 | –   | 73.4±1.8 | –     |
| GMT [41]  | 83.4±1.3 | –   | 75.1±0.6 | 73.48±0.8 |
| GraphCL [27] | 86.8±1.3 | –   | 74.4±0.5 | 71.1±0.4 |
| InfoGraph [47] | 89.0±1.1 | 61.7±1.4 | –   | 73.0±0.9 |
| M-GCL [25] | 89.7±1.1 | 62.5±1.7 | –   | 74.2±0.7 |
| GXN [21]  | 86.1±8.3 | 63.5±5.8 | 79.9±4.1 | 78.6±2.3 |
| sGIN [48] | 94.1±2.7 | 73.6±4.3 | 79.0±3.2 | 77.9±4.3 |
| SUGAR [26] | 96.7±4.6 | 77.5±2.8 | 81.3±0.9 | 73.0±3.5 |
| SupCosine | 98.3±2.3 | 87.8±10.4 | 80.0±3.6 | 83.0±3.2 |
on four datasets and report the results in Table II. The best results are marked in bold, and "-" denotes that the results are not available.

As shown in Table II, the accuracy of our method shows strong competitiveness with stable standard variance and achieves the best performance on most datasets. In detail, our method achieves 1.0%, 10%, and 4.4% improvements compared with the second-best methods in MUTAG, PTC, and IMDB-BINARY, respectively. Specially, compared with recent graph contrastive learning methods, our model achieves obvious performance improvements. The comparison with those graph contrastive learning methods shows the effectiveness of our strategies. When comparing with the most recent method, SUGAR, our method can obtain a maximal 10% improvement on several datasets. All these improvements come from the conspicuous identifiability of the network inference strategy and supervised contrastive learning process.

To further verify the effectiveness of the proposed model, we conduct additional experiments for multi-class graph classification on three datasets, namely, ENZYMES, COLLAB, and REDDIT-MULTI-5 K. For simplicity, we select four GNN-based methods that are closely related to our method as baselines. The feature extraction backbones of GIN and SUGAR are similar to ours. Triplet and N-pair loss are two methods that have one-vs-one and one-vs-many contrastive settings. To evaluate the stability of performance, we vary the ratio of training samples from 10% to 90% for ENZYMES and COLLAB and from 5% to 30% for REDDIT-MULTI-5 K. For REDDIT-MULTI-5 K, because the performance variations among different training ratios are small, up to 30% of samples are selected for training. We use two metrics, Micro-f1 and Macro-f1, to evaluate the effectiveness of the proposed model in the multi-class setting. The results of our method and the baselines on the three datasets are separately reported in Figs. 2, 3, and 4.

It can be observed from these figures that the proposed model outperforms all baseline models on the three datasets with respect to Micro-f1 and Macro-f1 scores. Particularly, the performance superiority over the three contrastive learning methods demonstrates the effectiveness of the supervised graph contrastive loss. Our model is more stable than Triplet and N-pair loss, which shows the advantages of the many-vs-many contrastive loss compared with one-vs-one and one-vs-many losses.

E. Ablation Studies

In this subsection, we conduct ablation experiments to separately demonstrate the roles of structure inference (StruInf) and supervised graph contrastive loss (SupGCon). The classification results on four datasets, MUTAG, PTC, PROTEIN, and IMDB-BINARY, are shown in Fig. 5. Base is the method that removes SupGCon and StruInf from SupCosine. +StruInf and +SupGCon denote the models that add the corresponding components in the Base model.

On the one hand, the results show that both strategies can separately improve the classification performances compared with the Base model, which shows the effectiveness of these components. In most cases, the performance gains obtained by StruInf are slightly higher than SupGCon. However, the model with StruInf only enhances the graphs and learns the embeddings with the original feature extractor. The information within the enhanced graphs cannot be adequately captured, resulting in limited performance improvements compared with SupCosine. On the other hand, SupGCon and StruInf have been nicely incorporated within the GNN framework and the classification performance has been largely improved when both strategies are employed. The performance superiority after adding the above strategies is more significant in PTC dataset. It can be observed that the structural information enhanced by StruInf can be properly utilized by a powerful graph classification framework with the SupGCon component.

F. Model Analysis

In this subsection, we give more detailed analyses of our model. Several experiments are conducted to demonstrate other properties of SupCosine.
First, we plot the training curves of our method and GIN in Fig. 6. As shown in this figure, our method starts from higher accuracies and is more stable during the training process. In contrast, the training curve of GIN fluctuates dynamically as the training proceeds. The training curve of GIN reaches nearly 100% after 100 epochs, which may suffer from over-fitting risks. The steady training process of our method mainly benefits from the structure inference process which refines the salient connections within the graphs. Besides, the combination of supervised graph contrastive learning equips the graph embeddings with more discriminative ability, which also facilitates the convergence of the training process.

Second, to show the different roles of the region-wise aggregation and layer-wise aggregation, we compare the performance of the model with only a region-wise aggregation layer and the model with only layer-wise aggregation layers. The results are shown in Fig. 7. It can be observed that region-wise and layer-wise aggregations alone can hardly accumulate sufficient information from graphs. Both models cannot achieve satisfactory performance in the graph classification task. In contrast, SupCosine combines both aggregation schemes and is more powerful in extracting structure information. The results demonstrate the effectiveness of the mix-aggregation scheme that is used in our model.

Third, we visualize the adjacent matrix before and after the structure inference process in Fig. 8 to show how this process improves the structural information in graphs. We select a graph in the MUTAG dataset and visualize the original and inferred adjacency matrices in Fig. 8(a) and (b), respectively. Light colors represent connections while dark colors represent disconnections. It is noticeable that the inferred matrix has more connections and contains more structural information. For example, node 8 has more connections in the inferred graph than these in the original graph. When combined with the classification results, it can be observed that the inferred matrices can improve the graph classification performance.

G. Parameter Analysis

In this subsection, we present the performance of our model over different hyper-parameters. The following five hyper-parameters are considered: the number of new edges after the structure inference phase, $C_2$, in (19), the number of layer-wise aggregation layers, the number of cascades $q$, and the cascade length $T$.

First, we analyze the influence of the number of edges added during the structure inference step. The performance of different numbers of edges on MUTAG, PTC, and IMDB-BINARY datasets are plotted in Fig. 9. We vary the number
of added edges from 0 to 5. For all the three datasets, the highest accuracy is achieved when the number is set to 1. Since the graphs are relatively sparse, the number of average edges for each node is small. When too many edges are added, the graph features are prone to be smoothed and are then less discriminative for the classification. Even though, our method remains robust when three or more edges are added, which proves the stability of SupCosine.

Second, Fig. 10 shows the influence of coefficient $\lambda$ that controls the weight of the SupGCon loss during the optimization. This coefficient controls the importance of SupGCon in the final optimization objective. The results show that the optimal $\lambda$ differs on different datasets. For example, the optimal value is 0.01 for PTC, and 0.02 for IMDB-BINARY. The performance on all datasets decreases with large values $\lambda$, which indicates that the excessive concerns on contrastive learning can be harmful to the classification performance.

Third, Fig. 11 gives the performance of our method with different numbers of layer-wise aggregation layers. We vary the number of layers from 3 to 7 and report their performance on MUTAG, IMDB-BINARY, and PTC datasets. For MUTAG and PTC, the models with 4 layers obtain the best performance. For IMDB-BINARY, the model with 5 layers achieves the best results. With fewer layers, the model cannot fully extract the structural information. Meanwhile, too many layers may lead to over-smoothing and generate indistinguishable node embeddings that ultimately degrade the classification performance.

Finally, we analyze the influence of the two hyper-parameters that are used during the diffusion process: the number of cascades $q$ and cascade length $T$. The results under different settings are reported on the MUTAG dataset. Because $q$ is highly related to the number of nodes for each graph, we set it to 25%, 50%, 75%, 100%, and 200% of the number of nodes. $T$ is selected from {2, 4, 8, 10, 15, 20, 50, 100}. The performance of the two hyper-parameters is reported in Fig. 12. For $q$, the best performance is achieved when $q$ equals the number of nodes for each graph. Notably, when $q$ is small, it is difficult to discover new edges, which affects the data augmentation strategy. When $q$ is too large, noisy connections will be introduced to the inferred graph, which is also harmful to the classification performance. To this end, we set $q$ as the number of nodes for each graph. For $T$, the best performance is achieved when it is near or equal to 10. Similarly, smaller or larger values of $T$ affect the quality of inferred graphs and eventually degrade the classification performance. For simplicity, we set the cascade length to 10 for all datasets.
IV. CONCLUSION

In this paper, we studied the challenges of graph contrastive learning in graph classification, and pointed out that the structure information and label information have not been fully explored in existing methods. From this perspective, we proposed a supervised contrastive learning model with structure inference to incorporate the above information within a hierarchical graph neural network framework. Compared with previous graph contrastive learning methods, our advantages are two-fold: First, we introduced a data-driven graph augmentation strategy to enhance the existing edges, from which the contrastive pairs can carry more topological information. Second, we extended the self-supervised contrastive loss to a supervised setting by integrating the label information into the loss. The one-vs-many contrast was then extended to many-vs-many contrasts so that the contrastive power of the model can be largely improved.

REFERENCES

[1] J. Zhou, J. Shen, S. Yu, G. Chen, and Q. Xuan, “M-evolve: Structural-mapping-based data augmentation for graph classification,” IEEE Trans. Netw. Sci. Eng., vol. 8, no. 1, pp. 190–200, Jan.–Mar. 2021.

[2] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, “An end-to-end deep learning architecture for graph classification,” in Proc. AAAI Conf. Artif. Intell., 2018, pp. 4438–4445.

[3] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” in Proc. Int. Conf. Mach. Learn., 2017, pp. 1261–1272.

[4] Z. Shuai and G. Carpyris, “Heterogeneous molecular graph neural networks for predicting molecule properties,” in Proc. IEEE Int. Conf. Data Mining, 2020, pp. 492–500.

[5] S. Parisot et al., “Disease prediction using graph convolutional networks: Application to autism spectrum disorder and Alzheimer’s disease,” Med. Image Anal., vol. 48, pp. 117–130, 2018.

[6] X. Li et al., “BrainGNN: Interpretable brain graph neural network for fMRI analysis,” Med. Image Anal., vol. 74, 2021, Art. no. 102233.

[7] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. Bronstein, and J. M. Solomon, “Dynamic graph CNN for learning on point clouds,” ACM Trans. Graph., vol. 38, no. 5, pp. 1–12, 2019.

[8] E. Nezhadahra, E. Taghavi, R. Razani, B. Liu, and J. Luo, “Adaptive hierarchical down-sampling for point cloud classification,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2020, pp. 12956–12964.

[9] Z. Zhang, Y. Li, H. Dong, H. Gao, Y. Jin, and W. Wang, “Spectral-based directed graph network for malware detection,” IEEE Trans. Netw. Sci. Eng., vol. 8, no. 2, pp. 957–970, Apr.–Jun. 2021.

[10] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, “Graph kernels,” J. Mach. Learn. Res., vol. 11, pp. 1201–1242, 2010.

[11] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, “Dynamic graph CNN for learning on point clouds,” ACM Trans. Graph., vol. 38, no. 5, pp. 1–12, 2019.

[12] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, “Graph contrastive learning with adaptive augmentation,” in Proc. Web Conf., 2021, pp. 2069–2080.

[13] I. Brugere, B. Gallagher, and T. Y. Berger-Wolf, “Network structure inference, a survey,” vol. 51, no. 2, pp. 1–19, 2018.

[14] P. Khosla et al., “Supervised contrastive learning,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2020, Art. no. 1567.

[15] F. Scarselli et al., “The graph neural network model,” IEEE Trans. Neural Netw., vol. 20, no. 1, pp. 61–80, Jan. 2009.

[16] J. Chen et al., “A simple yet effective layerd loss for pre-training of network embedding,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 2, pp. 1827–1837, May/Jun. 2022.

[17] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proc. Int. Conf. Learn. Representations, 2017, pp. 1–14.

[18] J. Klüpera, A. Bojchevski, and S. Günnemann, “Predict then propagate: Combining neural networks with personalized PageRank for classification on graphs,” in Proc. Int. Conf. Learn. Representations, 2019, pp. 1–15.

[19] S. Zhu, L. Zhou, S. Pan, C. Zhou, G. Yan, and B. Wang, “GSSNN: Graph smoothing splines neural networks,” in Proc. AAAI Conf. Artif. Intell., 2020, pp. 7007–7014.

[20] J. Wang et al., “Sampling subgraph network with application to graph classification,” IEEE Trans. Netw. Sci. Eng., vol. 8, no. 4, pp. 3478–3490, Oct.–Dec. 2021.

[21] J. Li, Y. Rong, H. Cheng, H. Meng, W. Huang, and J. Huang, “Semi-supervised graph classification: A hierarchical graph perspective,” in Proc. World Wide Web Conf., 2019, pp. 972–982.

[22] Q. Xuan et al., “Graph contrastive learning with jumping knowledge networks,” in Proc. AAAI Conf. Artif. Intell., 2020, pp. 5453–5462.

[23] Z. Zhang et al., “Hierarchical graph pooling with structure learning,” arXiv:1911.05954.

[24] J. Baek, M. Kang, and S. H. Hwang, “Accurate learning of graph representations with graph multi-task pooling,” in Proc. Int. Conf. Learn. Representations, 2021, pp. 1–22.

[25] P. Velickovic, V. Fedus, W. L. Hamilton, P. Liu, Y. Bengio, and R. D. Hjelm, “Deep graph infomax,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2020, pp. 14093–14105.

[26] X. Liu et al., “Self-supervised learning: Generative or contrastive,” Proc. Int. Conf. Lear. Represent., 2019, pp. 1–17.

[27] J. Lee, I. Lee, and J. Kang, “Self-attention graph pooling,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2020, pp. 5820–5829.

[28] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, “Graph convolutional networks with EigenPooling,” in Proc. 25th ACM SIGKDD Int. Knowl. Discov. Data Sci. Conf., 2019, pp. 723–731.
[45] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, “AutoAugment: Learning augmentation strategies from data,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2019, pp. 113–123.

[46] J. Qiu et al., “GCC: Graph contrastive coding for graph neural network pre-training,” in Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2020, pp. 1150–1160.

[47] F.-Y. Sun, J. Hoffmann, V. Verma, and J. Tang, “InfoGraph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization,” in Proc. Int. Conf. Learn. Representations, 2020, pp. 1–16.

[48] X. Di, P. Yu, R. Bu, and M. Sun, “Mutual information maximization in graph neural networks,” in Proc. Int. Joint Conf. Neural Netw., 2020, pp. 1–7.

[49] Y. Shi, M. Lei, H. Yang, and L. Niu, “Diffusion network embedding,” Pattern Recognit., vol. 88, pp. 518–531, 2019.

[50] M. G. Rodriguez et al., “Uncovering the temporal dynamics of diffusion networks,” in Proc. Int. Conf. Mach. Learn., 2011, pp. 561–568.

[51] Q. Zhang, J. Wu, P. Zhang, G. Long, I. W. Tsang, and C. Zhang, “Inferring latent network from cascade data for dynamic social recommendation,” in Proc. IEEE 16th Int. Conf. Data Mining, 2016, pp. 669–678.

[52] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, “Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity,” J. Med. Chem., vol. 34, no. 2, pp. 786–797, 1991.

[53] C. Helma, R. D. King, S. Kramer, and A. Srinivasan, “The predictive toxicochemistry challenge 2000–2001,” Bioinformatics, vol. 17, no. 1, pp. 107–108, 2001.

[54] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, “Protein function prediction via graph kernels,” Bioinformatics, vol. 21, no. 1, pp. i47–i56, 2005.

[55] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, “Efficient graphlet kernels for large graph comparison,” in Proc. Int. Conf. Artif. Intell. Statist., 2009, pp. 488–495.

[56] Z. Zhang, M. Wang, Y. Xiang, Y. Huang, and A. Nehorai, “RetGK: Graph kernels based on return probabilities of random walks,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2018, pp. 3968–3978.

[57] S. Kolouri, N. Naderializadeh, G. K. Rohde, and H. Hoffmann, “Wasserstein embedding for graph learning,” in Proc. Int. Conf. Learn. Representations, 2021, pp. 1–20.

[58] F. Chen, S. Pan, J. Jiang, H. Luo, and G. Long, “DAGCN: Dual attention graph convolutional networks,” in Proc. Int. Joint Conf. Neural Netw., 2019, pp. 1–8.

[59] C. Vignac, A. Loukas, and P. Frossard, “Building powerful and equivariant graph neural networks with structural message-passing,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2020, pp. 14143–14155.

[60] Z. Xinyi and L. Chen, “Capsule graph neural network,” in Proc. Int. Conf. Learn. Representations, 2018, pp. 1–16.

[61] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, “Hierarchical representation learning in graph neural networks with node decimation pooling,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 5, pp. 2195–2207, May 2022.

[62] K. Q. Weinberger and L. K. Saul, “Distance metric learning for large margin nearest neighbor classification,” J. Mach. Learn. Res., vol. 10, pp. 207–244, 2009.

[63] K. Sohn, “Improved deep metric learning with multi-class N-pair loss objective,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2016, pp. 1857–1865.

Jinzhuong Ji is currently a Professor and the Ph.D. Supervisor with the Faculty of Information Technology, Beijing University of Technology, Beijing, China, and Youth Skeleton Teacher in Beijing and Senior Member of the China Computer Federation, Beijing. He has authored or coauthored more than 150 articles in these areas. His research interests include data mining, machine learning, swarm intelligence, bioinformatics and brain image intelligence analysis.

Hao Jia received the bachelor’s degree in thermal energy and dynamic engineering from the Shanghai University of Electric Power, Shanghai, China, in 2016. He is currently working toward the master’s degree in computer science and technology with the Beijing University of Technology, Beijing, China. His research interests include machine learning and graph data mining.

Yating Ren received the bachelor’s degree in information and computing science from Hebei Finance University, Baoding, China, in 2019. She is currently working toward the master’s degree in computer science and technology with the Beijing University of Technology, Beijing, China. Her research interests include machine learning, graph mining and brain science.

Minglong Lei received the Ph.D. degree in computer science from the University of Chinese Academy of Sciences, Beijing, China, in 2019. He is currently a Lecturer with the Beijing University of Technology, Beijing. His research interests include machine learning and data mining.