

Masked Graph Auto-Encoder Constrained Graph Pooling

Chuang Liu^{1*}, Yibing Zhan², Xueqi Ma³, Dapeng Tao⁴,
Bo Du¹, and Wenbin Hu¹ (✉)

¹ School of Computer Science, Wuhan University, Wuhan, China

² JD Explore Academy,

³ School of Software, Tsinghua University, Beijing, China

⁴ Yunnan University, Kunming, China

{chuangliu,dubo,hwb}@whu.edu.cn, zhanyibing@jd.com,
xueqima@s.upc.edu.cn, dptao@ynu.edu.cn

Abstract. The node drop pooling is a significant type of graph pooling that is required for learning graph-level representations. However, existing node drop pooling models still suffer from the information loss problem, impairing their effectiveness in graph classification. To mitigate the detrimental effect of the information loss, we propose a novel and flexible technique called Masked Graph Auto-encoder constrained Pooling (MGAP), which enables vanilla node drop pooling methods to retain sufficient effective graph information from both node-attribute and network-topology perspectives. Specifically, MGAP reconstructs the original node attributes of the graph using a graph convolutional network and the node degree of the graph (i.e., structural information) using a feedforward neural network with exponential neurons from the pooled (masked) graphs generated by the vanilla node drop pooling models. Notably, MGAP is a plug-and-play technique that can be directly adopted in the current node drop pooling methods. To evaluate the effectiveness of MGAP, we conduct extensive experiments on eleven real-world datasets by applying MGAP to three commonly-used methods, i.e., TopKPool, SAGPool, and GSAPool. The experimental results reveal that MGAP has the capacity to consistently improve the performance of all the three node drop pooling models in the graph classification task.

Keywords: graph neural networks · graph pooling · graph auto-encoder · graph classification.

1 Introduction

Graph Neural Networks (GNNs) have demonstrated their significant effectiveness in a variety of graph classification tasks [6,2], including molecular property prediction [1], cancer diagnosis [26], and brain-data analysis [17]. In contrast to node-level tasks (e.g., the node classification), which mainly leverage the graph

* This work has been done when Chuang Liu was an intern at JD Explore Academy.

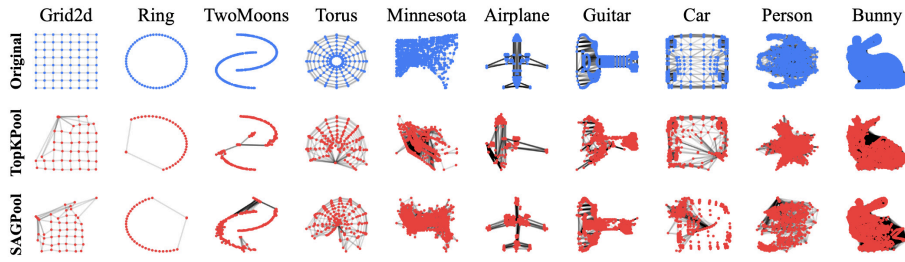


Fig. 1. Graph Reconstruction with two typical node drop pooling operators. The point locations in the figure above represent the node attributes.

convolutional network (GCN) to generate node representations for downstream tasks [13], graph classification requires obtaining holistic graph-level representations. Therefore, for graph classification, the pooling mechanism is an essential component that condenses the input graph with GCN-learned node representations into a single vector or a coarser graph with a smaller size.

Early adopted graph pooling techniques such as average and maximum pooling disregard node correlations, hence restricting overall performance [5,40]. Subsequently, graph pooling utilizes hierarchical architecture to model the node correlations [20,34] and can be roughly classified into node clustering pooling and node drop pooling. Node clustering pooling requires clustering nodes into new nodes, which is time- and space-consuming [37,3,38]. In comparison, node drop pooling preserves only representative nodes by assessing their importance, and is hence more efficient and suitable for large-scale networks [7,14,39].

Although efficient and effective, the current node drop pooling methods are affected by information loss, resulting in suboptimal graph-level representations and unsatisfactory performance in the graph classification task. To substantiate the above idea, we conduct experiments on graph reconstruction to directly quantify the amount of retained information after pooling. Specifically, we employ two node drop pooling algorithms (i.e., TopKPool [7] and SAGPool [14]) on 10 synthetic point cloud graphs. The experimental settings are consistent with those proposed in prior research [3,1]. Fig. 1 depicts the reconstructed results of the point cloud’s original attributes (i.e., coordinates) from its pooled graph, which is generated by the node drop pooling operators. As shown in Fig. 1, node drop pooling approaches frequently fail to recover the original graph signal, indicating that they discard part of critical graph information, which explains their inferior performance in graph classification.

We provide an intuitive explanation for the aforementioned phenomenon as follows. Indeed, nodes connected in a graph typically share similar attributes [22], and their similarity rises further after message propagation using GNNs (such as GCN [13] or GAT [33]). Node drop pooling methods, such as TopKPool and SAGPool, generate node scores based on the node attributes, resulting in a high potential for most selected nodes to share similar attributes or to be

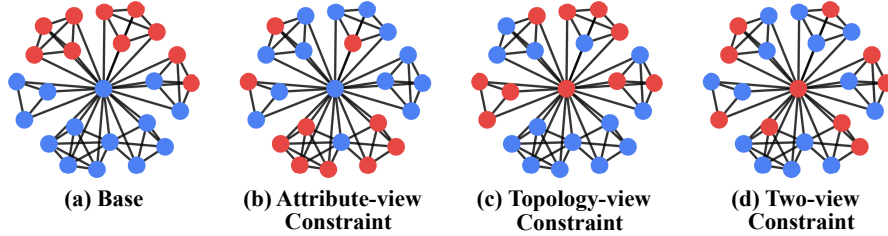


Fig. 2. Illustration of the constraint mechanism. Visualization of node selection results with and without (**Base**) constraints. The reserved nodes are highlighted in **Red**.

connected. Consequently, node drop pooling models may be stuck in significant local structures, thus selecting redundant nodes and ignoring significant nodes from other substructures. To empirically validate our analysis, we test the SAG-Pool model [14] on a real-world dataset (i.e., IMDB-BINARY). The experimental conditions are identical to those for the graph classification problem, which is described in detail in Section 4.1. In this example study, 40% of the nodes that are selected as significant nodes by the first pooling layer are marked in red. As shown in Fig. 2 (a), SAGPool (i.e., **Base**) is more likely to select nodes concentrated in the same area, confirming our hypothesis. As a result, existing node drop pooling methods may overlook critical information in other parts of a graph, causing loss of critical information and the lower performance in the graph classification task.

To address the limitations of existing node drop pooling methods, we design a masked graph auto-encoder constrained strategy called Masked Graph Auto-encoder constrained Pooling (MGAP), which mitigates the information-loss impact associated with graph pooling. Specifically, we incorporate a graph auto-encoder layer with two decoders into graph pooling models in order to impose implicit restrictions on the pooled graphs from two perspectives. Firstly, from the node-attribute perspective, we apply GCN layers to the embeddings of pooled nodes to reconstruct the original node attributes (**Section 3.1**), aiming to prevent the pooled graph from losing excessive critical attribute information. Secondly, from the network-topology perspective, we adopt a feedforward neural network to rebuild the node degree (**Section 3.2**), which enables the node drop pooling models to reserve more important nodes with regard to the topology aspect. As illustrated in Fig. 2 (b) and (c), the selected nodes are distributed across different substructures or cover the fundamental nodes in the graph, demonstrating that the proposed attribute- and topology-view constraints enable models to reserve significant nodes from the attribute and topology aspects (retaining more attribute and topological information), respectively. Additionally, Fig. 2 (d) illustrates the effect of combining constraints from the two views. Subsequently, we describe how to integrate MGAP with the present architecture for node drop

pooling (**Section 3.3**). To further demonstrate the practical efficiency of our MGAP, we provide an in-depth analysis of time efficiency (**Section 3.4**).

Furthermore, we extensively examine MGAP across three backbone models and eleven benchmark datasets, which vary in content domains and dataset sizes. The experimental results demonstrate that MGAP generally and consistently improves the performance of the current node drop pooling models (e.g., TopKPool, SAGPool, and GSAPool). Our contributions are summarized as follows.

- We propose MGAP to alleviate the information-loss effect in graph pooling from the perspectives of attribute space and topology space.
- We demonstrate that MGAP is a plug-and-play and easy-to-compute module, which can be combined with node drop pooling methods to enhance their performance in the graph classification task. Furthermore, MGAP maintains controllable time and memory complexities.
- We conduct extensive experiments using three typical node drop pooling methods with and without MGAP in the graph classification task across eleven real-world datasets. The experimental results comprehensively demonstrate the effectiveness of MGAP.

2 Preliminaries and Related Works

2.1 Notations

Let $G = (\mathcal{V}, \mathcal{E})$ denote a graph with the node set \mathcal{V} and edge set \mathcal{E} . The node attributes are denoted by $\mathbf{X} \in \mathbb{R}^{n \times d}$, where n is the number of nodes and d is the dimension of node attributes. The graph topology is represented by an adjacency matrix $\mathbf{A} \in \{0, 1\}^{n \times n}$.

2.2 Problem Statement

Definition 1 (Graph Classification). *The task of graph classification is to learn a mapping function f :*

$$f : \mathcal{G} \rightarrow \mathcal{Y}, \quad (1)$$

where $\mathcal{G} = \{G_1, G_2, \dots, G_t\}$ is the set of input graphs, $\mathcal{Y} = \{y_1, y_2, \dots, y_t\}$ is the set of labels associated with the graphs, and t is the number of graphs.

2.3 Graph Convolutional Networks

Recently, numerous studies have been conducted based on Graph Convolutional Networks, which generalize convolutional operation in graph data. The basic idea behind such methods as Graph Convolutional Network (GCN) [13], GraphSAGE [10], Graph Attention Network (GAT) [33], and Graph Isomorphism Network (GIN) [36], is to update the embedding of each node with messages from

its neighbor nodes. Formally, the above message-passing mechanism in the l -th layer can be formalized as follows:

$$\mathbf{h}_v^{(l+1)} = \text{COM}^{(l)} \left(\left\{ \mathbf{h}_v^{(l)}, \text{AGG}^{(l)} \left(\left\{ \mathbf{h}_{v'}^{(l)} : v' \in \mathcal{N}_v \right\} \right) \right\} \right), \quad (2)$$

where \mathcal{N}_v is the set of neighbors of node v . $\mathbf{h}_v^{(l)} \in \mathbb{R}^c$ is the representation vector for node v in the l -layer, where c is the dimension of the node embeddings. AGG and COM refer to the aggregation and combination functions, respectively. The methods mentioned above have achieved excellent performance in the node classification and link prediction tasks. However, an additional pooling operation is required to obtain a representation of the entire graph for downstream graph-level tasks (for example, the graph classification).

2.4 Graph Pooling

Definition 2 (Graph Pooling). *Let a graph pooling operator be defined as any function POOL that maps a graph G to a new pooled graph $G' = (\mathcal{V}', \mathcal{E}')$:*

$$G' = \text{POOL}(G), \quad (3)$$

where $|\mathcal{V}'| < |\mathcal{V}|$ and $|\mathcal{V}|$ is the number of nodes⁵. The generic goal of graph pooling is to reduce the number of nodes in a graph while preserving its semantic information.

Graph pooling, which plays a crucial role in representing the entire graph, could be roughly divided into global pooling and hierarchical pooling. Global pooling performs global sum/average/max-pooling [5] or more sophisticated operations [40,36] on all node attributes to produce graph-level representations, which disregard the topology of graphs. Contrarily, hierarchical pooling models are later proposed considering the graph topology, which could be classified into node clustering pooling and node drop pooling. **1) Node clustering pooling** considers the graph pooling problem as a node clustering problem to map the nodes into a set of clusters [37,3,20,38], which is limited by time and memory complexities caused by the dense soft-assignment matrix computation. Additionally, as discussed in previous studies [23,9], clustering-enforcing regularization that enforces clustering is typically ineffective. **2) Node drop pooling** exploits learnable scoring functions to eliminate nodes with relatively lower significance scores [7,14,16,39,8,41]. While the node drop pooling is more economical and suitable to large-scale networks than node clustering pooling, it suffers from an inevitable information loss. For a detailed description of graph pooling, please refer to the recent review [19].

⁵ In some very specific cases, there exists $|\mathcal{V}'| \geq |\mathcal{V}|$, causing the graph to be up scaled by pooling.

2.5 Graph Auto-Encoder

Recent years have seen a surge of interest in studying the framework of auto-encoder for graph embedding. The non-probabilistic graph auto-encoder model (GAE) [12] consists of a GCN encoder, integrating the graph topology and node attributes, and a nonlinear inner product decoder, reconstructing the adjacency matrix. Formally, the auto-encoder can be summarized as:

$$\hat{\mathbf{A}} = \sigma(\mathbf{Z}\mathbf{Z}^\top), \text{ with } \mathbf{Z} = \text{GCN}(\mathbf{X}, \mathbf{A}), \quad (4)$$

where $\hat{\mathbf{A}} \in \{0, 1\}^{n \times n}$ is the reconstructed adjacency matrix, and σ is the logistic sigmoid function. $\mathbf{Z} \in \mathbb{R}^{n \times c}$ is the node embedding matrix, where c is the dimension of the node embeddings.

Instead of reconstructing the graph topology in GAE, some methods attempt to design a decoder to reconstruct the node attributes [25,15] or both the topology and attributes [31]. However, these methods are unsuitable for large-scale graphs. Therefore, some methods [27,28] have introduced general frameworks to scale GAE to large-scale graphs. Unlike the above methods, which perform auto-encoder in the Euclidean space, some recent studies [24,21] have attempted to encode and decode graphs in the hyperbolic space. Furthermore, Salha et al. [30] extended the GAE frameworks to address link prediction in directed graphs using gravity-inspired decoder scheme. Due to the space limitation, some other GAE methods, such as linear GAE [29], permutation-invariant GAE [35], and adaptive GAE [18], are not presented here in detail. Compared with the above studies, our manuscript heuristically exploits auto-encoder to constrain the pooled graphs of node drop pooling methods.

3 MGAP: Masked Graph Auto-Encoder Constrained Pooling

The whole structure of the proposed MGAP is illustrated in Fig. 3, which contains two parts: the constraint from the perspective of attribute (**Section 3.1**) and the constraint from the perspective of topology (**Section 3.2**). Additionally, we discuss how to integrate the present node drop pooling methods with MGAP (**Section 3.3**). Finally, we conduct an extensive investigation of complexity (**Section 3.4**).

3.1 Constraint in Attribute Space

The node attribute in a graph is essential for graph representation learning because each node attribute depicts partial characteristics of the graph. However, as illustrated in Fig. 1, current node drop pooling methods tend to discard a large amount of node attribute information, which may cause the decreased performance in the graph classification task. Therefore, we suggest compensating for this information loss through the use of an auto-encoder system. Specifically,

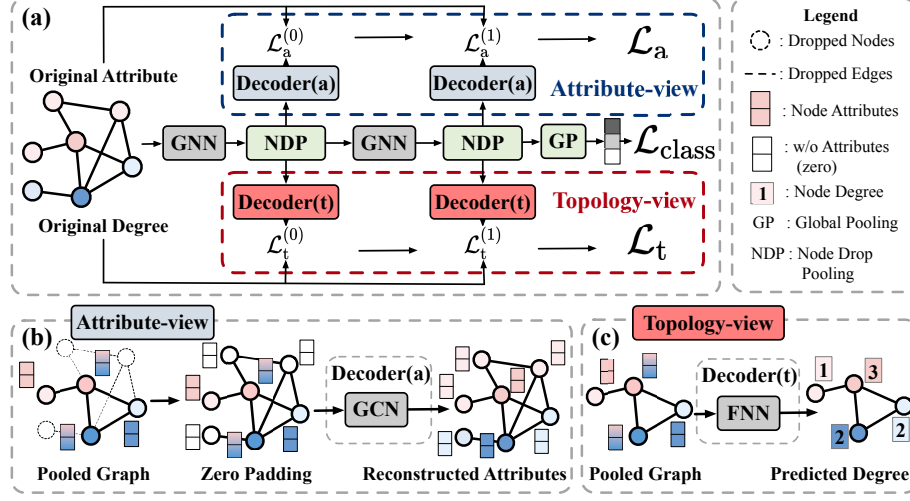


Fig. 3. The illustration of the proposed MGAP, which includes two parts: the attribute-view constrained module and the topology-view constrained module.

given the hidden representations of partially pooled nodes, we attempt to reconstruct the original node attributes of all nodes in the graph. In the proposed approach, there are three **Components**: **(C1)** the node drop pooling encoder, **(C2)** the designed decoder in attribute space, and **(C3)** the reconstruction target, the details of which are introduced as below.

(C1) Node Drop Pooling Encoder. Instead of following the standard GAE approaches [12], which adopt the well-established GNN models shown in Eq. (2) as an encoder, we employ one-layer GCN with graph pooling (GNN and NDP in Fig. 3 (a)) as the encoder. The objective of the encoder is to learn the embeddings of each node and to select which partial nodes to discard (mask). The embeddings of these masked nodes will not be observed by the decoder. As shown in Fig. 3 (a), the encoder first performs message propagation on the graphs to generate node embeddings using Eq. (2), and then generates coarsened graphs using node drop pooling methods. We used the SAGPool model [14] to describe the process of node drop pooling encoder, which consists of three disjoint parts:

1) *Generating Scores.* SAGPool predicts the significance scores for each node by using graph convolution as follows:

$$\mathbf{S}^{(l)} = \text{GCN}(\mathbf{Z}^{(l)}, \mathbf{A}^{(l)}) \in \mathbb{R}^{n^{(l)} \times 1}, \quad (5)$$

where $\mathbf{S}^{(l)} \in \mathbb{R}^{n^{(l)} \times 1}$ is the score matrix for the nodes, $\mathbf{A}^{(l)} \in \{0, 1\}^{n^{(l)} \times n^{(l)}}$ is the adjacency matrix of the coarsened graph in the layer l , and $n^{(l)}$ is the number of reserved nodes in the coarsened graph.

2) *Selecting Nodes.* Subsequently, SAGPool selects the nodes with the top- k significance scores as follows:

$$\text{idx}^{(l)} = \text{TOP}_k(\mathbf{S}^{(l)}), \quad (6)$$

where TOP_k ranks values and returns the indices of the largest k values in $\mathbf{S}^{(l)}$, and $\text{idx}^{(l)}$ indicates the reserved node indices for new graphs.

3) *Coarsening Graphs.* With the selected nodes, a new graph coarsened from the original one is obtained by learning new attribute and adjacency matrices:

$$\begin{aligned} \mathbf{Z}^{(l+1)} &= \mathbf{Z}_{\text{idx}^{(l)}}^{(l)} \odot \mathbf{S}_{\text{idx}^{(l)}}^{(l)} \in \mathbb{R}^{n^{(l+1)} \times 1}, \\ \mathbf{A}^{(l+1)} &= \mathbf{A}_{(\text{idx}^{(l)}, \text{idx}^{(l)})}^{(l)} \in \{0, 1\}^{n^{(l+1)} \times n^{(l+1)}}, \end{aligned} \quad (7)$$

where \cdot_{idx} is an indexing operation, $\mathbf{Z}_{\text{idx}^{(l)}}^{(l)}$ is the row-wise indexed embedding matrix, \odot is the broadcast elementwise product, and $\mathbf{A}_{(\text{idx}^{(l)}, \text{idx}^{(l)})}^{(l)}$ is the row-wise and column-wise indexed adjacency matrix. $\mathbf{Z}^{(l+1)}$ and $\mathbf{A}^{(l+1)}$ are the new attribute and corresponding adjacency matrices, respectively.

(C2) Decoder in Attribute Space. Unlike traditional GAE, MGAP, with the embeddings of pooled nodes, aims to recover the original attributes of graphs containing pooled nodes and masked (dropped) nodes. In particular, following [7], we first initialize an empty attribute matrix $\hat{\mathbf{X}}_0 \in \mathbb{R}^{n \times c}$ for the new graph. Subsequently, we insert the pooled node embeddings $\mathbf{Z}^{(l)} \in \mathbb{R}^{n^{(l)} \times c}$ into $\hat{\mathbf{X}}_0$ to obtain a new embedding matrix $\hat{\mathbf{X}}^{(l)} \in \mathbb{R}^{n \times c}$ (the zero padding operation in Fig. 3 (b)). The other row vectors (embeddings of the dropped nodes) remain zero. Then, we adopt graph convolution as the decoder, as introduced in Eq. (2), on the new node embedding matrix $\hat{\mathbf{X}}$ and the original adjacency matrix $\mathbf{A}^{(0)} = \mathbf{A} \in \mathbb{R}^{n \times n}$ to reconstruct the node attributes:

$$\psi_a(\hat{\mathbf{X}}^{(l)}) = \text{GCN}(\hat{\mathbf{X}}^{(l)}, \mathbf{A}^{(0)}) \in \mathbb{R}^{n \times d}. \quad (8)$$

(C3) Reconstruction Target. The constraint in attribute space aims to improve the power of pooling methods to preserve node information; that is, the learned embeddings of reserved nodes can recover the original attributes of the whole graph. Therefore, we directly measure the Euclidean distance between the reconstructed attribute matrix $\psi_a(\hat{\mathbf{X}}^{(l)})$ and the original input attribute matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, and consider it as the loss function, which is formalized as follows:

$$\mathcal{L}_a^{(l)} = \left\| \mathbf{X} - \psi_a(\hat{\mathbf{X}}^{(l)}) \right\|_F^2, \quad (9)$$

where $\mathcal{L}_a^{(l)}$ is the attribute-view constrained loss in the layer l , which enables pooling models to reserve additional important nodes from the perspective of node attributes, and $\|\cdot\|_F$ is the Frobenius norm.

3.2 Constraint in Topology Space

In addition to the attribute information discussed previously (Section 3.1), topological information in a graph is essential in graph representation learning. Therefore, it is logical and critical to ensure that the pooled nodes can reassemble the network topology. We propose to reconstruct the node degree, motivated by NWR-GAE [32], with the goal of capturing topological information. Specifically, our solution consists of three Components: **(C1)** the node drop pooling encoder, which is the same as the encoder in attribute-view constraint and will not be given any further details here, **(C2)** the designed decoder in topology space, and **(C3)** the reconstruction target, all of which are described in detail below.

(C2) Decoder in Topology Space. The decoders in existing graph auto-encoders are designed to drive the embeddings of the linked nodes similar, which appears away from our motivation that enables models to capture the topological information. Therefore, we suggest reconstructing the node degree, which is a typical graph topological feature that reflects the receptive field of a node. Specifically, given the embedding of the pooled nodes, we adopt an FNN layer with an activation function $\text{ReLU}(\cdot)$, which makes the predicted value non-negative, to reconstruct the node degree in the l -th layer:

$$\psi_t(\mathbf{Z}^{(l)}) = \text{ReLU}\left(\text{FNN}\left(\mathbf{Z}^{(l)}\right)\right) \in \mathbb{R}^{n^{(l)} \times 1}. \quad (10)$$

(C3) Reconstruction Target. We measure the Euclidean distance between the truth degree $\mathbf{D}^{(l)} \in \mathbb{R}^{n^{(l)} \times 1}$ and the predict degree $\psi_t(\mathbf{Z}^{(l)})$, which is formalized as follows:

$$\mathcal{L}_t^{(l)} = \left\| \mathbf{D}^{(l)} - \psi_t(\mathbf{Z}^{(l)}) \right\|_F^2, \quad (11)$$

where $\mathcal{L}_t^{(l)}$ is the topology-view constrained loss in the layer l . With this loss, a fraction of important nodes from the perspective of the topology can be reserved.

3.3 Node Drop Pooling Framework with MGAP

Fig. 3 (a) illustrates in detail how to apply MGAP to the node drop pooling framework. Concretely, we view a GCN layer followed by a node drop pooling layer, such as TopKPool or SAGPool, as a pooling function unit and name it GCN-Pool layer for convenience. A GCN-Pool layer takes a graph as input and outputs a pooled graph that is represented by an embedding matrix and a new adjacency matrix. The two decoders ψ_a (Decoder (a) in Fig. 3) and ψ_t (Decoder (t) in Fig. 3) are trained to simultaneously reconstruct the original node attributes and network topology, which generates two losses, \mathcal{L}_a and \mathcal{L}_t . The pooled graph is then fed into the next GCN-Pool layer and, simultaneously, a readout module, in which the node embeddings are added up as the graph embedding in this layer. Finally, the graph embeddings in all layers are added up to the final graph representation, that is taken as the input of a Multi-layer Perceptron (MLP) classifier for predicting the label of the original graph. Classification error is defined by the cross-entropy loss $\mathcal{L}_{\text{class}}$.

By combining the classification loss $\mathcal{L}_{\text{class}}$ and two constrained losses in Eq. (9) and (11), we obtained the total loss:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{class}} + \lambda_a \mathcal{L}_a + \lambda_t \mathcal{L}_t, \quad (12)$$

where λ_a and λ_t are the trade-off weight parameters, and \mathcal{L}_a and \mathcal{L}_t are the average attribute- and topology-view constrained losses of all layers, respectively. Notably, the graph classification task is performed by GCN-Pool layer in the proposed framework, and the decoders were only used for constraining the pooled nodes and their embeddings in GCN-Pool. Thus, the decoders are used only in the training phase. After obtaining the trained model, we apply GCN-Pool to perform graph classification in the test set without decoders.

3.4 Complexity Analysis

Our proposed MGAP is highly efficient because the major operations involved in it are only GCN and FNN, as shown in Fig. 3 (b) and (c), respectively. Theoretically, the time complexity of GCN layer is $\mathcal{O}(L\|\mathbf{A}\|_0 d + Lnd^2)$, where L is the number of layers, n is the number of nodes, and $\|\mathbf{A}\|_0$ is the number of nonzeros in the adjacency matrix \mathbf{A} . The time complexity of FNN layer is $\mathcal{O}(1)$. The time complexity for calculating \mathcal{L}_a by Eq. 9 is $\mathcal{O}(nd)$ and calculating \mathcal{L}_t by Eq. 11 is $\mathcal{O}(n')$, where n' is the number of the pooled nodes. Thus, the total time complexity of the proposed method is $\mathcal{O}(L\|\mathbf{A}\|_0 d + Lnd^2)$, which is on par with the neighborhood aggregation operation in node drop pooling methods.

4 Experiments

In this section, we study the effectiveness of MGAP for graph classification. Specifically, we would like to answer the following questions:

- Q1.** How often and how much does MGAP improve the performance of the base node drop pooling methods? (Section 4.2)
- Q2.** Does each component of MGAP contribute to the improvements in performance? (Section 4.3)
- Q3.** How much extra computation time and memory does MGAP incur? (Section 4.4)
- Q4.** How would the parameters affect the performance? (Section 4.5)

4.1 Experimental Settings

Datasets. To answer **Q1**, we use 11 publicly available and well-known benchmark datasets, including bioinformatics datasets (D&D, PROTEINS, and ENZYMES), molecule datasets (NCI1, NCI109, PTC-MR, MUTAG, MUTAGENICITY, and FRANKENSTEIN), and social network datasets (IMDB-BINARY and IMDB-MULTI). The above 11 real-world datasets vary in content domains and dataset sizes, and the dataset statistics are summarized in Table 1.

Backbone Models. We select three representative node drop pooling models as the backbone: **TopKPool** [7]. This method selects the top k nodes based

Table 1. Statistics and properties of benchmark datasets (TUDatasets).

	Datasets	# Graphs	# Classes	Avg. # Nodes	Avg. # Edges
Bioinformatics	D&D	1,178	2	284.32	715.66
	PROTEINS	1,113	2	39.06	72.82
	ENZYMES	600	6	32.63	124.20
Molecules	NCI1	4,110	2	29.87	32.30
	NCI109	4,127	2	29.68	32.13
	PTC-MR	344	2	14.30	14.69
	MUTAG	188	2	17.93	19.79
	MUTAGENICITY	4,337	2	30.32	30.77
	FRANKENSTEIN	4,337	2	16.90	17.88
Social Networks	IMDB-BINARY	1,000	2	19.77	96.53
	IMDB-MULTI	1,500	3	13.00	65.94

TUDatasets: <https://chrsmrrs.github.io/datasets/docs/datasets/>

on the scores generated by a learnable function that only considers node attributes. **SAGPool** [14]. This method selects the important nodes with higher scores that are generated by a graph convolution layer, which involves node attributes and network topology. Particularly, this method has two variants: 1) *SAGPool (G)* is a global node drop pooling method that drops unimportant nodes at one time at the end of the architecture. 2) *SAGPool (H)* is a hierarchical node drop pooling method that sequentially drops unimportant nodes with multiple graph convolution layers. We use *SAGPool (H)* in this study. **GSAPool** [39]. This method predicts scores from two perspectives: 1) using an MLP layer to capture the significant node attributes and 2) using a GNN layer to capture the significant network topology. Subsequently, the model linearly combines the two scores mentioned above.

Implementation Details. For a fair comparison, we adopt the same settings on all datasets and models. Specifically, we evaluate the model performance with a 10-fold cross validation setting, and the dataset split is based on the conventionally used training/test splits[36,1]. Each convolution layer consists of 128 hidden neurons, and the pooling ratio in each pooling layer is set as 0.5, i.e., removing 50% of nodes per graph after a pooling operation. We employ Adam [11] to optimize the parameters with learning rate as $5e^{-4}$ and weight decay as $1e^{-4}$, and adopt early stopping to control the training epochs based on validation loss with patience set as 50. We then report the average performance on the test sets, by performing overall experiments 100 times with different seeds from 42 to 51.

Hyper-parameter tuning. As described in Section 3.3, two hyper-parameters λ_a and λ_t are used in our MGAP, which serve as trade-off weights in the loss function. We utilize a grid search to tune the above two hyper-parameters with a search space $\{1, 1e^{-1}, 1e^{-2}\}$.

Environments. 1) Software. All models are implemented with Python 3.7, PyTorch 1.9.0 or above (which further requires CUDA 10.2 or above), and

Table 2. MGAP performance across three backbone models and 11 datasets in the graph classification task. The reported results are mean and standard deviation over 100 different runs.

	Molecules					
	NCI1	NCI109	MUTAG	PTC-MR	MUTAGE.	FRANK.
SAGPool	71.71 \pm 0.75	70.70 \pm 0.95	72.56 \pm 3.09	56.41 \pm 1.63	74.27 \pm 1.04	58.74 \pm 0.61
+ MGAP	73.02 \pm 1.00	71.95 \pm 0.63	73.72 \pm 1.88	58.80 \pm 2.29	74.99 \pm 1.22	59.06 \pm 0.81
Gain	1.83% \uparrow	1.76% \uparrow	1.60% \uparrow	4.23% \uparrow	0.97% \uparrow	0.54% \uparrow
TopKPool	71.90 \pm 1.22	70.69 \pm 1.00	71.83 \pm 1.66	57.15 \pm 3.14	75.10 \pm 0.94	58.84 \pm 0.80
+ MGAP	72.83 \pm 1.24	72.35 \pm 1.03	73.06 \pm 2.77	58.35 \pm 2.70	76.46 \pm 1.05	58.96 \pm 0.53
Gain	1.29% \uparrow	2.35% \uparrow	1.71% \uparrow	2.10% \uparrow	1.81% \uparrow	0.0% \uparrow
GSAPool	73.70 \pm 0.89	71.83 \pm 1.65	72.56 \pm 2.41	56.10 \pm 1.83	76.65 \pm 1.12	59.11 \pm 0.69
+ MGAP	75.36 \pm 1.68	74.10 \pm 1.95	72.44 \pm 3.42	57.59 \pm 2.81	77.94 \pm 1.03	59.57 \pm 0.32
Gain	2.25% \uparrow	3.16% \uparrow	0.16% \downarrow	2.66% \uparrow	1.68% \uparrow	0.78% \uparrow

	Bioinformatics			Social Networks		Average
	D&D	PROT.	ENZYM.	IMDB-B	IMDB-M	
SAGPool	74.21 \pm 1.23	72.65 \pm 1.26	47.42 \pm 1.54	70.71 \pm 1.36	48.43 \pm 0.81	65.25
+ MGAP	76.20 \pm 0.73	74.53 \pm 1.04	48.93 \pm 3.69	72.88 \pm 1.22	49.71 \pm 0.80	66.70
Gain	2.68% \uparrow	2.58% \uparrow	3.18% \uparrow	3.07% \uparrow	2.64% \uparrow	2.22% \uparrow
TopKPool	73.71 \pm 1.04	72.81 \pm 0.74	46.02 \pm 3.53	70.96 \pm 1.15	48.97 \pm 0.60	65.27
+MGAP	75.97 \pm 0.96	73.95 \pm 1.23	49.58 \pm 2.01	72.05 \pm 0.50	49.52 \pm 0.79	66.64
Gain	3.06% \uparrow	1.56% \uparrow	7.73% \uparrow	1.53% \uparrow	1.12% \uparrow	2.10% \uparrow
GSAPool	74.19 \pm 1.32	73.20 \pm 1.11	49.08 \pm 2.02	71.06 \pm 1.15	49.03 \pm 0.50	66.00
+ MGAP	76.24 \pm 1.03	73.49 \pm 1.39	55.05 \pm 2.86	72.34 \pm 1.26	49.68 \pm 0.61	67.59
Gain	2.76% \uparrow	0.40% \uparrow	12.16% \uparrow	1.80% \uparrow	1.32% \uparrow	2.40% \uparrow

PyTorch-Geometric 1.7.3 or above. **2) Hardware.** Each experiment was run on a single GPU (NVIDIA V100 with a 16 GB memory size), and the experiments were run on the server at any given time⁶.

4.2 Overall results

To answer **Q1**, we conduct extensive experiments for graph classification on 11 datasets using three backbone models. The accuracy results of all models summarized in Table 2 are averaged over **100 runs** with random weight initializations (10 different seeds through the 10-fold cross validation). We highlight the best performance **in bold** per backbone model and dataset. In Table 2, we report the improvement achieved by MGAP on each backbone model and each dataset. We obtain the following findings. **1)** Evidently, MGAP consistently improves the

⁶ The source code is available at <https://github.com/liucuo/mgap>.

Table 3. Ablation study results. **Bold:** the best performance per backbone model and dataset. Underline: the second best performance per backbone model and dataset.

	PTC-MR			IMDB-BINARY		
	SAGPool	TopKPool	GSAPool	SAGPool	TopKPool	GSAPool
Base	56.41 \pm 1.6	57.15 \pm 3.1	56.10 \pm 1.8	70.71 \pm 1.4	70.96 \pm 1.2	71.06 \pm 1.2
MGAP	58.80\pm2.2	58.35\pm2.7	57.59\pm2.8	72.88\pm1.2	72.05\pm0.5	72.34\pm1.3
w/o attr-const	58.44 \pm 1.9	57.21 \pm 3.5	56.21 \pm 3.1	72.43 \pm 0.9	71.87 \pm 0.9	71.89 \pm 0.9
w/o topo-const	58.35 \pm 2.6	57.76 \pm 1.6	56.56 \pm 2.0	71.72 \pm 1.1	71.29 \pm 0.85	71.50 \pm 0.9

accuracy of all three node drop pooling models on all datasets in most cases, sometimes by large margins. **2)** Specifically, MGAP achieves improvements over three node drop pooling models (averaged across datasets): 2.22% (SAGPool), 2.10% (TopKPool), and 2.40% (GSAPool). **3)** MGAP obtains more significant enhancement on bioinformatics datasets and increases the accuracy by up to 12.16%. Intuitively, this may be because the information loss, caused by the condensation of selected nodes into the local structure, makes a greater impact on bioinformatics datasets. In summary, the above results indicate that MGAP is a general framework for improving the performance of base node drop pooling methods.

4.3 Ablation Study

To answer **Q2**, we conduct ablation studies on the dataset PTC-MR (social domain) and IMDB-BINARY (biochemical domain) using the SAGPool model. For convenience, we name the methods without attribute-view and topology-view constraints as **w/o attr-const** and **w/o topo-const**, respectively. Note that except the selected component, the rest remain the same as the complete model. From Table 3, we obtain that all variants with some components removed exhibit clear performance drops compared to the complete model, indicating that each component contributes to the improvements. Furthermore, MGAP without the topology-view constraint performs poorly on the IMDB-BINARY dataset, thereby demonstrating the significance of the proposed topology-view constraint for datasets in social domain, where network topology plays an important role.

4.4 Efficiency Analysis

To answer **Q3**, we compare the time and memory efficiency of MGAP with that of three backbone models. **1) Time Efficiency.** Fig. 4 (a) illustrates the average per-epoch training time on all 11 datasets. We fix the training epochs to 10 with 10 different random seeds. It is observed that the additional time consumption keeps relatively low. **2) Memory Efficiency.** The experimental settings are the same as those in measuring the time efficiency. Fig. 4 (b) shows that our MGAP is efficient in terms of memory. The above results confirm that our MGAP is practically efficient.

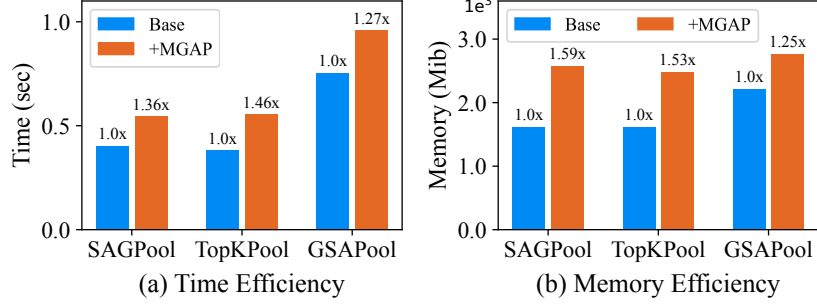


Fig. 4. Memory and time efficiency of MGAP compared with three backbone models. (a) The reported values are the average per-epoch training time on all 11 datasets. (b) The reported values are the average GPU memory usage on all 11 datasets.

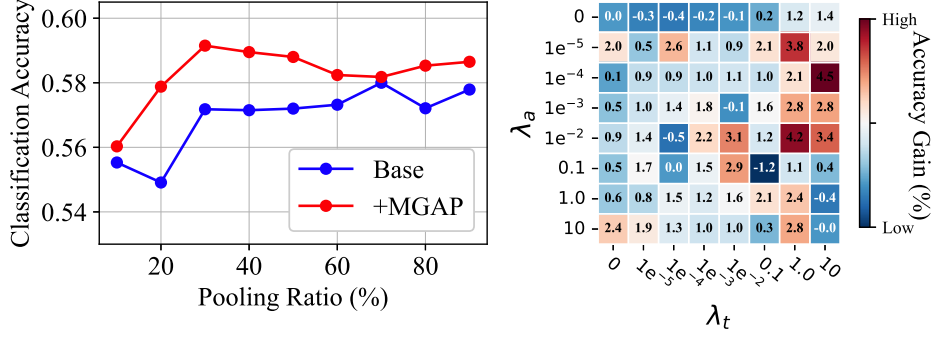


Fig. 5. Parameter analysis on the PTC-MR dataset. *Left.* Model performance varying with the pooling ratio. *Right.* Parameter sensitivity of trade-off weights λ_a and λ_t .

4.5 Parameter Analysis

To answer **Q4**, we investigate the sensitivity of the parameters of two types on the PTC-MR dataset using the SAGPool model. **1) Inherent Parameter Sensitivity.** We study how the graph pooling ratio would affect the graph classification performance. As shown in the left part of Fig. 5, SAGPool equipped with MGAP (+MGAP) performs better in all cases, suggesting that the proposed method enable node drop pooling methods to select the nodes that are essential for graph-level representation learning regardless of the pooling ratio. **2) Introduced Parameter Sensitivity.** We investigate the effects of two new parameters, λ_a and λ_t , which serve as the trade-off weights in the loss function. In this parameter sensitivity study, both parameters are searched within the range of $\{10, 1, 1e^{-1}, 1e^{-2}, 1e^{-3}, 1e^{-4}, 1e^{-5}, 0\}$. Note that the search space is only $\{1, 1e^{-1}, 1e^{-2}\}$ in the graph classification experiments. As shown in the

right part of Fig. 5, the method performs best when λ_a is $1e^{-4}$ and λ_t is 10, demonstrating the importance of combining attribute and topology constraints of the pooled graphs. The results of the above two experiments further validate the robustness and effectiveness of the proposed MGAP.

5 Conclusion and Future Work

Conclusion. In this study, we empirically verify the information-loss problem of current node drop pooling models and propose MGAP, a novel plug-in and easy-to-compute module, to solve this problem from the perspectives of attribute space and topology space. Through extensive experiments, we demonstrate that MGAP generally improves common node drop pooling methods across various benchmark datasets in the graph classification task.

Future Work. For future directions, **1)** choose various formulas of GNNs, such as attention mechanism [33], as a decoder, in addition to GCN, for reconstructing the original attributes of nodes. **2)** Consider other topological features, such as triangle count, local clustering score, eigenvector centrality, and betweenness, in addition to node degrees. **3)** Further design other evaluation criteria for topological information loss, such as some criteria studied in graph coarsening algorithms [4]. **4)** Explore the effects of MGAP on other tasks such as graph reconstruction, graph compression, and node classification, and further design more reasonable constraints for these tasks.

Acknowledgements

This work was supported in part by the Natural Science Foundation of China (Nos. 61976162, 82174230, 62002090), Artificial Intelligence Innovation Project of Wuhan Science and Technology Bureau (No.2022010702040070), Science and Technology Major Project of Hubei Province (Next Generation AI Technologies) (No. 2019AEA170), and Joint Fund for Translational Medicine and Interdisciplinary Research of Zhongnan Hospital of Wuhan University (No. ZNJC202016).

References

1. Baek, J., Kang, M., Hwang, S.J.: Accurate learning of graph representations with graph multiset pooling. In: ICLR (2021)
2. Bai, L., Jiao, Y., Cui, L., Hancock, E.R.: Learning aligned-spatial graph convolutional networks for graph classification. In: ECML-PKDD. pp. 464–482 (2019)
3. Bianchi, F.M., Grattarola, D., Alippi, C.: Spectral clustering with graph neural networks for graph pooling. In: ICML. vol. 119, pp. 874–883 (2020)
4. Cai, C., Wang, D., Wang, Y.: Graph coarsening with neural networks. In: ICLR (2021)
5. Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., Adams, R.P.: Convolutional networks on graphs for learning molecular fingerprints. In: NeurIPS. p. 2224–2232 (2015)

6. Errica, F., Podda, M., Bacciu, D., Micheli, A.: A fair comparison of graph neural networks for graph classification. In: ICLR (2020)
7. Gao, H., Ji, S.: Graph u-nets. In: ICML. pp. 2083–2092 (2019)
8. Gao, X., Dai, W., Li, C., Xiong, H., Frossard, P.: ipool-information-based pooling in hierarchical graph neural networks. IEEE TNNLS (2021)
9. Grattarola, D., Zambon, D., Bianchi, F.M., Alippi, C.: Understanding pooling in graph neural networks. arXiv:2110.05292 (2021)
10. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NeurIPS. vol. 30 (2017)
11. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv:1412.6980 (2014)
12. Kipf, T.N., Welling, M.: Variational graph auto-encoders. NeurIPS Workshop on Bayesian Deep Learning (2016)
13. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017)
14. Lee, J., Lee, I., Kang, J.: Self-attention graph pooling. In: ICML. pp. 3734–3743 (2019)
15. Li, J., Li, J., Liu, Y., Yu, J., Li, Y., Cheng, H.: Deconvolutional networks on graph data. NeurIPS (2021)
16. Li, M., Chen, S., Zhang, Y., Tsang, I.: Graph cross networks with vertex infomax pooling. In: NeurIPS. vol. 33, pp. 14093–14105 (2020)
17. Li, X., Zhou, Y., Dvornek, N., Zhang, M., Gao, S., Zhuang, J., Scheinost, D., Staib, L.H., Ventola, P., Duncan, J.S.: Braingnn: Interpretable brain graph neural network for fmri analysis. Medical Image Analysis (2021)
18. Li, X., Zhang, H., Zhang, R.: Adaptive graph auto-encoder for general data clustering. IEEE TPAMI (2021)
19. Liu, C., Zhan, Y., Li, C., Du, B., Wu, J., Hu, W., Liu, T., Tao, D.: Graph pooling for graph neural networks: Progress, challenges, and opportunities. arXiv:2204.07321 (2022)
20. Ma, Y., Wang, S., Aggarwal, C.C., Tang, J.: Graph convolutional networks with eigenpooling. In: SIGKDD. pp. 723–731 (2019)
21. Mathieu, E., Le Lan, C., Maddison, C.J., Tomioka, R., Teh, Y.W.: Continuous hierarchical representations with poincaré variational auto-encoders. NeurIPS (2019)
22. McPherson, M., Smith-Lovin, L., Cook, J.M.: Birds of a feather: Homophily in social networks. Ann. Rev. sociology pp. 415–444 (2001)
23. Mesquita, D., Souza, A., Kaski, S.: Rethinking pooling in graph neural networks. In: NeurIPS. vol. 33, pp. 2220–2231 (2020)
24. Park, J., Cho, J., Chang, H.J., Choi, J.Y.: Unsupervised hyperbolic representation learning via message passing auto-encoders. In: CVPR. pp. 5516–5526 (2021)
25. Park, J., Lee, M., Chang, H.J., Lee, K., Choi, J.Y.: Symmetric graph convolutional autoencoder for unsupervised graph representation learning. In: ICCV. pp. 6519–6528 (2019)
26. Rhee, S., Seo, S., Kim, S.: Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. In: IJCAI. pp. 3527–3534 (2018)
27. Salha, G., Hennequin, R., Remy, J.B., Moussallam, M., Vazirgiannis, M.: Fastgae: Scalable graph autoencoders with stochastic subgraph decoding. Neural Networks **142**, 1–19 (2021)
28. Salha, G., Hennequin, R., Tran, V.A., Vazirgiannis, M.: A degeneracy framework for scalable graph autoencoders. In: IJCAI. pp. 3353–3359 (2019)

29. Salha, G., Hennequin, R., Vazirgiannis, M.: Simple and effective graph autoencoders with one-hop linear models. In: ECML-PKDD. pp. 319–334 (2020)
30. Salha, G., Limnios, S., Hennequin, R., Tran, V.A., Vazirgiannis, M.: Gravity-inspired graph autoencoders for directed link prediction. In: CIKM. pp. 589–598 (2019)
31. Sun, D., Li, D., Ding, Z., Zhang, X., Tang, J.: Dual-decoder graph autoencoder for unsupervised graph representation learning. *Knowledge-Based Systems* **234**, 107564 (2021)
32. Tang, M., Li, P., Yang, C.: Graph auto-encoder via neighborhood wasserstein reconstruction. In: ICLR (2022)
33. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: ICLR (2018)
34. Wang, Z., Ji, S.: Second-order pooling for graph neural networks. *IEEE TPAMI* (2020)
35. Winter, R., Noé, F., Clevert, D.A.: Permutation-invariant variational autoencoder for graph-level representation learning. *NeurIPS* **34** (2021)
36. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: ICLR (2019)
37. Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. In: *NeurIPS*. p. 4805–4815 (2018)
38. Yuan, H., Ji, S.: Structpool: Structured graph pooling via conditional random fields. In: ICLR (2020)
39. Zhang, L., Wang, X., Li, H., Zhu, G., Shen, P., Li, P., Lu, X., Shah, S.A.A., Bennamoun, M.: Structure-feature based graph self-adaptive pooling. In: *WWW*. pp. 3098–3104 (2020)
40. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: *AAAI* (2018)
41. Zhang, Z., Bu, J., Ester, M., Zhang, J., Li, Z., Yao, C., Huifen, D., Yu, Z., Wang, C.: Hierarchical multi-view graph pooling with structure learning. *IEEE TKDE* (2021)