Hierarchical Adaptive Pooling by Capturing High-order Dependency for Graph Representation Learning

Ning Liu, Songlei Jian†, Dongsheng Li†, Yiming Zhang, Zhiquan Lai and Hongzuo Xu

Abstract—Graph neural networks (GNN) have been proven to be mature enough for handling graph-structured data on node-level graph representation learning tasks. However, the graph pooling technique for learning expressive graph-level representation is critical yet still challenging. Existing pooling methods either struggle to capture the local substructure or fail to effectively utilize high-order dependency, thus diminishing the expression capability. In this paper we propose HAP, a hierarchical graph-level representation learning framework, which is adaptively sensitive to graph structures, i.e., HAP clusters local substructures incorporating with high-order dependencies. HAP utilizes a novel cross-level attention mechanism MOA to naturally focus more on close neighborhood while effectively capture higher-order dependency that may contain crucial information. It also learns a global graph content GCont that extracts the graph pattern properties to make the pre- and post-coarsening graph content maintain stable, thus providing global guidance in graph coarsening. This novel innovation also facilitates generalization across graphs with the same form of features. Extensive experiments on ten datasets show that HAP significantly outperforms twelve popular graph pooling methods on graph classification task with an maximum accuracy improvement of 20.18%, and exceeds the performance of state-of-the-art graph matching and graph similarity learning algorithms by over 3.42% and 16%.

Index Terms—Graph Representation Learning, Graph Pooling, Attention Mechanism, Hierarchical Manner.

1 Introduction

Although data that can be represented as grid structure on Euclidean domains, such as images [1], video [2], speech [3], and texts [4], has closely connections with daily life, there is another major category of Non-Euclidean data, namely graph, which is constructed by irregularly-arranged nodes and the connection-indicated edges. Examples include social networks [5], citation networks [6], road networks [7] and bioinformatics [8]. Different from Euclidean data, convolution and pooling operation in Convolutional Neural Networks (CNNs) cannot be directly applied to Non-Euclidean graph-structured data due to the irregularity and nondeterminacy of the neighborhood for the central node. Consequently, it is important to learn a sufficiently expressive representation for graph-structured data in a reasonable way.

1.1 Motivation

A great deal of research on Graph Neural Networks (GNNs) has emerged to generalize the great success of convolution in CNNs to graph-structured data. In GNNs, convolution operation is evolved into neighborhood message aggregation of the central node along edges, thus capturing both node features and graph structural information. Following this principle, various GNNs have been proposed, such as GCN [9], [10] and GAT [11]. All of them have achieved significant prosperity for graph representation learning tasks, especially for node-level representation based tasks, including node classification [11] and link prediction [12]. However, as for graph-level representation learning tasks, such as graph classification [13], [14], graph matching [15] and graph similarity learning [16], convolution operation alone is deficient. It is nontrivial to potentially empower GNNs to produce discriminative graph-level representations with the help of pooling operation.

To remedy this problem, a few researchers have tried to further generalize the pooling mechanism from CNNs to GNNs for graph-level representation learning. Accordingly, it is natural to raise the question: what are the basic criteria for a high-quality pooling method in GNNs? Actually, different from the pooling operation in CNNs for reducing the number of computational parameters and preserving the invariance, the basic idea of graph pooling techniques is a node feature aggregator throughout the entire graph, analogous to the neighborhood aggregator in graph convolution. Therefore, a good graph pooling method should encourage graphs with approximate topology and similar node features to have resemblant representations to some extent. As a result, the major challenge for a high-quality pooling mechanism is to define a method which both effectively maintains pivotal node features and explicitly captures important structural information. To address this challenge, previous works have proposed graph pooling architectures in three ways.

First, universal maximum/average pooling meth-

- †*Corresponding author.*
- *Ning Liu, Songlei Jian, Dongsheng Li, Yiming Zhang, Zhiquan Lai and Hongzuo Xu are with the College of Computer, National University of Defense Technology, China.*
  E-mail: liuning17a, jiansonglei, dsli, ymzhang, zqlai, xuhongzuo13 @nudt.edu.cn

Manuscript received December 19, 2020; revised April 26, 2021.

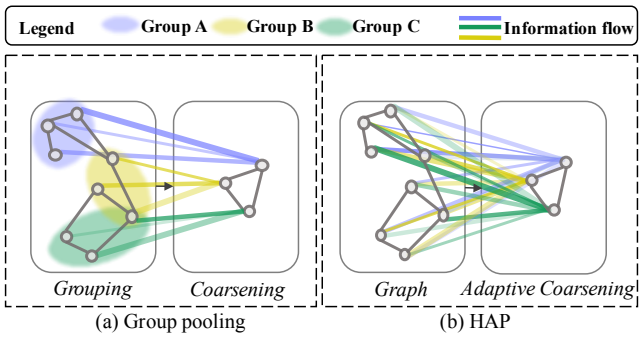


Fig. 1: Differences of receptive field during clustering between group pooling methods and HAP.

ods [16], [17] are intuitively extended to graph models by simple element-wise max- or mean-downsampling through all node features. But such methods have been proved to ignore feature multiplicities, as well as completely miss the structural information [18]. Graphs with different corresponding structures may get the same representation.

Second, *Top-K* methods [14], [19], [20], [21], [22] sort graph nodes in a consistent order with scores on behalf of the importance, only K nodes with the highest scores are selected to form the pooled graph. As a result, sets of nodes except for the top K ones are definitely discarded, which may involve important features. Meanwhile, very few *Top-K* methods utilize the local substructure information for scoring stage, thus resulting a strong possibility of the selected nodes to be isolated, which may influence the information propagation in subsequent GNN layers.

Third, advanced graph pooling methods, such as DiffPool [13] and ASAP [23], learn graph representations in a hierarchical grouping manner for capturing more comprehensive local substructures widespread in graphs. They usually group nodes into smaller clusters with multiple levels and utilize the final level as the graph representation. However, grouping operation is usually executed on a fixed 1-hop neighborhood, thus forcing the information to flow from a certain neighborhood to the specific coarsened cluster and neglecting the higher-order dependency among nodes that may hold significant information, as shown in Fig. 1(a).

1.2 Contributions

To the best of our knowledge, no existing graph pooling methods adaptively handle the graph local substructures and high-order dependency, while capturing node features. And there is a general lack of systematical consideration to the generalization of graph-level representations affected by pooling methods. To bridge the above gap, we propose a novel hierarchical graph pooling framework called **H**ierarchical **A**daptive **P**ooing (**HAP**). The main contributations can be summarized as follows.

- We intruduce **HAP**, a supervised hierarchical pooling framework. HAP is capable of preserving node feature information with adaptive graph structure sensitivity to both local substructures and high-order dependency. We provide a more comprehensible framework by offering exhaustive theoretical analysis for computational complexity, permutation invariance and the design validity of HAP.
- We propose **master-orthogonal attention (MOA)**, a novel cross-level attention mechanism specifically designed for hierarchical graph pooling. MOA can be leveraged to capture cross-level interactions under the guidance of graph pattern properties in a more efficient and effective way. MOA also acts as a soft substructure extractor. Attention weights for nodes in the receptive field of a possible local cluster are much higher than those beyond the receptive field. This ensures the local-substructure sensitivity and introduces high-order node features to it.
- We design **GCont**, an auto-learned global graph content playing a significant role in MOA. The key innovation is that it incorporates high-level global pattern properties into pooling method, making MOA sensitive to the latent graph characteristics and produce a more adequate graph-level representation without interference of artificial factors. It is relatively stable during hierarchical pooling and flexible enough to be learned heuristically. GCont also guarantees the generalization ability across graphs with the same form of features.

Extensive experiments demonstrate that (1) HAP significantly outperforms twelve graph pooling methods on seven real-world datasets for graph classification task with a maximum accuracy improvement of 20.18%; (2) HAP sharply outperforms the state-of-the-art GMN [15], which is specifically designed for graph matching task, by boosting the accuracy up to 3.42%; (3) HAP also achieves a maximum accuracy gain of 16% comparing with conventional approximate GED algorithms; (4) The graph coarsening module in HAP dramatically enhances the expression ability of existing graph pooling architecture for graph-level tasks; (5) HAP achieves good generalization ability across graphs with the same form of features; and (6) HAP provides meaningful visualization of graph-level representations.

2 Related Work

2.1 Supervised Pooling

Supervised pooling methods can be divided into flat pooling and hierarchical pooling according to whether the graph-level representation is aggregated in a flat or hierarchical way with a view to local substructures. Further, flat pooling methods also cover two families: universal pooling and *TopK* pooling, depending on the number of nodes participated in the final aggregation.

2.1.1 Flat Universal Pooling

Flat universal pooling methods take all the nodes into consideration. Earlier works directly learn from CNNs to use mean- or max-pooling method to extract features. Subsequently, Xu et al. [18] find that sum-pooling is much more powerful because no matter mean or max aggregator ignores the multiformity of features, thus struggling in distinguishing graphs with nodes that have repeating features. Some other works rely on content-based attention operation. In Gated Graph Neural Networks (GG-NNs) [17],

the graph-level output is defined by a soft attention mechanism for each node to decide which is more relevant to the graph-level task. Message Passing Neural Networks (MPNNs) [24] further utilizes Set2Set [19] method to take the order of nodes into consideration and find the importance of each node to the graph-level representation through time-consuming iterative soft-attention. In SimGNN [16] and UGRAPHEMB [25], a graph content is defined as the average of node features and the attention is executed between nodes and it. Obviously, such man-made design makes the final graph-level representation infinitely close to the output of mean-pooling method, which is an inefficient method mentioned above. Besides, to make the graph-level representation do not depend on the number of nodes anymore, SAGE [26] utilizes the self-attentive mechanism to learn node importance and encode it into a unified graph representation matrix, which is size invariant. Further, SOPool [27] not only produces a size-invariant graph representation matrix but also empowers it by collecting second-order statistics.

2.1.2 Flat Top-K pooling

Flat Top-K pooling methods score the nodes according to their importance. Nodes with *k*-largest scores are preserved to form the new graph. SortPooling [20] method refers to the graph label method WL [28], it regards the output node features of each GCN layer as the continuous WL colors and sorts the nodes according to the last GCN layer's colors. AttPool [29] calculates the scores using a global soft-attention mechanism. Furthermore, a local attention method accesses node degree information, which contributes to keeping a balance between the importance and the dispersion. iPool [30] utilizes neighborhood information gain criterion and only preserves nodes with high information. gPool [14] develops new ideas that use the projection of node features to a trainable projection vector as a node score. SAGPool [21] considers both node features and graph topology during pooling by taking GCN to calculate attention scores. However, important information that existed in the abandoned nodes may be ignored, which should be explicitly captured in graph pooling. Moreover, no structural relationships among nodes are acquired during pooling, thus may lead to the unconnectedness of the selected nodes. To overcome the unconnectedness limitation, HGP-SL [31] applies a structure learning mechanism with sparse attention after the selection of TopK nodes. However, its node selection process suffers from quadratic computational complexity w.r.t the number of nodes, making HGP-SL formidable to be generalized to relatively larger graphs. TAP [32] reduces the time complexity of node selection by computing similarity scores only between every pair of connected nodes, which also explicitly encodes the topology information.

2.1.3 Hierarchical Group Pooling

Due to the fact that local substructures are present in real-world graphs, hierarchical group pooling methods come into being. DiffPool [13] is the first differentiable group pooling approach that learns a dense assignment matrix to group a 1-hop neighborhood of nodes into clusters in each hierarchical layer. Subsequently, to address the sparsity concerns in DiffPool, ASAP is proposed, which combines both TopK and group methods. Clusters are generated by aggregating *h*-hop neighbors of each central node to leverage the graph structure, then only top-scoring clusters are maintained. However, ASAP [23] still cannot guarantee connectivity between the selected clusters. Actually, Mesquita et al. in [33] indicate that a successful graph pooling method should not be restricted to nearby nodes. Thus we propose that high-order structural dependency also contributes to the construction of a good pooling region. Recently, HaarPooling [34] introduces a new node grouping strategy by using the Haar basis. Nevertheless, the construction of Haar basis totally neglects the graph structural information. EigenPooling [35] preserves graph substructure during node clustering based on the graph Fourier transform, but the eigen decomposition of the graph Laplacian is time-consuming.

2.2 Unsupervised Pooling

Loss functions of the aforementioned graph pooling methods are usually task-based supervised except for DiffPool that exploits a link prediction loss and enforces nearby nodes to be pooled together. Recently, there is a growing interest in unsupervised graph pooling by minimizing the objective related to graph structure characteristic or borrowed from graph theory. StructPool [36] employs conditional random fields (CRFs) to capture high-order structural relationships by minimizing the Gibbs energy. MinCutPool [37] continuously relaxes the normalized min-CUT problem in graph theory and optimizes the cluster assignments by minimizing this objective. UGRAPHEMB [25] utilizes well-accepted and domain-agnostic graph proximity metrics to provide extra graph-graph proximity guidance during learning. These novel ideas offer the possibility of breaking a logjam of current graph pooling research.

Something also worth mentioning is that there is a common challenge for no matter universal pooling, *Top-K* pooling or group pooling, i.e., the element-wise aggregation, score ranking and cluster assignment learning processes are merely executed on a single fixed graph, lacking the inductive capability for entirely new graphs.

3 PRELIMINARIES

3.1 Problem Statement

A graph is represented as $G = (\mathcal{V}, \mathcal{E}, \mathcal{X})$, where $\mathcal{V} = \{\mathbf{v}_i\}_{i=\{1,\cdots,N\}}$ denotes the set of nodes, $\mathbf{e}_{ij} = (\mathbf{v}_i, \mathbf{v}_j) \in \mathcal{E}$ is the edge link between node $\mathbf{v}_i$ and $\mathbf{v}_j$, and $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$ consists the node labels (no node labels are provided in some cases). For a graph G with $N = |\mathcal{V}|$ nodes and $|\mathcal{E}|$ edges, $\mathbf{A} \in \mathbb{R}^{N\times N}$ represents the weighted adjacency matrix and $\mathbf{D} \in \mathbb{R}^{N\times N}$ is a diagonal matrix that diagonal elements stand for the degree of nodes. $\mathbf{H} \in \mathbb{R}^{N\times F}$ denotes the node feature matrix and $\mathbf{h}_G \in \mathbb{R}^{F_G}$ is the graph-level embedding. A label Y may also be attached to the graph G. Detailed notations are summarized in TABLE 1. Given a graph dataset, the graph pooling task aims to learn a mapping $f : \mathbf{H} \rightarrow \mathbf{h}_G$ from a node feature matrix to a single graph representation.

TABLE 1: Notations

Notations	Definitions or Descriptions
G, G'	the input/coarsened graph
$\mathcal{V}$, $\mathcal{E}$, $\mathcal{X}$	the node/edge/node label set of G
N, N'	the number of nodes of G/G'
$\mathbf{A}$, $\mathbf{A}'$	the adjacent matrix of G/G'
$\mathbf{H}$, $\mathbf{H}'$	the node feature matrix of G/G'
$\mathbf{h}_G$	the graph-level embedding of G
K	the number of graph coarsening modules
$\mathbf{C}$	the auto-learned global graph content
$\mathbf{C}_{(i,\cdot)}$	row in C refers to a node of the source graph G
$\mathbf{C}_{(\cdot,j)}$	column in C refer to a cluster of the target graph G'
$\mathbf{M}$	the MOA matrix
F	the dimension of input node feature
F'	the dimension of the output node feature
F_G	the dimension of the graph level embedding
Y	the label of graph G

3.2 Graph Neural Networks

Given node features $\mathbf{H} = \{h_1, \cdots, h_N\}$ and graph structure $\mathbf{A}$, modern GNNs usually learn useful node representations in an neighborhood aggregation fashion following general "message-passing" architecture. The forward process comprises two phases, each of which iteratively runs for $\mathbf{L}$ time steps. The message passing phase aggregates information along edges of the central node from its neighbors. Then the combination phase updates the representation of the central node based on the message:

$$\mathbf{MESSAGE}^{(l)}_{\mathcal{V}\to i} = \mathbf{AGGREGATE}^{(l)}\left(h_j^{(l-1)} : (j,i) \in \mathcal{E}\right) \tag{1}$$

$$h_i^{(l)} = \mathbf{COMBINE}^{(l)}\left(h_i^{(l-1)}, \mathbf{MESSAGE}^{(l)}_{\mathcal{V}\to i}\right) \tag{2}$$

where $h_i^{(l)}$ is the embedding of node i at the l-th iteration that is initialized as $h_i^{(0)} = h_i$, and $h_j^{(l-1)}$ is the node feature vector of node i's neighbor depending on the adjacency matrix.

There are multiple selectable implementations of $\mathbf{AGGREGATE}^{(l)}(\cdot)$ and $\mathbf{COMBINE}^{(l)}(\cdot)$ adapted successfully to different GNN models. Actually, our HAP pooling framework can be consolidated into any GNN models following the implementation of Equation 1 and Equation 2. After $\mathbf{L}$ times iteration, the representation of the central node captures the features and structural information within its $\mathbf{L}$-hop neighborhood.

3.3 Graph Attention

Graph attention mechanism executed between a query q and a key k allows for allocating diverse alignment scores α_{ij} to different parts of the input, making the model focus on the most relevant portion. Existing graph attention mechanism can be divided into *node-level attention* and *master-level attention* according to the attention scope. Specifically, node-level attention covers both self-attention and cross-attention.

Hard-Self-Attention (HSA) [11] chooses both q and k from the node features of the single input graph to find the node dependency on itself:

$$\alpha_{ij} = softmax\left(\sigma\left(u^\top\left[\mathbf{W}h_i \| \mathbf{W}h_j\right]\right)\right), \{h_i\}_{i\in\mathcal{V}_1}, \{h_j\}_{j\in\mathcal{V}_1} \tag{3}$$

where $u^\top$ and $\mathbf{W}$ are trainable parameters, and $[\cdot\|\cdot]$ is a concatenation operation.

Soft-Self-Attention (SSA) [17] decides which nodes are relevant to the current graph-level task, so that q is defined as node feature but no specific key k is provided:

$$\alpha_{ij} = softmax\left(\sigma\left(u^\top\left(\mathbf{W}h_j\right)\right)\right), \{h_j\}_{j\in\mathcal{V}_1} \tag{4}$$

Cross-Attention (CA) [15] captures the differences between graphs by doing comparisons across the pair of graphs through choosing q and k from the node features of pairwise input, thus fusing information from both graphs:

$$\alpha_{ij} = softmax\left(\sigma\left(u^\top\left[\mathbf{W}h_i \| \mathbf{W}h_j\right]\right)\right), \{h_i\}_{i\in\mathcal{V}_1}, \{h_j\}_{j\in\mathcal{V}_2} \tag{5}$$

Master-Attention (MA) [16], [23], [25] concentrates on the interaction between nodes and the master they belong to, so that q and k denote node feature and master feature separately. The master function is generally defined as sum or max operation of the constituent nodes:

$$\alpha_{ij} = softmax\left(\sigma\left(u^\top\left[\mathbf{W}\cdot\mathbf{Master}\left(h_j\right)\|\mathbf{W}h_j\right]\right)\right), \{h_j\}_{j\in\mathcal{V}_1} \tag{6}$$

$$\mathbf{Master} = \sum_{v_j\in c_i}\left(h_j\right) \tag{7}$$

4 The Proposed Method: HAP

In this section, we present HAP, a hierarchical graph pooling framework for graph-level tasks. Its key idea is the graph coarsening module supported by novel graph pattern property extracting technique GCont and cross-level attention mechanism MOA complementing and reinforcing each other, which not only prompts the GNN model to be sensitive to both local substructures and high-order dependency, but also empowers it with stronger generalization ability. Below, we discuss the components of HAP in details.

4.1 Hierarchical Framework

Figure 2 illustrates the overall architecture of the HAP. Given single, pairwise or triplet input graphs for differentiated graph-level tasks, HAP extracts the node features and graph structure information for an end-to-end training. The process can be decomposed into six main steps:

- **Input Construction** A triplet generator is necessary for graph similarity learning task.
- **Node & Cluster Embedding** Subsequently, inputs are transferred into a node & cluster embedding module to learn a low-dimensional node vector representation for each node or coarsened cluster.
- **Graph Coarsening-I** Then, a learnable GCont defines a coarsening preparation step for each graph by extracting global pattern properties.
- **Graph Coarsening-II** Furthermore, the MOA mechanism derived from the GCont is utilized to obtain an attention assignment.
- **Graph Coarsening-III** Afterwards, a cluster formation function $\Omega : G \in \mathbb{R}^{N\times N} \to G' \in \mathbb{R}^{N'\times N'}$ is learned to compute the cluster representation after one coarsening.

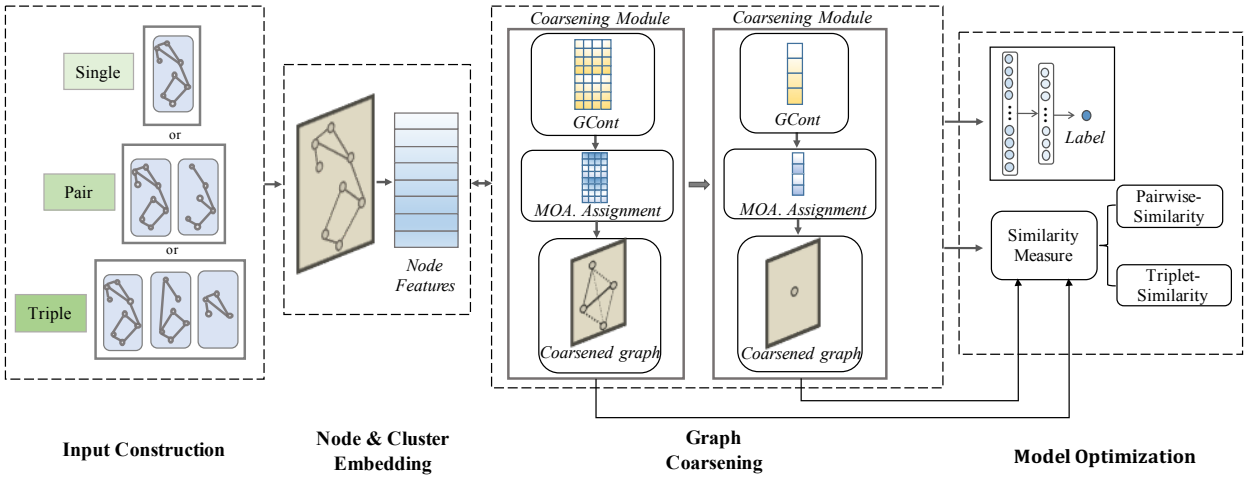


Fig. 2: The overview framework of HAP which conducts graph coarsening with GCont and MOA mechanism, and compares graphs with the hierarchical similarity measure.

- **Model Optimization** Executes the loop between Step-2 and Step-5 until reaching a satisfied graph scale. HAP then calculates corresponding task loss with hierarchical graph representation to constantly optimize all the weight parameters.

4.2 Input Construction

For graph similarity learning task, training and testing data in the form of triplet is essential to learn the relative similarity among graphs. Based on the best of our knowledge, there is no ready-to-use graph dataset in triplet form. To bridge the above gap, we propose a triplet generator in this subsection.

Given a dataset with single graphs, we denote it as $\mathcal{S}$, the similarity between every two graphs G_i and G_j can be measured under a graph-graph proximity metric $f(\cdot)$, such as Graph Edit Distance (GED). The smaller the GED, the more similar the pair. Then the pairwise ground-truth proximity is denoted as follows:

$$\mathcal{P}_{GED} \leftarrow \left\{ g_{ij} \middle| g_{ij} = f(G_i, G_j), \forall i, j \in |\mathcal{S}| \right\} \tag{8}$$

Afterwards, we conduct triplets by fixing the first position with one graph and randomly choose two disparate graphs to fill the rest two positions:

$$\mathcal{T} \leftarrow \left\{ \gamma \middle| \gamma = \langle G_i, G_j, G_k \rangle, \forall i, j, k \in |\mathcal{S}|, j \neq k \right\} \tag{9}$$

Synchronously, the ground truth triplet proximity is generated as follows, in which a positive number for the element r_{ijk} means that graph G_i is much similar to graph G_k and a negative number means that graph G_i is much similar to graph G_j:

$$\mathcal{T}_{GED} \leftarrow \left\{ r_{ijk} \middle| r_{ijk} = g_{ij} - g_{ik}, \forall i, j, k \in |\mathcal{S}|, j \neq k \right\} \tag{10}$$

4.3 Node & Cluster Embedding

There is a demand for node or cluster embedding to extract node or cluster features before going to the next graph coarsening module. In this paper, we choose to employ a two-layer GAT [11] or GCN [9] as basic components since they are all well capable of capturing the local structure information of a node. Actually, any mainstream GNNs can also be integrated into the HAP framework. And please note that the number of GAT or GCN layers depends on the real application graph data.

Take GAT for example, for the k-th layer in GAT, it takes graph G's adjacent matrix $\mathbf{A}_k$ and the hidden representation matrix $\mathbf{H}_k$ as input, then formulates the $\mathbf{AGGREGATE}^{(\mathbf{k})}(\cdot)$ phase in a weighted-attention-based operator:

$$\mathbf{H}_{k+1} = \sigma\left(\mathbf{A}_k \mathbf{O}_{att} \mathbf{H}_k \mathbf{W}_k\right) \tag{11}$$

where $\sigma(\cdot)$ is the non-linear activation function such as *ReLU* or *Sigmoid*, $\mathbf{O}_{att}$ is a trainable global attention assignment among all nodes, and $\mathbf{A}_k \mathbf{O}_{att}$ picks one-hop neighborhood attention. $\mathbf{W}_k$ is a trainable weight matrix.

Specifically, different from classical GAT or GCN where graph scale is stable throughout the whole training, HAP scales down nodes into clusters in the graph coarsening module before transferring them to the next node & cluster embedding layer. As a result, $\mathbf{A}_k$ changes with the action of graph coarsening (cf. Eq. 17).

4.4 Graph Coarsening

We achieve graph coarsening through graph global pattern property extracting technique GCont and cross-level attention mechanism MOA. We show the graph coarsening module architecture in Fig. 3 and elaborate the details in this subsection.

4.4.1 *Attention Preparation using GCont*

Given node features for the source graph, the task of coarsening process is to learn the cluster assignment matrix through attention mechanism. However, one important thing ignored by all the group pooling methods is that

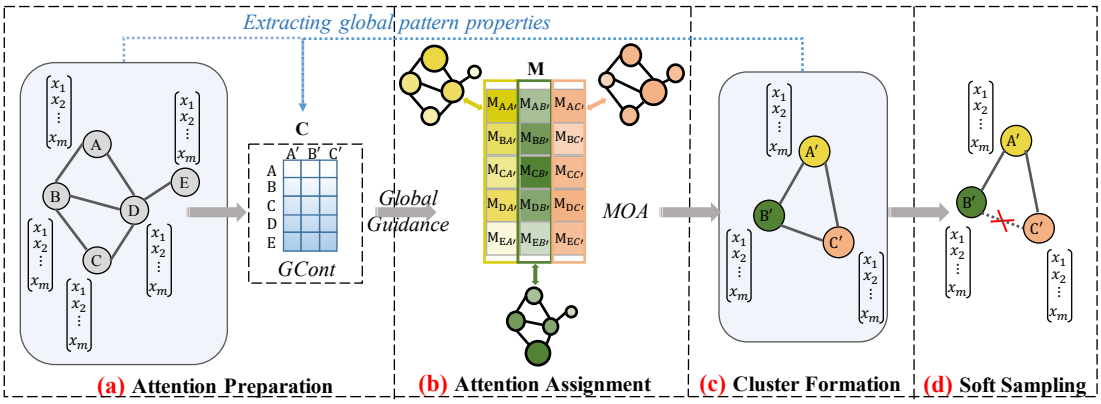


Fig. 3: An illustration of the graph coarsening module. The whole procedure contains four steps: (a) attention preparation by constructing GCont in which rows and columns stand for nodes from the source graph and the coarsened graph separately; (b) computing cross-level attention assignment by using MOA; (c) cluster formation by aggregating information from the source graph; (d) executing soft sampling for edges in the target coarsened graph.

the pre- and post-coarsening graph content should remain stable without loss of important information. We observe that both DiffPool [13] and ASAP [23] receive no global guidance. Hence, we propose *GCont*, an auto-learned global graph content sustaining the coarsening process.

As an initial step, we propose using one learnable linear transformation, parametrized by the weight matrix $\mathbf{T} \in \mathbb{R}^{F \times N'}$ to generate GCont. The simple linear transformer also combines scalability with the ability to deal with relatively larger graphs. The global graph content is converted from the node feature matrix $\mathbf{H}$ as:

$$\mathbf{C} = \mathbf{H}\mathbf{T} \tag{12}$$

where $\mathbf{C} \in \mathbb{R}^{N \times N'}$ is the automatically learned global graph content matrix in which each row $\mathbf{C}_{(i,\cdot)} \in \mathbb{R}^{N'}$ is equivalent to a node of the source graph G and each column $\mathbf{C}_{(\cdot,j)} \in \mathbb{R}^{N}$ is corresponding to a cluster node of the target coarsened graph G'.

The GCont bridges the gaps between the source graph and the target graph and maintains the consistency. On one hand, the elements in $\mathbf{C}$ reflect the interaction between nodes from source graph and clusters from coarsened graph. On the other hand, they contain the graph pattern properties cohered before and after coarsening, thus facilitating generalization across graphs with the same form of features.

4.4.2 Attention Assignment using MOA

HAP intends to achieve graph downsampling through a cross-level attention-based aggregator for information interaction between the source graph and the coarsened target graph utilizing global graph property guidance. However, we observe that both HSA and SSA described in Sec. 3.3 only focus on one single graph while CA does not utilize any global information. Although MA introduces master information into the attention process, it is highly affected by the manmade master function. To that end, we propose a new variant of attention mechanism called *Master-Orthogonal-Attention (MOA)*.

Computation of Attention Assignment: The input of MOA mechanism is a well-learned representation matrix $\mathbf{H} = \{h_1, h_2, \cdots, h_N\}$, $h_i \in \mathbb{R}^F$, where N is the number of nodes of the source graph G, and F is the feature dimension for each node. Then the graph coarsening module produces a new coarsened graph representation matrix $\mathbf{H}' = \{h'_1, h'_2, \cdots, h_{N'}\}$, $h'_i \in \mathbb{R}^F$ as its output, where N' is the number of clusters of the coarsened graph. Each cluster will then be regarded as an individual node. Meanwhile, adjacent matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ will also be updated to $\mathbf{A}' \in \mathbb{R}^{N' \times N'}$. Please note that the number of graph coarsening modules and the coarsened graph size N' are determined by the real application graph data. In our experiment, we employ two coarsening modules and we evaluate it in the experiment.

After having obtained the global graph content matrix, we can employ an orthogonal[1] cross-level attention mechanism between nodes of the source graph and clusters of the target coarsened graph. The attention matrix $\mathbf{M} \in \mathbb{R}^{N \times N'}$ is formed with elements as follows:

$$\mathbf{M}_{ij} = \sigma\left(\mathbf{a}^\top \left[\mathbf{C}_{(i,\cdot)} \| \mathbf{C}_{(\cdot,j)}\right]\right) \tag{13}$$

where σ is the *LeakyReLU* nonlinearity, $[\cdot\|\cdot]$ is a concatenation operation with relaxed dimension of $\mathbf{C}_{(\cdot,j)}$ from $\mathbb{R}^N$ to $\mathbb{R}^{N'}$, and $\mathbf{a}^\top \in \mathbb{R}^{2N'}$ is the trainable shared attentional parameter with relaxed dimension $2N'$. The reason for the relaxation will be given below.

$\mathbf{M}$, which is equivalent to a cross-level aggregator, offers a fully-connected information channel between the source-graph nodes and the target coarsened-graph clusters, with each element $\mathbf{M}_{ij}$ indicating the importance of node i's feature to cluster j. The local substructure is preserved by attention mechanism while the high-order dependency is also captured through the fully-connected information channel, thus strengthening feature reservations. We normalize it for better evaluation:

$$\mathbf{M} = \frac{exp\left(\mathbf{M}_{ij}\right)}{\sum_{\mathbf{k} \in N'} exp\left(\mathbf{M}_{ik}\right)}. \tag{14}$$

MOA mechanism synthesizes both self-attention and cross-attention with master-attention. On one hand, the proposed MOA mechanism calculates the attention coefficients based on the GCont alone, so we can sort it into self-attention mechanism. On the other hand, the attention is

1. The terminology "orthogonal" here means rows and columns of a 2D matrix, which is different from the meaning of orthogonal vectors in a mathematical sense.

predicted between the source graph and the target coarsened graph, so we may also classify it as cross-attention mechanism.

Relaxation of Attentional Parameter: In traditional graph attention scheme [11], attention coefficients are calculated as follows:

$$\mathbf{M}_{ij} = \sigma\left(\mathbf{a}^{\top}\left[\mathbf{W}h_i \| \mathbf{W}h_j\right]\right) \tag{15}$$

where σ is the *LeakyReLU* nonlinearity, $\mathbf{a}^{\top} \in \mathbb{R}^{2F'}$ is the trainable shared attentional mechanism, $\mathbf{W} \in \mathbb{R}^{F' \times F}$ is a weight matrix to produce new node features from cardinality F to F', $h_i \in \mathbb{R}^F$ and $h_j \in \mathbb{R}^F$ are the input node features, and $[\cdot\|\cdot]$ is a concatenation operation.

Apparently, the trainable shared attentional parameter $\mathbf{a}^{\top} \in \mathbb{R}^{2F'}$ in the conventional graph attention mechanism is irrelevant to the node number of the input graph. However, in our MOA mechanism, the dimension of $\mathbf{C}_{(\cdot,j)} \in \mathbb{R}^N$ is related to the node number N of the inputted source graph, making the concatenation $\left[\mathbf{C}_{(i,\cdot)} \| \mathbf{C}_{(\cdot,j)}\right] \in \mathbb{R}^{N+N'}$. As a result, the trainable shared attentional mechanism would be initialized as $\mathbf{a} \in \mathbb{R}^{N+N'}$, which is sensitive to the node number N of the inputted source graph.

Manifestly, N varies with the input and it is unknown for parameter initializing stage. Thus, proper relaxation offers intriguingly good performance when standard techniques appear to suffer. We loose N to N', so that $\left[\mathbf{C}_{(i,\cdot)} \| \mathbf{C}_{(\cdot,j)}\right] \in \mathbb{R}^{2N'}$. We theoretically analyse the validity of the relaxation on the prediction outcome in Sec. 5.3.

4.4.3 Cluster Formation

The learning of the global graph content and the cross-level aggregator constitutes a concordant unity, and complement and restrict mutually. Subsequently, we generate the coarsened graph representation matrix $\mathbf{H}' \in \mathbb{R}^{N' \times F}$ and update the adjacent matrix $\mathbf{A}' \in \mathbb{R}^{N' \times N'}$:

$$\mathbf{H}' = \mathbf{Aggregate}\left(\mathbf{H}\right) = \mathbf{M}^{\top}\mathbf{H}, \tag{16}$$

$$\mathbf{A}' = \mathbf{M}^{\top}\mathbf{A}\mathbf{M}. \tag{17}$$

4.4.4 Soft Sampling

According to Lee et al. [21], handling adjacent data with a sparse matrix in GNN contributes to decreasing the computational complexity from $\mathcal{O}(|\mathcal{V}|^2)$ to $\mathcal{O}(|\mathcal{E}|)$ and also reduces space complexity. However, the adjacent matrix $\mathbf{A}'$ turns to be a dense one from a sparse assignment. That said, the structure of the source graph G is refined to a fully-connected downsampled one. Proper edge sampling will lead to saving both time and storage without a dramatic loss of accuracy. As a workaround, we adopt the Gumbel-SoftMax [38] to achieve soft sampling for neighborhood relationship, thus decreasing edge density for the sampled adjacent matrix $\widetilde{\mathbf{A}}'$:

$$\widetilde{\mathbf{A}'}_{\mathbf{ij}} = \frac{exp\left(\left(log\mathbf{A}_{ij} + \mathbf{g}_{ij}\right)/\tau\right)}{\sum_{k=1}^{N'} exp\left(\left(log\mathbf{A}_{ik} + \mathbf{g}_{ik}\right)/\tau\right)} \tag{18}$$

where $\mathbf{g} = -log\left(-log\left(\mathbf{u}\right)\right)$ and $\mathbf{u} \sim \mathbf{Uniform}\left(0,1\right)$. Here, we set the softmax temperature parameter $\tau = 0.1$ to make the adjacent matrix distribution close to one-hot. This operation reduces edge density as much as possible but preserves the connectivity of graphs.

4.5 Model Optimization

The proposed HAP supports three types of input: single graph G for graph classification, pairwise graphs (G_1, G_2) for graph matching, and triplet graphs (G_1, G_2, G_3) for graph similarity learning. All of the input graphs will be coarsened to a 1D vector at the final graph embedding layer, which can be used to compute graph similarity directly. Meanwhile, as is demonstrated in model structure, HAP alternates between node embedding and graph coarsening, thus generating different graph representation matrix $\mathbf{G}_i^k$ at graph coarsening layer $\mathbf{k}$. As a result, we also propose a hierarchical similarity measure by jointly utilizing hierarchical graph representations.

4.5.1 Prediction

For graph classification tasks with a single input graph G, the final graph representation $\mathbf{G}$ is directed fed into two fully-connected layers with a $softmax\left(\cdot\right)$ activation on the output to get the predicted label $\hat{Y}$. Then we optimize the model with a standard cross-entropy on the graph that has ground truth labels Y. The fully-connected layers and the objective function can be represented separately as follows:

$$\begin{cases} \mathbf{f}_1 = \sigma\left(\mathbf{W}_1\mathbf{G} + \mathbf{b}_1\right) \\ \mathbf{f}_2 = \sigma\left(\mathbf{W}_2\mathbf{f}_1 + \mathbf{b}_2\right) \end{cases} \tag{19}$$

$$\mathcal{L}_{single} = -\sum_{g \in \mathcal{B}} \sum_{m=1}^{c} Y_m^g log\hat{Y}_m^g \tag{20}$$

where $\mathbf{W}_i$ and $\mathbf{b}_i$ represent weights and biases in the i-th fully-connected layer respectively for $i \in \{1,2\}$. σ is the adopted *ReLU* and *Softmax* activation function for $\mathbf{f}_1$ and $\mathbf{f}_2$ separately. $\mathcal{B}$ is the training set of single graphs and c denotes the number of classes.

For graph matching tasks with pairwise input graphs (G_1, G_2), pairs are labeled with $true$ or $false$ representing similar or dissimilar respectively. We optimize the normalization function to push the model to convert graph distances to similarity scores with distribution $s \in (0,1)$:

$$s_{(G_1,G_2)}^k = exp(-scale \times d_{(G_1,G_2)}^k) \tag{21}$$

where $scale \in (-\infty, 0)$ denotes a softmax parameter sensitive to different range of distances and is determined by the real application graph data. Basically, we set it to 0.5. $d_{(G_1,G_2)}^k$ represents graph distances of graph pair (G_1, G_2) at coarsen level $k \in K$, and here we use Euclidean distance. Then the model is optimized by hierarchical cross-entropy function as follows:

$$\mathcal{L}_{pair} = -\frac{1}{K\left|\mathcal{P}\right|} \sum_{k \in K} \sum_{(G_1,G_2) \in \mathcal{P}} Y_p log\left(s_{(G_1,G_2)}^k\right) \tag{22}$$

where $\mathcal{P}$ is the training set of pairwise graphs. $Y_p \in \{0,1\}$ is the label for this pair.

For graph similarity learning tasks with triplet input graphs (G_1, G_2, G_3), hierarchical Mean Squared Error (MSE) loss function is employed as follows:

$$\mathcal{L}_{triple} = \frac{1}{K\left|\mathcal{T}\right|} \sum_{k \in K} \sum_{t \in \mathcal{T}} \left(\left(d_{(G_1,G_2)} - d_{(G_1,G_3)}\right) - \mathcal{T}_{GED}\right) \tag{23}$$

where $\mathcal{T}$ is the training set of triplet graphs, $\mathcal{T}_{GED}$ denotes ground truth triplet proximity defined by relative Graph Edit Distance (GED) at Sec. 4.2.

4.5.2 Hierarchical Prediction

As shown in Fig. 2, we adopt a hierarchical prediction strategy to further facilitate the training process and fully utilize the hierarchical intermediate features of coarsened graphs. The outputs of every coarsening process are summarized as the intermediate graph feature, which will be fed into the learning module for graph matching or graph similarity learning.

5 Theoretical Analysis

5.1 Computational Complexity Analysis

In the following, we theoretically analysis the computational complexity of the proposed HAP and show the superiority of the proposed graph coarsening module.

CLAIM 1 (Time Complexity). The time complexity of the proposed HAP with K graph coarsening modules in dowmsampling ratio r is approximately $\mathcal{O}(N^2)$, where N is the number of nodes of the original input graph.

PROOF 1. The time complexity of HAP involves three parts corresponding to the three stages of GNN-based graph-level representation learning models: (1) node embedding; (2) graph coarsening; and (3) learning. The time complexity of node embedding stage is $\mathcal{O}(NFF' + |\mathcal{E}|F')$ [11], where F' is the dimension of output node features. After that, to downsample node number in the k-th graph coarsening module, where $k \in \{1, 2, \cdots, K\}$, it requires $\mathcal{O}(r^{k-1}N \cdot r^k N)$. Let's suppose r remains constant among all the coarsening modules. Then the time complexity for all the K graph coarsening modules is $\mathcal{O}(rN^2 + r^3N^2 + \cdots + r^{2K-1}N^2)$. Due to the fact that r is less than 1, $r^3 + \cdots + r^{2K-1}$ is a couple of orders of magnitude smaller than r. So the time complexity of graph coarsening stage is roughly equivalent to $\mathcal{O}(rN^2)$. Eventually, for the learning stage, the time complexity is $\mathcal{O}(F_G^2)$, where F_G is the dimension of the graph level embedding for the input graph G. Therefore, the overall computational complexity of the proposed HAP framework is $\mathcal{O}(rN^2 + NFF' + |\mathcal{E}|F' + F_G^2) \approx \mathcal{O}(rN^2) \approx \mathcal{O}(N^2)$.

Specifically, when a proper coarsening ratio r is chosen where $rN \ll N$ (e.g., $r = 0.05$ and $N = 100$), the actual execution time of the proposed HAP will become almost linear to N.

5.2 Permutation Invariance

Graph pooling methods need to be permutation invariant since they should guarantee that the graph-level representation does not vary with the input order of node-level representations. As for the proposed graph coarsening module, we proof that it is graph permutation invariant.

DEFINITION 1 (Permutation matrix). $\mathbf{P}_n \in \{0,1\}^{n \times n}$ is a permutation matrix of size n iff $\sum_i \mathbf{P}_{i,j} = 1 \ \forall j$ and $\sum_j \mathbf{P}_{i,j} = 1 \ \forall i$.

CLAIM 2 (Permutation invariance). Let $\mathbf{P}_n \in \{0,1\}^{n \times n}$ be any permutation matrix, $G = (\mathbf{A}, \mathbf{X})$ be any undirected graph, a function $f(\mathbf{A}, \mathbf{X})$ be a pooling operation depending on graph G, graph permutation is defined as $f(\mathbf{A}, \mathbf{X}) = f\left(\mathbf{P}_n \mathbf{A} \mathbf{P}_n^\top, \mathbf{P}_n \mathbf{X}\right)$. The proposed graph coarsening module is graph permutation invariant.

PROOF 2. $\mathbf{M}$ is computed by an attention mechanism between source nodes and target coarsened clusters. Since the attention function are operated between node set and cluster set, the order of nodes or clusters has no effect to the result, we have:

$$\mathbf{M} \to \mathbf{P}_n \mathbf{M} \mathbf{P}_n^\top \tag{24}$$

Since $\mathbf{A}' = \mathbf{M}^\top \mathbf{A} \mathbf{M}$ and any permutation matrix is orthogonal, applying $\mathbf{P}_n^\top \mathbf{P}_n = \mathbf{I}$ to it, we get:

$$\mathbf{A}' \to \mathbf{P}_n \mathbf{A}' \mathbf{P}_n^\top \tag{25}$$

Since $\mathbf{X}' = \mathbf{E}^\top \mathbf{X}$, applying $\mathbf{P}_n^\top \mathbf{P}_n = \mathbf{I}$ to it, we get:

$$\mathbf{X}' \to \mathbf{P}_n \mathbf{X}' \tag{26}$$

As a result, $f(\mathbf{A}, \mathbf{X}) \to f\left(\mathbf{P}\mathbf{A}\mathbf{P}^\top, \mathbf{P}\mathbf{X}\right)$, HAP is graph invariant.

5.3 Validity of Relaxation for Attentional Parameter

In Sec. 4.4.2, we conduct a relaxation operation $\psi : \mathbf{a} \in \mathbb{R}^{N+N'} \to \mathbf{a} \in \mathbb{R}^{2N'}$ for the attentional parameter. Substantially, the relaxation is applied to the column dimension of GCont $\mathbf{C}$ during concatenation. A natural question is that whether the relaxation affects the accuracy of attention coefficients, which may directly lead to neglecting important information during cross-level aggregation. We now theoretically analyze this question.

DEFINITION 2 (LeakyReLU). LeakyReLU is a monotone increasing activation function:

$$\varphi(x) = \begin{cases} x & x \geq 0 \\ \frac{x}{a} & x < 0, \end{cases} where \ a \in (1, +\infty) \tag{27}$$

CLAIM 3. Let $N > N'$, $\mathbf{C}_i \in \mathbb{R}^{N'}$, $\mathbf{C}_j \in \mathbb{R}^N$ and $\mathbf{a} \in \mathbb{R}^{N+N'}$ be vectors before relaxation, let $\mathbf{C}'_j \in \mathbb{R}^{N'}$ and $\mathbf{a}' \in \mathbb{R}^{2N'}$ be vectors after relaxation, let $[\cdot \| \cdot]$ be concatenation operation, and *LeakyReLU* be a nonlinearity. Then $\textit{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{C}_i \| \mathbf{C}_j]\right) = \textit{LeakyReLU}\left(\mathbf{a}'^\top [\mathbf{C}_i \| \mathbf{C}'_j]\right)$.

PROOF 3. The essence of $\mathbf{a}^\top [\mathbf{C}_i \| \mathbf{C}_j]$ is a similarity comparison between vector $\mathbf{C}_i \in \mathbb{R}^{N'}$ and vector $\mathbf{C}_j \in \mathbb{R}^N$. Due to the reason that vectors with different dimensions are non-comparable, the lacking dimension needs to be padded with zero. So that:

$$\mathbf{C}_i = (c_1, \cdots, c_{N'}) \to \mathbf{C}'_i = \left(c_1, \cdots, c_{N'}, \underbrace{0, 0, \cdots, 0}_{N - N'}\right) \tag{28}$$

While do comparison between $\mathbf{C}_i$ and $\mathbf{C}'_j$, we can also pad them as follows:

$$\mathbf{C}_i = (c_1, \cdots, c_{N'}) \to \mathbf{C}'_i = \left(c_1, \cdots, c_{N'}, \underbrace{0, 0, \cdots, 0}_{N - N'}\right) \tag{29}$$

$$\mathbf{C}'_j = (c_1, \cdots, c_{N'}) \rightarrow \hat{\mathbf{C}}_j = \left(c_1, \cdots, c_{N'}, \underbrace{0, 0, \cdots, 0}_{N-N'} \right) \tag{30}$$

Hence, $\mathbf{a}^\top[\mathbf{C}_i \| \mathbf{C}_j] = \mathbf{a'}^\top[\mathbf{C}_i \| \mathbf{C}'_j]$. Based on known conditions that *LeakyReLU* is monotonically increasing, so that $\textit{LeakyReLU}\left(\mathbf{a}^\top[\mathbf{C}_i \| \mathbf{C}_j]\right) = \textit{LeakyReLU}\left(\mathbf{a'}^\top[\mathbf{C}_i \| \mathbf{C}'_j]\right)$.

As a result, the relaxation for attentional parameter has no negative effects for the attention computation and feature extraction.

6 Experiments and Evaluation

We evaluate HAP against a number of state-of-the-art methods to answer the following questions:

Q1: How does HAP compare with other baselines when evaluated with downstream tasks including graph classification, graph matching and graph similarity learning? (Sec. 6.2, Sec. 6.3, Sec. 6.4)

Q2: How dose the original HAP compare with ablated ones with graph coarsening module replaced by other state-of-the-art pooling algorithms? (Sec. 6.5.1)

Q3: How does the number of the graph coarsening modules influence the quality of graph-level representations generated by HAP? (Sec. 6.5.2)

Q4: Do key designs of HAP contribute to better generalization performance? (Sec. 6.5.3)

TABLE 2: Statistics of datasets. IMDB-B, IMDB-M, COLLAB, MUTAG, PROTEINS, PTC and HIV are for graph classification; AIDS and LINUX are for graph similarity learning; and the Synthetic data is for graph classification.

Dataset	#Graphs	#Triples	#Pairs	Max.$\mathcal{V}$	Avg.$\mathcal{V}$	#Classes
IMDB-B	1000	-	-	136	20	2
IMDB-M	1500	-	-	89	13	3
COLLAB	5000	-	-	492	75	3
MUTAG	188	-	-	28	18	2
PROTEINS	1113	-	-	620	39	2
PTC	344	-	-	109	26	2
HIV	41127	-	-	222	26	2
AIDS	-	171900	-	10	9	-
LINUX	-	409600	-	10	8	-
Synthetic data	-	-	8750	300	106	2

6.1 Experimental Setup

6.1.1 Datasets

We perform experiments on nine real-world datasets and one synthetic dataset varying with tasks. The graph statistics are summarized in TABLE 2.

For graph classification, we use seven real-world benchmarks from [39] and [40], including three social network datasets (IMDB-BINARY, IMDB-MULTI, COLLAB) and four bioinformatics datasets (MUTAG, PROTEINS, PTC, HIV). IMDB-BINARY is a movie collaboration dataset and IMDB-MULTI is a multi-class version of IMDB-BINARY. COLLAB is a scientific collaboration dataset labeled by the corresponding field. MUTAG is a nitro compounds dataset where classes indicate whether the compound has a mutagenic effect on a bacterium. PROTEINS is a protein dataset where graphs are classified as enzyme or non-enzyme. PTC is a chemical compounds dataset where classes indicate carcinogenicity for male and female rats. HIV is a molecular property prediction dataset to indicate whether a molecule inhibits HIV virus replication or not.

Evaluating graph matching task requires benchmark datasets with ground-truth labels (true for matching and false for unmatched). To the best of our knowledge, no public-available real-world dataset holds such ground-truth labels. To fill this gap, we conduct a synthetic dataset, a collection of labeled graph pair (G_1, G_2) with edge probability $p \in [0.2, 0.5]$ generated by the VF2 graph matching library [41]. Given a graph G, a positive sample is the maximum connected subgraph randomly extracted with 1 to 3 nodes less than G. And a negative sample is created by randomly adding 3 to 7 nodes with the same edge probability.

For graph similarity learning, we use two small graph datasets within 10 nodes. AIDS [42] provides antivirus chemical compounds. LINUX [43] consists of Program Dependence Graphs (PDG) generated from Linux kernel. Each graph expresses dependencies between statements in a program.

6.1.2 Baselines

We compare HAP with three kinds of baselines:

Graph pooling baseline: For comparison of total pooling, we choose GCN-concat (concatenation of GCN-based node-level representations), SumPool [44], MeanPool, MeanAttPool [16], and Set2Set [19]. For *TopK* pooling, we use SortPooling [20], AttPool [29], gPool [14] and SAGPool [21]. For group pooling, we compare with DiffPool [13] and ASAP [23]. We also conduct evaluation on an unsupervised method StructPool [36].

Graph matching baseline: We focus on the Graph Matching Network (GMN) [15] specifically designed for pairwise graph similarity learning.

Graph similarity learning baseline: There are two categories of graph similarity learning baselines. Due to the reason that the ground-truth triplet proximity for graph similarity learning task is calculated by conventional rigorous GED algorithm, the first type is referred to as conventional approximate GED algorithms for comparison, including Beam search [45], VJ [46] and Hungarian [47] algorithm. The other type includes SimGNN [16] and GMN, which are GNN-based models.

6.1.3 Parameter Settings

For the basic model structure of HAP, we set two node & cluster embedding layers before every following graph coarsening module, and a total of two coarsening modules are needed. Adma optimizer is used with initial learning rate 0.01 for graph classification datasets, 0.0015 for AIDS, 0.0001 for LINUX and synthetic data. For social network datasets IMDB and COLLAB with no informative node features, we use one-hot encoding of node degrees as initial node input. Similarly, we adopt one-hot encoding of node labels for AIDS dataset, while others are initialized identically. For graph classification and other tasks, the initial dimension is 64 and 128, respectively. All of the datasets for graph matching and similarity learning are randomly partitioned into 8:1:1 for training/validation/testing. For all

TABLE 3: Graph classification accuracy and AUC score in percent. Four horizontal lines are used to split the results of universal pooling, TopK pooling, group pooling and unsupervised pooling.

Model	IMDB-B		IMDB-M		COLLAB		MUTAG		PROTEINS		PTC		HIV	
	acc	auc	acc	auc	acc	auc	acc	auc	acc	auc	acc	auc	acc	auc
GCN-concat	62.00±4.02	60.56±3.68	49.13±4.04	-	68.19±1.09	-	84.44±7.37	82.88±9.19	72.17±5.69	70.81±3.57	69.71±8.73	67.72±6.08	71.25±1.68	71.19±1.87
SumPool	78.00±3.71	77.90±3.74	52.87±1.79	-	73.26±5.78	-	91.11±6.19	92.01±6.39	78.38±4.65	77.21±4.31	73.82±4.25	72.36±5.04	71.79±2.15	71.46±2.33
MeanPool	72.24±4.35	72.94±2.59	48.11±2.86	-	76.72±2.94	-	85.00±7.05	83.86±8.10	73.38±3.75	71.74±3.41	70.59±3.94	71.41±4.97	63.69±0.73	64.08±1.20
MeanAttPool	77.80±3.28	77.69±3.36	52.40±2.53	-	76.94±1.69	-	91.67±7.14	90.44±8.69	74.09±3.89	74.69±4.07	74.41±6.45	72.33±5.27	72.38±0.60	72.03±0.26
Set2Set	60.20±8.21	59.19±8.39	43.13±4.31	-	68.17±5.82	-	91.11±6.67	88.86±9.39	70.72±9.31	69.43±9.49	68.82±4.96	65.05±5.96	70.71±2.20	70.47±2.12
SortPooling	69.60±3.14	76.45±3.42	48.53±4.27	-	74.44±2.18	-	83.33±5.56	84.92±9.33	74.05±4.41	72.72±4.75	56.47±7.53	59.56±9.42	69.61±3.02	50.29±3.98
Attpool-global	67.70±4.10	71.56±4.52	50.88±3.42	-	78.58±1.29	-	91.11±5.09	90.75±8.29	74.82±3.78	74.85±3.94	68.34±6.40	72.99±6.40	73.50±1.71	60.46±1.78
Attpool-local	66.10±1.37	66.53±3.34	51.93±3.81	-	79.90±1.10	-	81.11±7.11	84.13±9.45	74.64±3.74	74.56±2.48	70.00±5.55	72.56±4.23	75.24±1.11	68.12±2.46
gPool	77.60±2.27	77.60±2.27	54.80±2.74	-	**81.34±0.91**	-	87.72±7.19	85.22±8.64	78.52±4.18	77.23±4.13	73.15±3.91	72.42±3.87	71.99±1.61	72.26±1.82
SAGPool	77.10±3.18	76.71±2.79	51.93±3.06	-	74.44±3.64	-	77.78±7.45	68.74±8.59	74.33±3.89	72.15±4.04	65.88±4.20	61.65±5.22	73.43±2.28	73.31±2.29
DiffPool	74.80±3.22	74.99±3.09	50.93±4.05	-	72.15±1.46	-	90.00±6.94	89.53±6.63	71.18±6.14	70.84±4.44	70.03±5.99	66.69±5.82	74.29±0.51	74.16±0.56
ASAP	73.70±3.95	73.97±3.94	48.53±3.54	-	79.68±2.47	-	73.70±3.18	68.26±5.96	68.47±4.17	63.92±6.06	56.74±2.24	54.36±1.37	70.50±7.51	70.11±7.69
StructPool	73.90±3.24	74.60±4.82	50.93±4.02	-	70.20±1.99	-	88.33±6.78	92.69±6.49	79.01±4.36	77.99±4.45	69.12±4.21	69.48±9.30	72.37±2.14	52.41±1.14
HAP (ours)	**78.80±2.68**	**78.85±2.69**	**55.53±2.47**	-	77.27±2.10	-	**93.88±5.80**	**94.60±5.29**	**79.28±4.37**	**78.29±4.95**	**74.71±2.35**	**73.84±2.13**	**75.95±1.18**	**75.71±1.33**

the baseline methods, we conduct experiments under the default settings reported in the original work.

6.1.4 Evaluation Metrics

For graph classification task, we perform the same 10-fold cross-validation as SAGPool [21] on all of the seven datasets, then report the average accuracy and AUC score with standard deviation. For graph matching and similarity learning tasks, average accuracy and standard deviation is reported for 10 random seeds.

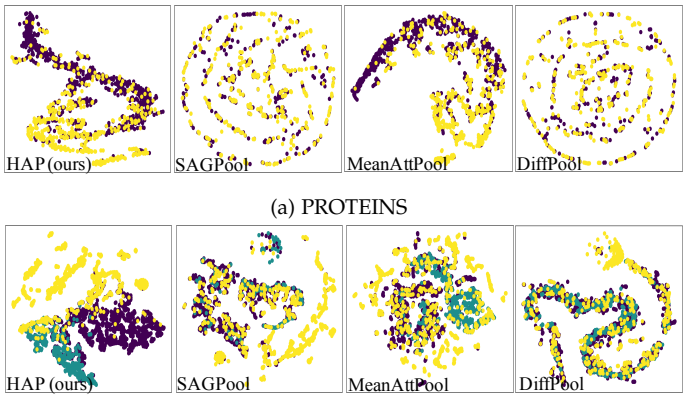
6.2 Task1: Graph Classification

We evaluate HAP on seven benchmark graph classification datasets and compare it with several state-of-the-art approaches belonging to different pooling categories respectively. For AttPool, we try different attention functions (global attention and local attention) to obtain the graph-level representations. For HAP, we try GAT and GCN for node & cluster embedding operation and report the better accuracy. Table 3 shows the 10-fold cross-validation results in terms of the mean and standard variation of classification accuracy and AUC score. The best performance on each dataset is marked in bold. We can observe that HAP obtains the best performance on six out of seven datasets with an average improvement of 6.07% for accuracy and 7.3% for AUC score.

Of all the graph pooling methods, universal pooling approaches are the most straightforward ones but achieve considerable effect, especially the SumPool which is consistent in underlying concept with our HAP. Intuitively, higher the quality of graph-level representations, better the graph classification result. The element-wise sum aggregator in SumPool tends to capture all node features in consideration of higher-order node dependency, but the generated graph-level representations fail to obtain sufficient quality, i.e., the quality of graph-level representations is not positively associated with how much node features are acquired. Irrelevant features that may interfere the results are obtained without reducing the weights, thus the final graph-level representations mixed with excessive irrelevant information is detrimental to the graph classification accuracy.

TopK pooling approaches produce score-based representations that drop nodes from the original graph with lower scores. As a result, potentially valuable information attached with these nodes and the related substructures may be ignored. From TABLE 3, the performance of *TopK* pooling approaches are universally inferior than other methods that capture more features or structural information. More damaging, SortPool and AttPool-global fail to return a result within 72 hours in practical execution. But there is an exception to the rule: gPool, with consistently better performance than other methods, even excels HAP on COLLAB. gPool computes scores by the multiplication of node feature matrix and a trainable projection vector, so that feature of each node is covered in the estimated scalar projection values by assigning with different weights. This crucial ingredient leads to the outstanding performance. As for the incredible performance on COLLAB, it might be due to the nature of COLLAB dataset. COLLAB covers scientific collaboration between authors. Nodes represent authors and edges indicate co-author relationship between authors. The classification task of each graph is to estimate the field the corresponding researcher belongs to. In this situation, it can be distinguished easily by the authors with Top-K quantity of papers that may be domain experts, while other unknown authors are actually noisy information. Nevertheless, our HAP still has advantages except for such exceptional circumstances.

Visualization: To further conceptualize the effectiveness of the learned graph-level representations, we provide a visualization of the t-SNE on PROTEINS and COLLAB dataset with features extracted by the methods HAP, SAGPool, MeanAttPool and DiffPool (Fig. 4). In each figure, points of different colors exhibits discernible graph clusters with different labels in the projected two-dimensional space. Note that the separability of the cluster border verifies the discriminative power. We can find that HAP performing consistently with MeanAttPool on PROTEINS shows better discriminability of the two classes than SAGPool and DiffPool. As for COLLAB, HAP is far superior to its competitors where three classes are clearly separated, all of which are in accordance with the results suggested in TABLE 3.



(a) PROTEINS

(b) COLLAB

Fig. 4: The t-SNE visualization of graph-level representations from HAP and three baselines on PROTEINS (above) and COLLAB (below). Different colors indicate graph samples with different labels.

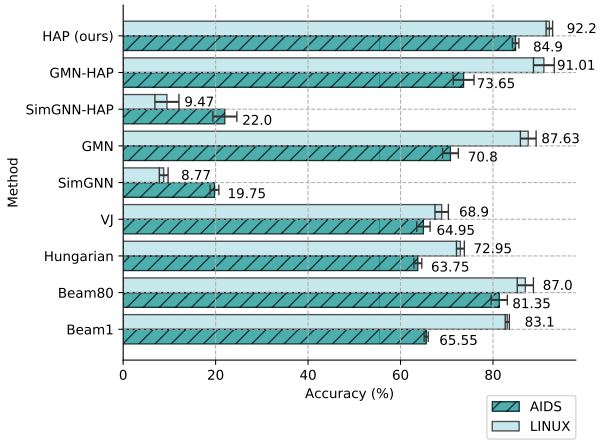


Fig. 5: Graph similarity accuracy in percent. The triplet similarity based on Beam1, Beam80, Hungarian and VJ is reflected by whether the relative GED is positive or negative.

TABLE 4: Graph matching accuracy in percent varying with graph size. GMN-HAP is a variant of GMN where the pooling algorithm in GMN is replaced by the graph coarsening module of HAP.

Model	$\|\mathcal{V}\|$=20	$\|\mathcal{V}\|$=30	$\|\mathcal{V}\|$=40	$\|\mathcal{V}\|$=50
GMN	95.89±0.70	97.47±0.26	95.91±0.77	98.74±1.09
GMN-HAP	97.70±0.38	98.36±0.52	98.48±0.13	99.07±0.53
HAP (ours)	**99.31±0.32**	**99.41±0.42**	**98.73±0.76**	**99.23±0.46**

6.3 Task2: Graph Matching

Four synthetic datasets are generated with different data size $|\mathcal{V}| \in \{20, 30, 40, 50\}$ for graph matching task. TABLE4 shows the graph matching results w.r.t. graph size.

GMN, specifically designed for graph matching task, makes the node embedding phase dependent on the pair through a cross-graph attention mechanism. However, as shown in TABLE4, HAP drastically boosts the matching accuracy up to 3.42% compared to GMN on graph size $|\mathcal{V}| = 20$. When increasing graph size, HAP remains stable while GMN decreases gracefully from graph size $|\mathcal{V}| = 30$ to $|\mathcal{V}| = 40$. This shows the key point: basic node embedding models have been perfectly capable of getting high-quality node-level representations. On the contrary, the core to enhancing graph matching accuracy is improving the quality of graph-level representations. After replacing the basic pooling module in GMN, the performance of GMN-HAP grows tremendously to be comparable with HAP, further confirming the strong ability of the proposed graph coarsening module.

6.4 Task3: Graph Similarity Learning

We show the results of HAP for graph similarity learning compared with both conventional approximate GED algorithms and GNN-based models on dataset AIDS and LINUX in Fig. 5. Note that evaluating graph similarity learning task requires benchmark datasets with ground-truth GEDs processed by the exact algorithm A*. A recent research [48] shows that "no currently available algorithm manages to reliably compute GED within reasonable time between graphs with more than 16 nodes". And the experiments on A* show that 10 nodes seem to be reaching the limit of its ability to deal with the problem. To address the gap, we only accept benchmark datasets with the max number of nodes no more than 10 in each graph. Our results demonstrate that HAP is capable of boosting the accuracy of state-of-the-art methods.

More specifically, for conventional approximate GED algorithms with high computational complexity, HAP improves accuracy by a relative gain of 16% and 14.21% on AIDS and LINUX, respectively. In regard to comparing with GNN-based models, HAP is overwhelming to SimGNN, which focuses more on optimizing the exact similarity score between graphs while neglecting the relativity. The result, in one aspect, reflects that a single-minded pursuit of the optimization of pairwise absolute similarity is not necessarily favorable to the relative similarity tasks which are more common in real-world applications to some extent. Similarly, HAP outperforms GMN by a margin of 14.1% and 4.57% on AIDS and LINUX, respectively. When replacing pooling methods in SimGNN and GMN with the proposed graph coarsening module, both of them achieve slight promotion and GMN-HAP obtains comparable accuracy with HAP. These results indicate that our HAP and coarsening module are conducive to a high-quality graph-level representation.

6.5 Ablation Studies

6.5.1 *Comparison of Graph Pooling Mechanisms*

To study the effectiveness of our proposed graph coarsening module, we fix other components of HAP framework, and replace our coarsening module with other four differentiable graph pooling methods, i.e., MeanPool, MeanAttPool, SAGPool and DiffPool, referring these variants as HAP-MeanPool, HAP-MeanAttPool, HAP-SAGPool and HAP-DiffPool, respectively. The performance of HAP and its four ablated variants on graph classification, graph matching and graph similarity learning task is shown in TABLE 5.

We observe that compared with other four ablated variants whose performance fluctuates wildly among tasks,

TABLE 5: Ablation study results on graph classification, graph matching and graph similarity learning. HAP-x is a variant of HAP where the graph coarsening module is replaced by x.

Ablated Model	Graph Classification								Graph Matching				Graph Similarity Learning	
	IMDB-B		MUTAG		PTC		HIV		v=20	v=30	v=40	v=50	AIDS	LINUX
	acc	auc	acc	auc	acc	auc	acc	auc						
HAP-MeanPool	72.24±4.35	72.94±2.59	85.00±7.05	83.86±8.10	70.59±3.94	71.41±4.97	63.69±0.73	64.08±1.20	53.03±1.99	54.28±1.86	54.68±1.32	58.64±0.53	66.35±2.25	63.45±2.85
HAP-MeanAttPool	77.80±3.28	77.69±3.36	91.67±7.14	90.44±8.69	74.41±6.45	72.33±5.27	72.38±0.60	72.03±0.26	89.08±2.89	83.47±0.90	84.47±1.73	87.81±1.17	76.97±1.34	88.16±0.86
HAP-SAGPool	76.33±1.25	76.79±1.28	73.33±7.37	56.29±8.50	64.12±4.32	60.64±4.52	63.21±5.06	62.99±4.81	60.74±1.11	61.94±1.68	58.22±1.51	58.52±1.36	73.25±1.17	65.34±1.19
HAP-DiffPool	75.75±3.56	75.95±3.73	86.11±6.21	80.50±7.80	67.65±5.50	66.99±5.91	69.40±4.42	69.59±4.48	63.93±1.48	63.14±0.86	65.85±1.63	61.38±0.43	79.52±1.21	90.72±0.59
HAP (ours)	**78.80±2.68**	**78.85±2.69**	**93.88±5.80**	**94.60±5.29**	**74.71±2.35**	**73.84±2.13**	**75.95±1.18**	**75.71±1.33**	**99.31±0.32**	**99.41±0.42**	**98.73±0.76**	**99.23±0.46**	**84.90±0.70**	**92.20±0.91**

our HAP achieves superior performances on all of the ten datasets for the three tasks. We also find that HAP-MeanPool is competent in graph classification task, but inferior on graph matching and graph similarity learning to our method by a margin of 18.55% to 46.28%. This validates that the multiformity features which may be redundant information in single-input graph classification task is crucial for multiple-input graph-level tasks to do horizontal comparison. On the contrary, HAP-MeanAttPool brings about performance benefits against other ablated variants. This indicates that global-wise information aggregation can be helpful for graph-level representation learning. Further, with the help of the proposed graph coarsening module, our HAP achieves adaptive graph structure sensibility based on a global-wise information aggregation, which utilizes both local structure and global pattern properties, thus contributing to a high-quality graph-level representation. Similarly, when comparing with HAP-DiffPool, a one-hop neighborhood aggregator, HAP can also improve graph-level representation quality by joining high-order dependency among nodes that may hold significant information. Moreover, the performance comparison between HAP-SAGPool and HAP reveals that our HAP can indeed retain key graph information that may be attached to the abandoned nodes.

TABLE 6: The effects of different number of graph coarsening modules in percent.

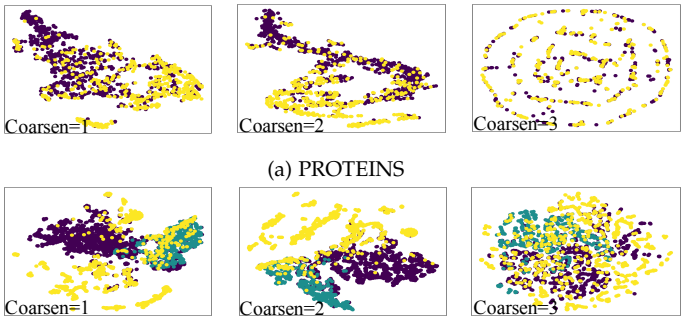
Model	Graph Matching				Graph Similarity Learning	
	$\|\mathcal{V}\|$=20	$\|\mathcal{V}\|$=30	$\|\mathcal{V}\|$=40	$\|\mathcal{V}\|$=50	AIDS	LINUX
baseline	53.03±1.99	54.28±1.86	54.68±1.32	58.64±1.32	66.35±2.25	63.45±2.85
coarsen=1	98.48±0.47	97.44±0.60	97.66±0.89	96.82±0.63	81.82±1.42	88.12±3.29
coarsen=2	**99.31±0.32**	**99.41±0.42**	98.73±0.76	99.23±0.46	**84.90±0.70**	**92.20±0.91**
coarsen=3	97.20±0.32	99.36±0.47	**99.23±0.30**	**99.25±0.14**	84.44±0.61	88.89±1.77

6.5.2 Comparison of Different Number of Graph Coarsening Module

TABLE 6 shows the performance of graph matching and graph similarity learning by adopting different numbers of graph coarsening modules in HAP. All experiments are conducted using HAP-MeanPool as the baseline with a fixed coarsen ratio for the same dataset. We observe that replacing the MeanAttPool with our graph coarsening module, denoted as Coarsen = 1, improves the performance by at least 15.47%, which can effectively demonstrate the significance of the proposed coarsening module. Furthermore, increasing coarsening modules from one to two can improve the performance by at most 4.08%. Finally, increasing coarsening modules from two to three makes the performance slightly fluctuate. These results demonstrate that the proposed graph coarsening module can improve the performance by coarsening graphs in a properly hierarchical manner.

Visualization: Fig. 6 visualizes how graph-level representations react with different number of graph coarsening modules in graph classification task. It can be seen that the challenging classification is progressively corrected with the number of graph coarsening modules increasing from one to two, but is easily to be misclassified when there are three coarsening modules.

Synthesizing the above results, the greater the number of graph coarsening modules, the more attached parameters and additional memory usage. To balance the performance and resource usage, we choose Coarsen = 2 as default settings.



(a) PROTEINS

(b) COLLAB

Fig. 6: The t-SNE visualization of graph-level representations of HAP with different number of graph coarsening modules on PROTEINS (above) and COLLAB (below). Different colors indicate graph samples with different labels.

6.5.3 Comparison of Generalization Performance

While most GNNs are designed to consider the generalization ability to unseen nodes, there are few researches in graph pooling area to address the generalization to unseen graphs. However, in practical applications such as protein molecular structure recognition, researchers are often interested in generalizing the knowledge learned from small-sized molecules to large-sized molecules with the same form of structures.

In this subsection, we justify the generalization capability of the models by training on small-size graphs and testing on large-sized graphs with the same edge probability for graph matching task. The results shown in TABLE 7 indicate that only HAP can achieve a natural generalization

of the small-sized results to the scenarios of large-sized graphs. This is credited to the key strength of HAP: it can effectively learn the global graph content that involves high-level pattern information for the training graph by GCont, thus preserving the pattern properties that are inherited between the training and testing graphs. When applying our graph coarsening module to GMN, GMN-HAP achieves a significant improvement of the prediction performance by 8.52% and 10.03%, respectively.

TABLE 7: Generalization performance in percent on graph matching task. Models are trained on graphs with $20 \leq |\mathcal{V}| \leq 50$ and tested on graphs with $|\mathcal{V}| = 100$ or $|\mathcal{V}| = 200$.

Model	$\|\mathcal{V}\|$=100	$\|\mathcal{V}\|$=200
GMN	85.37±0.85	74.35±1.33
GMN-HAP	93.89±0.44	84.38±0.78
HAP-MeanPool	56.66±0.66	57.27±0.89
HAP-MeanAttPool	83.84±0.30	88.11±0.84
HAP-SAGPool	57.88±0.90	57.89±1.25
HAP-DiffPool	63.86±0.66	59.89±0.60
HAP (ours)	**98.16±0.25**	**97.58±0.79**

7 Conclusion and Future Work

In this paper, we introduce a novel graph pooling framework HAP for hierarchical graph-level representation learning by adaptively leveraging the graph structures. The key innovation of HAP is the graph coarsening module, assisted by novel graph pattern property extracting technique GCont and cross-level attention mechanism MOA. HAP clusters local substructures through a newly proposed cross-level attention mechanism MOA. MOA mechanism helps it to naturally focus more on close neighborhood while effectively capture higher-order dependency that may contain important information. We also propose GCont, an auto-learned global graph content that sustains the cross-attention process. HAP leverages GCont to provide global guidance in graph coarsening. It extracts graph pattern properties to make the pre- and post-coarsening graph content maintain stable without loss of significant information. The learning of GCont also facilitates generalization across graphs with the same form of features. Theoretically analysis and extensive experiments demonstrate that HAP and the key component graph coarsening module achieve state-of-the-art performance on four downstream tasks.

HAP shows its potential to improve other graph learning methods by getting a more informative graph embedding. Furthermore, there are incredible opportunities for HAP to be further extended to more complex networks such as attributed networks and heterogeneous networks which may be more common in real-world applications.

8 Acknowledgments

This document is supported by National Natural Science Foundation of China (No. 62025208, No. 61932001 and No. 62002371), State Administration of Science Technology and Industry for National Defense Foundation(No. WDZC20205250104) and the National University of Defense Technology Foundation (No. ZK21-17).

References

[1] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[2] Y. Fan, X. Lu, D. Li, and Y. Liu, "Video-based emotion recognition using cnn-rnn and c3d hybrid networks," in *Proceedings of the 18th ACM International Conference on Multimodal Interaction*, 2016, pp. 445–450.

[3] D. Palaz, R. Collobert *et al.*, "Analysis of cnn-based speech recognition system using raw speech as input," Idiap, Tech. Rep., 2015.

[4] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of cnn and rnn for natural language processing," *arXiv preprint arXiv:1702.01923*, 2017.

[5] S. A. Myers, A. Sharma, P. Gupta, and J. Lin, "Information network or social network? the structure of the twitter follow graph," in *Proceedings of the 23rd International Conference on World Wide Web*, 2014, pp. 493–498.

[6] N. Shibata, Y. Kajikawa, and I. Sakata, "Link prediction in citation networks," *Journal of the American society for information science and technology*, vol. 63, no. 1, pp. 78–85, 2012.

[7] J. Hu, C. Guo, B. Yang, and C. S. Jensen, "Stochastic weight completion for road networks using graph convolutional networks," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1274–1285.

[8] Y. Li, C. Huang, L. Ding, Z. Li, Y. Pan, and X. Gao, "Deep learning in bioinformatics: Introduction, application, and perspective in the big data era," *Methods*, vol. 166, pp. 4–21, 2019.

[9] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv: Learning*, 2016.

[10] Y. Zhao, J. Qi, Q. Liu, and R. Zhang, "Wgcn: Graph convolutional networks with weighted structural features," *arXiv preprint arXiv:2104.14060*, 2021.

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

[12] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar, "Composition-based multi-relational graph convolutional networks," *arXiv preprint arXiv:1911.03082*, 2019.

[13] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," *arXiv preprint arXiv:1806.08804*, 2018.

[14] H. Gao and S. Ji, "Graph u-nets," *arXiv preprint arXiv:1905.05178*, 2019.

[15] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," *arXiv preprint arXiv:1904.12787*, 2019.

[16] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, "Simgnn: A neural network approach to fast graph similarity computation," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019, pp. 384–392.

[17] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," *arXiv: Learning*, 2016.

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

[19] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," *arXiv preprint arXiv:1511.06391*, 2015.

[20] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification." in *AAAI*, vol. 18, 2018, pp. 4438–4445.

[21] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3734–3743.

[22] Y. Pang, Y. Zhao, and D. Li, "Graph pooling via coarsened graph infomax," *arXiv preprint arXiv:2105.01275*, 2021.

[23] E. Ranjan, S. Sanyal, and P. P. Talukdar, "Asap: Adaptive structure aware pooling for learning hierarchical graph representations." in *AAAI*, 2020, pp. 5470–5477.

[24] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1263–1272.

[25] Y. Bai, H. Ding, Y. Qiao, A. Marinovic, K. Gu, T. Chen, Y. Sun, and W. Wang, "Unsupervised inductive graph-level representation learning via graph-graph proximity," *arXiv preprint arXiv:1904.01098*, 2019.

[26] J. Li, Y. Rong, H. Cheng, H. Meng, W. Huang, and J. Huang, "Semi-supervised graph classification: A hierarchical graph perspective," in *The World Wide Web Conference*, 2019, pp. 972–982.

[27] Z. Wang and S. Ji, "Second-order pooling for graph neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[28] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, pp. 2539–2561, 2011.

[29] J. Huang, Z. Li, N. Li, S. Liu, and G. Li, "Attpool: Towards hierarchical feature representation in graph convolutional networks via attention mechanism," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 6480–6489.

[30] X. Gao, W. Dai, C. Li, H. Xiong, and P. Frossard, "ipool–information-based pooling in hierarchical graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[31] Z. Zhang, J. Bu, M. Ester, J. Zhang, C. Yao, Z. Yu, and C. Wang, "Hierarchical graph pooling with structure learning," *arXiv preprint arXiv:1911.05954*, 2019.

[32] H. Gao, Y. Liu, and S. Ji, "Topology-aware graph pooling networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[33] D. Mesquita, A. Souza, and S. Kaski, "Rethinking pooling in graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[34] Y. G. Wang, M. Li, Z. Ma, G. Montufar, X. Zhuang, and Y. Fan, "Haar graph pooling," in *International conference on machine learning*. PMLR, 2020, pp. 9952–9962.

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

[36] H. Yuan and S. Ji, "Structpool: Structured graph pooling via conditional random fields," in *International Conference on Learning Representations*, 2019.

[37] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *International Conference on Machine Learning*. PMLR, 2020, pp. 874–883.

[38] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[39] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 1365–1374.

[40] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *arXiv preprint arXiv:2005.00687*, 2020.

[41] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

[42] K. Riesen and H. Bunke, "Iam graph database repository for graph based pattern recognition and machine learning," in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 2008, pp. 287–297.

[43] X. Wang, X. Ding, A. K. Tung, S. Ying, and H. Jin, "An efficient graph indexing method," in *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 2012, pp. 210–221.

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

[45] M. Neuhaus, K. Riesen, and H. Bunke, "Fast suboptimal algorithms for the computation of graph edit distance," in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 2006, pp. 163–172.

[46] S. Fankhauser, K. Riesen, and H. Bunke, "Speeding up graph edit distance computation through fast bipartite matching," in *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, 2011, pp. 102–111.

[47] K. Riesen and H. Bunke, "Approximate graph edit distance computation by means of bipartite graph matching," *Image and Vision computing*, vol. 27, no. 7, pp. 950–959, 2009.

[48] D. B. Blumenthal and J. Gamper, "On the exact computation of the graph edit distance," *Pattern Recognition Letters*, vol. 134, pp. 46–57, 2020.

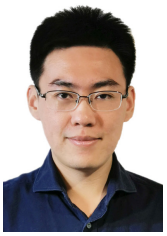
Ning Liu is a Ph.D. student in computer science and technology from College of Computer, National University of Defense Technology, Changsha, China. Her research interests include graph representation learning, graph data analytic, and graph processing systems.

Songlei Jian received the B.Sc. degree and Ph.D. degree in computer science from College of Computer, National University of Defense Technology, Changsha, China, in 2013 and 2019, respectively. She is currently an Assistant Research Fellow with the School of Computer, NUDT. Her research interests include representation learning, graph learning, multimodal learning and anomaly detection.

Dongsheng Li received the B.Sc. degree (with honors) and Ph.D. degree (with honors) in computer science from College of Computer, National University of Defense Technology, Changsha, China, in 1999 and 2005, respectively. He was awarded the prize of National Excellent Doctoral Dissertation of PR China by Ministry of Education of China in 2008. He is now a full professor at National Lab for Parallel and Distributed Processing, National University of Defense Technology, China. His research interests include parallel and distributed computing, Cloud computing, and large-scale data management.

Yiming Zhang (Member, IEEE) received the B.Sc. and M.Sc. degrees in mechanics engineering and the Ph.D. degree in computer science from the National University of Defense Technology (NUDT), Changsha, Hunan, China, in 2001, 2003, and 2008, respectively. He is currently an Associate Professor with the School of Computer, NUDT. He is an associate editor of IEEE Transactions on Services Computing. His current research interests include operating systems, networking, and distributed storage. He received the China Computer Federation (CCF) Distinguished Ph.D. Dissertation Award in 2011.

Zhiquan Lai received his Ph.D, M.S. and B.S. degrees in Computer Science from National University of Defense Technology (NUDT) in 2015, 2010 and 2008 respectively. He is currently an assistant professor in the National Key Laboratory for Parallel and Distributed Processing of NUDT. He worked as a research assistant at Department of Computer Science, the University of Hong Kong during Oct. 2012 to Oct. 2013. His current research interests include high-performance system software, distributed machine learning, and power-aware computing.



Hongzuo Xu received the bachelor's and master's degree from the National University of Defense Technology, China in 2017 and 2019. He is currently pursuing his Ph.D. degree in the College of Computer, National University of Defense Technology. His research interests include anomaly detection, weakly-supervised learning and data mining. He is a student member of the IEEE.