Multi-grained Semantics-aware Graph Neural Networks

Zhiqiang Zhong, Cheng-Te Li, and Jun Pang

1 INTRODUCTION

In many real-world applications, such as social networks, recommendation systems, and biological networks, data can be naturally organised as graphs [10]. Nevertheless, working with this powerful node and graph representations remains a challenge, since it requires integrating the rich inherent features and complex structural information. To address this challenge, Graph Neural Networks (GNNs), which generalise deep neural networks to graph-structured data, have drawn remarkable attention from academia and industry, and achieve state-of-the-art performances in a multitude of applications [32], [43]. The current literature on GNNs can be used for tasks with two categories. One is to learn node representations to perform tasks such as link prediction [41] and node classification [15], [34]. The other is to learn graph representations for tasks, such as graph classification [9], [37], [39].

On node-wise tasks, existing GNN models on learning node representations rely on a similar methodology that utilises a GNN layer to aggregate the sampled neighbouring nodes’ features in a number of iterations, via non-linear transformation and aggregation functions. Its effectiveness has been widely proved, however, a major limitation of these GNN models is that they are inherently flat as they only propagate information across the observed edges in the original graph. Thus, they lack the capacity to encode features in the high-order neighbourhood in the graphs [1]. For example, in an academic collaboration network, flat GNN models could capture the micro semantic (e.g., co-authorships) between authors, but neglect their macro semantics (e.g., belonging to different research institutes).

On the other hand, graph classification is to predict the label associated with an entire graph by utilising the given graph structure and initial node features. Nevertheless, existing GNNs for graph classification are unable to learn graph representations in a multi-grained manner, which is crucial to encode better meso- and macro-level graph semantics hidden in the graphs for many practical applications such as drug design [27] and program analysis [18]. To remedy this limitation, novel pooling approaches have been proposed, where sets of nodes are recursively aggregated to form super nodes in the pooled graph. DIFFPOOL [37] is a differentiable pooling operator but its assignment matrix is too dense [4] to apply on large graphs. G-U-NET [9], SAGPOOL [16], GXN [20], and ASAP [26] are four recently proposed methods that adopt the Top-k selection strategy to address the sparsity concerns of DIFFPOOL. They score nodes based on a learnable projection vector and select a fraction of high scoring nodes as super nodes. However, the pre-defined pooling ratio limits these models’ adaptivity on graphs with different sizes, and the Top-k selection may easily lose important node features or graph structure by simply ignoring low scoring nodes. As shown in Figure 1 (Section 2), different numbers of k will significantly affect the number of covered nodes of super nodes in the pooled graph, which means the important nodes’ features could get lost during the trivial pooling strategy. The hyper-parameter k is also crucial for the final performance [9], thus reduces their convenience in applications.

Index Terms—Graph neural networks, multi-grained semantic, hierarchical structure, representation learning
In this work, we propose a novel framework, Adaptive Multi-grained Graph Neural Networks (AdamGNN), which integrates graph convolution, adaptive pooling and unpooling operations into one framework to generate both node and graph level representations interactively. Unlike the above-mentioned GNN models, we treat node and graph representation learning tasks in a unified framework so that they can collectively optimise each other during training. In modelling multi-grained semantics, the adaptive pooling and unpooling operators preserve the important node features and hierarchical structural features. More concretely, as shown in Figure 2-(a), we employ (i) an adaptive graph pooling (AGP) operators to generate a multi-grained structure based on the derived primary node representations by a GNN layer, (ii) graph unpooling (GUP) operators to further distribute the explored meso- and macro-level semantics to the corresponding nodes of the original graph, and (iii) a flyback mechanism to integrate all received multi-grained semantic messages as the evolved node representations. Besides, the attention-enhanced flyback aggregator provides a reasonable explanation of the importance of messages from different grains. Experimental results reveal the effectiveness of AdamGNN, and the ablation and empirical studies confirm the effectiveness and flexibility of different components in AdamGNN. At last, through case studies, AdamGNN is shown to highlight variant-range node interactions in different graph datasets.

Our contributions can be summarised as follows. (1) We propose a novel framework, AdamGNN, that adaptively integrates multi-grained semantics into node representations and achieves mutual optimisation between node-wise and graph-wise tasks in one unified process. (2) An adaptive and efficient pooling operator is devised in AdamGNN to generate the multi-grained structure without introducing any hyper-parameters. (3) An attention-based flyback aggregation can provide model explainability on how different grains benefit the prediction tasks. (4) Extensive experiments on 14 real-world datasets demonstrate the promising performance of AdamGNN, along with providing insightful explanations with case studies.

2 RELATED WORK

Graph neural networks. The existing GNN models can be generally categorised into spectral and spatial approaches. The spectral approach utilises the Fourier transformation to define convolution operation in the graph domain [3]. However, its incurred heavy computation cost hinders it from being applied to large-scale graphs. Later on, a series of spatial models drawn remarkable attention due to their effectiveness and efficiency in node-wise tasks [6], [7], [8], [10], [15], [22], [29], [31], [34], such as link prediction, node classification and node clustering. They mainly rely on the flat message-passing mechanism that defines convolution by iteratively aggregating messages from the neighbouring nodes. Recent studies have proved that the spatial approach is a special form of Laplacian smoothing and is limited to summarising each node’s local information [3]. Besides, they are either unable to capture global information or incapable of aggregating messages in a multi-grained manner to support graph classification tasks.

Graph pooling. Pooling operation overcomes GNN’s weakness in generating graph-level representation by recursively merge sets of nodes to form super nodes in the pooled graph. DIFFPOOL [37] is a differentiable pooling operator, which learns a soft assign matrix that maps each node to a set of clusters, treated as super nodes. Since this assignment is rather dense that incurs high computation cost, it is not scalable for large graphs [4]. Following this direction, a Top-k selection based pooling layer (G-U-Net) is proposed to select important nodes from the original graph to build a pooled graph [9]. SAGPOOL [16] and ASAP [26] further use attention and self-attention for cluster assignment, GNN [20] uses mutual information estimation for super node selection. They address the problem of sparsity in DIFFPOOL, however, such a manual-defined hyper-parameter k is quite sensitive to the final performance [9], thus limits the adaptivity of these models on graphs of different sizes. In addition, as shown in Figure 1 different k values will significantly affect the number of covered nodes in the graph, which means the important node features could get lost during the trivial pooling operation. Note that nodes covered by a super node refer to nodes involved in the super node’s aggregation tree.

Recently, EIGENPOOL [23] proposes a pooling operator based on graph Fourier which does not rely on the Top-k selection strategy, STRUCTPOOL [39] designs strategies to involve both node and graph structures, and includes
3 PROPOSED APPROACH

3.1 Preliminaries
A graph with \( n \) nodes can be formally represented as \( G = (V, E, X) \), where \( V = \{v_1, \ldots, v_n\} \) is the node set, \( E \subseteq V \times V \) denotes the set of edges, and \( X \in \mathbb{R}^{n \times \pi} \) represents nodes' features (\( \pi \) is the dimensionality of node features). Besides, \( V \) and \( E \) can be summarised in adjacency matrix \( A \in \{0, 1\}^{n \times n} \).

For node-wise tasks, the goal is to learn a mapping function \( f_n : G \rightarrow \mathbb{Z} \), where \( \mathbb{Z} \in \mathbb{R}^d \), and each row \( z_i \in \mathbb{Z} \) corresponds to node \( v_i \)'s representation. For graph-wise tasks, similarly it aims to learn a mapping \( f_g : G \rightarrow \mathbb{Z} \), where \( G = \{G_1, G_2, \ldots\} \) is a set of graphs, each row \( z_i \in \mathbb{Z} \) corresponds to the graph \( G_i \)'s representation. The mapping function's effectiveness \( f_n \) and \( f_g \) is evaluated by applying \( Z \) to different downstream tasks.

Primary node representation. We use Graph Convolution Network (GCN) \([15]\) as an example primary GNN encoder to obtain the node representation:

\[
H^{(\ell+1)} = \text{ReLU}(D^{-\frac{1}{2}} \hat{A} D^{\frac{1}{2}} H^{(\ell)} W^{(\ell)}),
\]

where \( \hat{A} = A + I \), \( D = \sum_{i} \hat{A}_{ii} \) and \( W^{(\ell)} \in \mathbb{R}^n \times d \) is a trainable weight matrix for layer \( \ell \). \( H^{(\ell)} \) is the generated node representation of layer \( \ell \) which is defined as the primary node representations \( H = H^{(0)} \). Node representations are generated based on each target node's local neighbours, which are aggregated via learning based on the adjacency matrix \( A \). GCN cannot capture meso/macro-level knowledge, even with stacking multiple layers. Hence we term such generated node representations as primary node representations.

3.2 Adaptive Graph Pooling for Multi-grained Structure Generation
The proposed model, AdamGNN, adaptively generates a multi-grained structure to realise the collective optimisation between the node and graph level tasks within one unified framework. The key idea is that applying an adaptive graph pooling operator to present the multi-grained semantics of \( G \) explicitly and improve the node representation generation with the derived meso/macro information. While AdamGNN is usually performed under multiple levels of granularity (\( T \) different grains), in this section, we present how level \( t \)'s super graph is adaptively constructed based on graph of level \( t-1 \), i.e., \( G_{t-1} = (V_{t-1}, \mathcal{E}_{t-1}, X_{t-1}) \).

Ego-network formation. We initially consider the graph pooling as an ego-network selection problem, i.e., each ego node can determine whether to aggregate its local neighbours to form a super node, resolving the dense issue of DiffPOOL by avoiding to use a dense assignment matrix. As shown in Figure 3(b)-(i), each ego-network \( c_{\lambda} \) contains the ego and its local neighbours \( N_{\lambda}^{v_i} \) within \( \lambda \)-hops, i.e., \( N_{\lambda}^{v_i} = \{v_j \mid d(v_i, v_j) \leq \lambda\} \), where \( d(v_i, v_j) \) means the shortest-path length between \( v_i \) and \( v_j \). Thus an ego-network can be formally presented as: \( c_{\lambda}(v_i) = \{v_j \mid \forall v_j \in N_{\lambda}^{v_i}\} \), and a set of ego-networks \( C_{\lambda} = \{c_{\lambda}(v_1), \ldots, c_{\lambda}(v_n)\} \) can be generated from \( G \). We will investigate the impact of the ego-network size in the ablation studies of Section 4.2.

Super node determination. Given \( G \) with \( n \) nodes has \( n \) ego-networks, forming a super graph with all ego-networks will blur the useful multi-grained semantics and lead to a high computation cost. We select a fraction of ego-networks from \( C_{\lambda} \) to organise the multi-grained semantics of \( G \). We make the selection based on a closeness score \( \phi(v_i) \) that evaluates the representativeness of the ego \( v_i \) to its local neighbours \( v_j \in c_{\lambda}(v_i) \). We first create a function to calculate the closeness score \( \phi_{ij} \) between \( v_i \) and \( v_j \):

\[
\phi_{ij} = f_{\phi_{on}}^{on}(v_i, v_j) \times f_{\phi_{in}}^{in}(v_i, v_j) \nonumber
\]

\[
= \text{Softmax}(\alpha^T \sigma(WH[j] \| WH[i])) \times \text{Sigmoid}(H^T[j] \cdot H[i]),
\]

where \( \alpha \) denotes the set of edges, and \( \sigma \) denotes the set of features. Besides, \( \phi \) requires positive and negative node sets sampling which introduces additional operation and the random sampling strategy brings unmanageable uncertainty on model performance \([36]\). On the contrary, AdamGNN is a unified framework that adaptively integrates multi-grained semantics into node representations, and achieves a mutual optimisation between node- and graph-wise tasks. Besides, AdamGNN supports efficient mini-batch pooling and unpooling, and also provides model explanation via the multi-grained semantics. Therefore, we believe AdamGNN’s framework is more general, effective and scalable than G-U-Net.
where $\overrightarrow{a} \in \mathbb{R}^{2n}$ is the weight vector, $\| \cdot \|$ is the concatenation operator and $\sigma$ is an activation function (LeakyReLU) to avoid the vanishing gradient problem [12] during the model training process. $f^{\text{lin}}(v_i, v_j) = \frac{\exp(\sigma(\overrightarrow{WH}[j] \overrightarrow{WH}[i]))}{\sum_{v_r \in N_i} \exp(\sigma(\overrightarrow{WH}[j] \overrightarrow{WH}[r]))}$ calculates one component of $\phi_{ij}$ considering the non-linear relationship between node $v_j$’s and ego $v_i$’s representations, and its output lies in $(0, 1)$ as a valid probability for ego-network selection. Meanwhile, we further add another component $f^{\text{lin}}(v_i, v_j) = \text{Sigmoid}(\overrightarrow{H}^T[j] \cdot \overrightarrow{H}[i])$ to supercharge $\phi_{ij}$ with the linearity between node $v_j$ and ego $v_i$ to capture more comprehensive information. Consequently, nodes have similar features and structure information to the ego will contribute to higher closeness scores. In the end, we produce the closeness score of $c_\lambda(v_i)$ as:

$$\phi_i = \frac{1}{|N_i^\lambda|} \sum_{v_j \in N_i^\lambda} \phi_{ij},$$

where $|N_i^\lambda|$ indicates the number of nodes in $N_i^\lambda$.

After obtaining ego-networks’ closeness scores, we propose an adaptive approach to select a fraction of ego-networks to form super nodes without pre-defined hyper-parameters (cf. the Top-$k$ selection strategy [9]). Our key intuition is that a high diameter ego-network could be composed of multiple low diameter ego-networks. Therefore, we intend first to find these low diameter ego-networks, posed of multiple low diameter ego-networks. Therefore, we allow overlapping between different selected ego-networks and utilise $N_i^\lambda$ instead of $N_i^\lambda$. If we adopt $N_i^\lambda$ here, the selected ego node $v_i$ cannot be involved in other ego-networks anymore. Following this, we can select a fraction of ego-networks to form super nodes at granularity level $t$.

**Proposition 1.** Let $\mathcal{G}$ be a connected graph with $n$ nodes, and a total number of $n$ ego-networks can be formed from the graph $\mathcal{G}$, i.e., $\mathcal{C}_\lambda = \{c_\lambda(v_1), c_\lambda(v_2), \ldots, c_\lambda(v_n)\}$. Each ego-network $c_\lambda(v_i)$ will be assigned with a closeness score $\phi_i$. Then, there exist at least one ego-network $c_\lambda(v_i)$ that satisfies $\phi_i > \phi_j, \forall v_j \in N_i^1$.

**Proof of Proposition 1.** For $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ with $n$ nodes. $n$ ego-networks can be generated by following the procedures, $c_\lambda(v_i) = \{\{j \mid \forall j \in N_i^1\}\}$, and each ego-network will be given a closeness score $\phi_i$ as Equation 3. We assume that these cluster closeness scores are not all the same, thus there exists at least one maximum $\phi_{\text{max}}$. Hence, the clusters with closeness score $\phi_{\text{max}}$ satisfy the requirements of ego-network selection requirement that:

$$\phi_{\text{max}} \geq \phi_j, \forall v_j \in N_i^1.$$  

where $N_i^1 = \{v_j | \text{if } d(v_i, v_j) = 1\}$. So, for any connected $\mathcal{G}$ with $n$ nodes, there exists at least one cluster satisfies the requirements of our ego-network selection approach. $\square$

Proposition 1 ensures that our super node determination method can find at least one ego-network to generate a super graph for any graph. It guarantees the generality of our strategy.

Meanwhile, we would also retain nodes that do not belong to any selected ego-networks, denoted as $\tilde{N}_r$, to maintain the graph structure: $\tilde{N}_r = \{v_j | v_j \notin c_\lambda(v_i), \forall v_i \in \tilde{N}_p\}$. In this way, a super node formation matrix $\mathbf{S}_t \in \mathbb{R}^{n \times (|N_p| + |N_r|)}$ can be formed, where $(|N_p| + |N_r|)$ is number of nodes of the generated super graph, rows of $\mathbf{S}_t$ corresponds to the $n$ nodes of $\mathcal{G}_{t-1}$, and columns of $\mathbf{S}_t$ corresponds to the selected ego-networks ($N_p$) plus the remaining nodes ($\tilde{N}_r$). We have $S_t[i,j] = 0$, and columns of $\mathbf{S}_t$ corresponds to the selected ego-networks ($N_p$) plus the remaining nodes ($\tilde{N}_r$). We have $S_t[i,j] = 1$ if node $v_j$ belongs to the selected ego-network $c_\lambda(v_i)$ and $S_t[i,j] = 0$ if node $v_j$ is a remaining node corresponds to node $v_i$ in the super graph; otherwise $S_t[i,j] = 0$. The weighted super node formation matrix $\mathbf{S}_t$ can better maintain the relation between different super nodes in the pooled graph.

**Maintaining super graph connectivity.** After selecting the ego-networks and retaining nodes in level $t-1$, as shown in Figure 2(b)-(iii-iv), we construct the new adjacent matrix $\mathbf{A}_t$ for the super graph using $\tilde{A}_{t-1}$ and $\mathbf{S}_t$ as follows: $\mathbf{A}_t = \mathbf{S}_t^T \tilde{A}_{t-1} \mathbf{S}_t$. This formula makes any two super nodes connected if they share any common nodes or any two
nodes are already neighbours in \( \mathcal{G}_{t-1} \). In addition, \( \mathbf{A}_t \) will retain the edge weights passed by \( \mathcal{S}_{t-1} \) that involves the relation weights between super nodes. Eventually, we obtain a generated super graph \( \tilde{\mathcal{G}}_t \) at granularity level \( t \).

**Super node feature initialisation.** All nodes in the super graph \( \tilde{\mathcal{G}}_t \) need initial feature vectors to support the graph convolution operation. Recall that we have the closeness score as calculated in Eq. \( 2 \) between node \( v_j \) to the ego \( v_i \). However, this is not equivalent to the contribution of node \( v_j \)'s feature to the super node feature, since we need to compare the relationship strength between ego \( v_i \) and \( v_j \) with the relation between other \( v_r \in c_\lambda(v_i) \). Therefore, we further propose a super node feature initialisation method through a self-attention mechanism \([31]\). Specifically, it can be described as:

\[
\hat{X}_t[i] = H_{t-1}[i] + \sum_{v_j \in c_\lambda(v_i) \setminus v_i} \alpha_{ij} H_{t-1}[j], \tag{5}
\]

where \( H_{t-1} \) is the generated node representation by the \((t-1)\)-th primary GNN layer, i.e., at level \( t-1 \) with a similar method as Equation \( 4 \) \( \alpha_{ij} \) describes the importance of node \( v_j \) to the initial feature of \( c_\lambda(v_i) \) at level \( t \). And \( \alpha_{ij} \) can be learned as follows:

\[
\alpha_{ij} = \frac{\exp(\hat{a}_1^T \sigma(W(\phi_j, H_{t-1}[i]))H_{t-1}[i])}{\sum_{v_j \in c_\lambda(v_i) \setminus v_i} \exp(\hat{a}_1^T \sigma(W(\phi_j, H_{t-1}[i]))H_{t-1}[i])} \quad \text{where } \hat{a}_1 \in \mathbb{R}^{2\pi}
\]

is the weight vector. For the remaining nodes \( \bar{N}_r \) that do not belong to any super nodes, we keep their representations of \( H_{t-1} \) as initial node features.

### 3.3 Graph Unpooling

Different from existing graph pooling models \([4, 16, 25, 37, 39]\) which only coarsen graphs to generate graph representations, we aim to mutually utilise node-wise and graph-wise tasks to better encode multi-grained semantics into both node and graph representations under a unified framework. We design a mechanism to allow the learned multi-grained semantics to enrich the node representations of the original graph \( \mathcal{G} \) as shown in Figure 2(a). Vice versa, the updated node representation can further ameliorate the graph representation in the next training iteration. A reasonable unpooling operation that passes macro-level information to original nodes has not been well studied in the literature. For instance, Gao et al. \([9]\) directly relocate the super node back into the original graph and utilise other GNN layers to spread its message to other nodes. However, these additional aggregation operations cannot allow each node to receive meaningful information since some nodes may distant from super nodes, in such case these operations can exacerbate local-smoothing \([21]\).

We implement the unpooling process by devising a **top-down** message-passing mechanism, which endows GNN models with meso/macro level knowledge. Specifically, since \( \mathbf{S}_t \) records how nodes of \( \mathcal{G}_{t-1} \) form the super nodes of \( \mathcal{G}_t \), so we utilise \( \mathbf{S}_t \) to restore the generated ego-network representation at level \( t \) to that at level \( t-1 \) until we arrive at the original graph \( \mathcal{G} \), i.e., \( t \to 0 \), as follows:

\[
\hat{\mathbf{H}}_t = (\mathbf{H}_t^T \mathbf{S}_t^T \mathbf{S}_{t-1}^T \ldots \mathbf{S}_1^T)^T, \tag{6}
\]

where \( \hat{\mathbf{H}}_t \in \mathbb{R}^{nxd} \). At the end of each iteration, nodes in the original graph \( \mathcal{G} \) will receive high-level semantic messages from the different levels, i.e., \( \{\hat{\mathbf{H}}_1, \ldots, \hat{\mathbf{H}}_T\} \). As illustrated in Figure 2(b)-(iv-i), the graph unpooling process can be treated as an inverse process of adaptive graph pooling process.

### 3.4 Flyback Aggregation

Since the super graphs at different granularity levels present multi-grained semantics and how each node utilises the received semantic information with its flat representation is a challenging question. And nodes of the same graph may need different granularity levels’ information. Therefore, we propose a novel attention mechanism to integrate the derived representations at different levels, given by:

\[
Z = H + \sum_{\forall t} \beta \hat{\mathbf{H}}_t, \tag{7}
\]

where the attention score \( \beta \) estimates the importance of the message from level \( t \) given by:

\[
\beta_i[j] = \frac{\exp(\bar{a}_2^T \sigma(W(H_t[i])|H_t[j]))}{\sum_{\forall i} \exp(\bar{a}_2^T \sigma(W(H_t[i])|H_t[j]))} \quad \text{where } \bar{a}_2 \in \mathbb{R}^{2\pi}
\]

is the weight vector. We term this process as the flyback aggregation, which considers the attention scores of different levels and allows each node to decide whether/how to utilise semantic information from different granularity levels. We will verify the effectiveness of flyback aggregation in the ablation study of Section 4.2 and discuss the explainability in Section 4.3.

### 3.5 Training Strategy

Till now, there are still two challenges when training the model. The first is how to highlight the difference among nodes’ representations from different ego-networks. Nodes belong to neighbouring ego-networks receive closely related messages from super nodes, since their super nodes are connected in the super graph, and local smoothing makes their representation vectors similar. Representations of proximal nodes could be further closer to each other in the representation latent space. To address this problem and enhance the discrimination capability between ego-networks, we exploit a self-optimisation strategy \([33]\), which let nodes in different ego-networks distinguishable. Specifically, we use the Student’s \( t \)-distribution (\( t \)) as a kernel to measure the similarity between representation vectors of \( v_j \) and ego \( v_i \)’s feature to the second power and then normalises by frequency per ego-network:

\[
q_{ij} = \frac{(1+||Z[j]-Z[i]||^2/\mu)}{\sum_{i'}(1+||Z[j]-Z[i']||^2/\mu)} - \frac{1}{\mu}, \quad \text{where } v_j \in c_\lambda(v_i), \quad v_i' \text{ are other ego nodes, } \mu \text{ are the degrees of freedom of Student’s } t\text{-distribution. Following this, } q_{ij} \text{ can be integrated the probability of assigning node } v_j \text{ to ego } v_i. \text{ In this paper, we set } \mu = 1 \text{ the same as } [33]. \text{ After, we propose to learn better node representations by matching } Q \text{ to the auxiliary target distribution } P, \text{ and we choose the proposition of } [33] \text{ which first raises } q \text{ to the second power and then normalises by frequency per ego-network:}
\]

\[
p_{ij} = \sum_{i'} (q_{ij}/q_{i'i}) \quad \text{where } q_i = \sum_{j} q_{ij}. \text{ Therefore, apart from the task-related loss function } L_{\text{task}}, \text{ we further define a KL divergence loss as:}
\]

\[
L_{KL} = KL(P \parallel Q) = \sum_{v_j} \sum_{v_i} p_{ij} \log\frac{p_{ij}}{q_{ij}} \tag{8}
\]

The second challenge is to avoid the over-smoothing problem that nodes of a graph tend to have indistinguishable representations. GNN is proved as a special form of
where $L$ function will demonstrate the effectiveness of each component of loss $L$. We have presented the idea of AdamGNN and the design of multi-grained structure with $t$-th level (line 3-13) with the proposed adaptive graph pooling operator. Meanwhile, we also propose a method to define the initial features of pooled super nodes (line 14-19). The graph connectivity of the pooled graph is maintained by line 20. We apply GNN encoder on the pooled graph to summarise the relationships between super nodes (line 21) to learn macro-grained semantics of $t$-th granularity level. The learned multi-grained semantics will be further distributed to the original graph following an unpooling operator (line 22). Last, the flyback aggregator generates the meso/macro level knowledge from different levels as the node representations of $G$ (line 24), and additional READOUT operators [7] produce the node representations as to the graph representation (line 25).

**Model scalability.** According to the design for AdamGNN framework, we can find that the primary node representation learning module of each level and the adaptive graph pooling and unpooling operators are categorised as a local network algorithm [25], which only involves local exploration of the network structure. Therefore, our design enables AdamGNN to scale to representation learning on large-scale networks and to be friendly to distributed computing settings [25]. We present instances that utilise multi-GPU computing framework to accelerate the training process of AdamGNN in Section 4.3.

### 4 Experiments

#### 4.1 Experimental Setup

We evaluate our proposed model, AdamGNN, on 14 benchmark datasets, and compare with 16 competing methods over both node- and graph-wise tasks, including node classification, link prediction and graph classification.

| Dataset       | #Nodes | #Edges | #Features | #Classes | #Train | #Val | #Test |
|---------------|--------|--------|-----------|----------|--------|------|-------|
| ACM           | 3,025  | 13,126 | 1,280     | 3        | 60-500-1,000 |
| Citeseer      | 3,327  | 4,552  | 3,705     | 6        | 120-500-1,000 |
| Cora          | 2,708  | 5,278  | 1,433     | 7        | 140-500-1,000 |
| DBLP          | 4,057  | 3,528  | 334       | 4        | 80-500-1,000  |
| Emails        | 799    | 10,182 | N.A.      | 18       | 180-309-310  |
| Pubmed        | 19,717 | 88,648 | 508       | 3        | 60-500-1,000  |
| Wiki          | 2,405  | 1,2178 | 4973      | 17       | 481-962-962   |
| Ogbn-arxiv    | 169,345| 1,166,243| 128     | 40       | N.A.           |

#### 4.2 Data statistics for node-wise tasks and the split for the semi-supervised node classification task. N.A. means a dataset does not contain node attributes or does not support semi-supervised settings.

| Dataset       | #Nodes (avg) | #Edges (avg) | #Features | #Classes |
|---------------|--------------|--------------|-----------|----------|
| NC1019        | 4,127        | 29.69        | 32.3      | 3        |
| NCI109        | 4,127        | 29.68        | 32.33     | 3        |
| D&D           | 1,178        | 284.32       | 715.66    | 89       |
| MUTAG         | 188          | 17.93        | 19.79     | 7        |
| Mutagenicity  | 4,337        | 30.32        | 30.77     | 14       |
| PROTEINS      | 1,113        | 39.06        | 72.82     | 32       |

**Datasets.** To validate the effectiveness of our model on real-world applications, we adopt datasets come from different domains with different topics and relations. We use 8 datasets for node-wise tasks (data statistics are summarised in Table 2). Ogbn-arxiv [13], ACM [2], Cora [15].

#### 3.6 Algorithm

We have presented the idea of AdamGNN and the design details of each component in Section 3. Given a graph $G$, we first apply a primary GNN encoder to generate the primary node embedding (line 1). Then we construct a multi-grained structure with $t$-th level (line 3-13) with the proposed adaptive graph pooling operator. Meanwhile, we also propose a method to define the initial features of pooled super nodes (line 14-19). The graph connectivity of the pooled graph is maintained by line 20. We apply GNN encoder on the pooled graph to summarise the relationships between super nodes (line 21) to learn macro-grained semantics of $t$-th granularity level. The learned multi-grained semantics will be further distributed to the original graph following an unpooling operator (line 22). Last, the flyback aggregator generates the meso/macro level knowledge from different levels as the node representations of $G$ (line 24), and additional READOUT operators [7] produce the node representations as to the graph representation (line 25).

To validate the effectiveness of our model on real-world applications, we adopt datasets come from different domains with different topics and relations. We use 8 datasets for node-wise tasks (data statistics are summarised in Table 2). Ogbn-arxiv [13], ACM [2], Cora [15].
Citee[15] and Pubmed[15] are paper citation graph datasets. DBLP[2] is an author graph dataset from the DBLP dataset. Emails[17] is an email communication graph dataset. Wiki[35] is a webpage graph dataset.

For the graph classification task, we adopt 6 bioinformatics datasets[39] (data statistics are summarised in Table[3]). D&D and PROTEINS are dataset containing proteins as graphs. NCI1 and NCI109 involve anticancer activity graphs. The MUTAG and Mutagenicity consist of chemical compounds divided into two classes according to their mutagenic effect on a bacterium. Note that all datasets can be downloaded with our published code automatically.

Competing methods. For node-wise tasks, we adopt 10 competing methods that include 7 GNN models with flat message-passing mechanism, and 2 state-of-the-art methods that contain a hierarchical structure: MLP[11], GCN[15], GraphSAGE[10], GAT[31], GIN[34], PNA[7], GCNII[6], GRAND[8], GXN[20] and G-U-NET[9]. For the graph classification task, except for GIN, PNA, GXN and G-U-NET which support graph-wise tasks, we adopt extra 7 competing methods that involve the state-of-the-art models: 3WL-GNN[24], SORTPOOL[12], DIFFPOOL[37], SAGPOOL[16], EIGENPOOL[23], STRUCTPOOL[39] and ASAP[26]. Note that we already carefully discussed these competing methods in Section[2] therefore, we do not repeat the method description to save page space. The competing model implementations can be found in our published project on Github.

Evaluation settings. For the node-wise tasks, we follow the supervised node classification (Sup-NC) settings of PNA[7], i.e., using two sets of 10% labelled nodes as validation and test sets, with the remaining 80% labelled nodes used as the training set. Meanwhile, we follow the semi-supervised node classification (Semi-NC) settings of GCN[15], and the data split is shown in Table[2]. That said, for Cora, Citeeseer and Pubmed, we use the fixed splits and for other datasets, we randomly assign 20 labelled nodes for each class for training, and 500 and 1000 nodes for validation and testing, respectively. Note that since the Email does not have a sufficient number of nodes for classic Semi-NC setting, we choose 10 labelled nodes for each class for training, and the rest data is evenly separated as validation and test sets. Wiki is imbalanced, where some classes only have very few labelled nodes, e.g., class 12 has 9 labelled nodes and class 4 has 10 labelled nodes, which cannot support Semi-NC settings. Therefore, we follow Sup-NC settings to split Wiki for the Semi-NC experimental parts but use only 20% labelled nodes for training and the remaining nodes for validation and testing, respectively. Ogbn-arxiv follows the fixed split of OGB leaderboard[13]. For the Link Prediction (LP) task, we follow the settings of [38], i.e., using two sets of 10% existing edges as validation and test sets, with the remaining 80% edges used as the training set. Note that, an equal number of non-existent links are randomly sampled and used for every set. We present the average performance of 10 times experiments with random seeds. The AUC score evaluates link prediction, and node classification tasks are evaluated by accuracy. We conduct the experiments with random parameter initialisation with 10 random seeds and report the average performance.

For the graph-wise task, i.e., graph classification (GC) task, we perform all experiments following the pooling pipeline of SAGPOOL[16]. 80% of the graphs are randomly selected as training, and the rest 10% graphs are used for validation and testing, respectively. We conduct the experiments using 10-fold cross-validation and report the average classification accuracy on 10 random seeds.

Model configuration. For all methods, we set the embedding dimension $d = 64$ and utilise the same learning rate $= 0.01$, Adam optimiser, number of training epochs $= 1000$ with early stop (100). In terms of the neural network layers, we report the one with better performance of GCNII with better performance among {8, 16, 32, 64, 128}; for other models, we report the one with better performance between 2–4; For all models with hierarchical structure (including AdamGNN), we use GCN as the GNN encoder for fair comparison. In terms of the number of levels that is required by hierarchical models, we present the one with better performance, between 2–5. On other hyper-parameter settings of competing methods, we employ the default values of each competing method as shown in the paper’s official implementation. Particularly, for AdamGNN, by tuning the hyper-parameters based on the validation set, we have $\gamma = 0.1$ and $\delta = 0.01$ for Eq. [10] for the experiments to let loss values lie in a reasonable range, i.e., (0, 10). We employ Pytorch and PyTorch Geometric to implement all models. Experiments were conducted with GPU (NVIDIA Tesla V100) machines.

4.2 Experimental Evaluation and Ablation Study

Performance on node-wise tasks. We compare AdamGNN with 7 GNN models and one pooling-based model, i.e., G-U-NET, since other pooling approaches do not provide an unpooling operator and thus cannot support node-wise tasks. Results on node classification (with supervised and semi-supervised settings) are summarised in Table[4]. They show that AdamGNN can outperform most competing methods with up to 10.47% and 5.39% improvements on semi-supervised and supervised settings, respectively. AdamGNN brings the most significant improvement in Wiki data with semi-supervised settings, and the competing method that only adopts node features, i.e., MLP, achieve terrible accuracy, 17.46%. We argue that because the node features and node labels are weakly correlated in this dataset, multi-grained semantics provided by AdamGNN help to ameliorate the performance.

Link prediction results in Table[5] show that AdamGNN can significantly outperform the 7 competing methods by up to 25.3% improvement in terms of AUC. It indicates the versatility of AdamGNN on different node-wise tasks and exhibits the usefulness of modelling multi-grained semantics into node representations. Similar to node classification task, AdamGNN again brings the most significant AUC improvement on the Wiki dataset, i.e., achieving 29.76% improvement compared with the flat GNN models.

Performance on graph-wise task. Experimental results are summarised in Table[6]. It is apparent that our AdamGNN achieves the best performance on 4 of the 6 datasets, and consistently outperforms most of competing pooling-based techniques by 1.76% improvement. For the datasets MUTAG and PROTEINS, our results are still competitive since
Ablation study of different loss functions. The loss function of our AdamGNN consists of three parts, i.e., $\mathcal{L}_{\text{Task}}, \mathcal{L}_R$ and $\mathcal{L}_{KL}$. We examine how each part contributes to the performance. Table 2 provides the results. For the link prediction task, we have $\mathcal{L} = \mathcal{L}_R + \mathcal{L}_{KL}$, since $\mathcal{L}_{Task}$ equals to $\mathcal{L}_R$. Thus, two comparison experiments are missing in link prediction. From the results, we can see that $\mathcal{L}_R$ can significantly improve the performance over three tasks. This is because it can eliminate the over-smoothing problem caused by the received messages from different granularity levels. Meanwhile, $\mathcal{L}_{KL}$ can slightly improve the results of node-wise tasks as well.

### Table 4

| Models | ACM | Citeeseer | Cora | Emails | DBLP | Pubmed | Wiki | Ogbn-arxiv |
|--------|-----|-----------|------|--------|------|--------|------|-----------|
| MLP [31] | 0.965 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |
| GCN [15] | 0.957 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |
| GraphSAGE [10] | 0.956 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |
| GAT [31] | 0.955 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |
| GCNII [6] | 0.954 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |
| PNA [7] | 0.953 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |
| GCNII [5] | 0.952 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |
| GRAND [4] | 0.951 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |

PNA and ASAP only slightly outperform AdamGNN. Nevertheless, AdamGNN is still better than other baselines. This is because our model involves adaptive pooling and unpooling operators to update node- and graph-wise information interactively, and further enhance the representations of nodes and graphs during the training process.

Ablation study of different loss functions. The loss function of our AdamGNN consists of three parts, i.e., $\mathcal{L}_{\text{Task}}, \mathcal{L}_R$ and $\mathcal{L}_{KL}$. We examine how each part contributes to the performance. Table 2 provides the results. For the link prediction task, we have $\mathcal{L} = \mathcal{L}_R + \mathcal{L}_{KL}$, since $\mathcal{L}_{Task}$ equals to $\mathcal{L}_R$. Thus, two comparison experiments are missing in link prediction. From the results, we can see that $\mathcal{L}_R$ can significantly improve the performance over three tasks. This is because it can eliminate the over-smoothing problem caused by the received messages from different granularity levels. Meanwhile, $\mathcal{L}_{KL}$ can slightly improve the results of node-wise tasks as well.

### Table 5

| Models | ACM | Citeeseer | Cora | Emails | DBLP | Pubmed | Wiki | Ogbn-arxiv |
|--------|-----|-----------|------|--------|------|--------|------|-----------|
| MLP [31] | 0.965 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |
| GCN [15] | 0.957 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |
| GraphSAGE [10] | 0.956 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |
| GAT [31] | 0.955 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |
| GCNII [6] | 0.954 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |
| PNA [7] | 0.953 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |
| GCNII [5] | 0.952 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |
| GRAND [4] | 0.951 | 76.17 | 82.06 | 84.66 | 78.05 | 82.04 | 84.66 | 80.11 |

PNA and ASAP only slightly outperform AdamGNN. Nevertheless, AdamGNN is still better than other baselines. This is because our model involves adaptive pooling and unpooling operators to update node- and graph-wise information interactively, and further enhance the representations of nodes and graphs during the training process.

Ablation study of the flyback aggregator. Experimental results of node-wise tasks confirm that capturing multi-grained semantics in AdamGNN can help to learn more meaningful node representations. Here, we further study whether flyback aggregator can improve graph representations. Specifically, we aim to see how the flyback aggregator contributes to graph classification performance by removing and keeping it. The results are summarised in Table 8. It is
clear that the node representations enhanced by the flyback aggregation can indeed improve the graph representation in the classification task.

### Table 9
Comparison of AdamGNN with different number of granularity levels in terms of different tasks. NC task follows the supervised settings.

| # Levels (J) | DBLP | Wiki | ACM | Citeseer | Emails | Mutagenicity |
|--------------|------|------|-----|----------|--------|-------------|
|              | LP   | LP   | NC  | NC       | NC     | GC          |
| 1            | 0.951| 0.912| 92.60| 77.68    | 86.83  | 78.16       |
| 2            | 0.958| 0.913| 93.38| 74.67    | 91.88  | 82.04       |
| 3            | 0.959| 0.917| 94.37| 76.15    | 90.61  | 81.58       |
| 4            | 0.965| 0.920| 90.84| 78.92    | -      | 81.01       |

### Ablation study of different primary GNN encoders
We present an ablation study to investigate the influence of different primary GNN encoders. We adopted GCN as the default primary GNN encoder in Table 11. The table demonstrates that AdamGNN w/ GCN demonstrates a good performance gain over other baselines, which shows that AdamGNN can still benefit from the learned multi-grained semantics.

### Table 10
Comparison of AdamGNN with different number of granularity levels in terms of different tasks. NC task follows the supervised settings.

| Models      | Cora | Citeseer | Pubmed |
|-------------|------|----------|--------|
| GCN         | 81.50| 71.13    | 79.02  |
| AdamGNN w/ GCN | 84.66| 73.03    | 80.48  |
| PNA         | 79.54| 67.31    | 76.61  |
| AdamGNN w/ PNA | 81.21| 70.90    | 78.73  |
| GCNII       | 85.43| 72.37    | 80.03  |
| AdamGNN w/ GCNII | 85.81| 74.13    | 81.37  |

### 4.3 More Model Analysis

#### Running time comparison
We present the average epoch training time of different node and graph classification models in Table 11 and Table 12, respectively. In terms of node classification task, AdamGNN requires more training time due to the computation cost of $\alpha$ and $\beta$, similar to any attention-mechanism enhanced models [[30]]. However, AdamGNN is designed as a local network algorithm, maintaining good scalability; hence it can be easily accelerated by mini-batch and multi-GPUs computing frameworks [[14]]. It will significantly mitigate the computational issues. On the other hand, AdamGNN follows the sparse design similar to SAGPOOL, ASAP, striking a balance between performance improvement and maintaining proper time efficiency. DIFFPOOL and STRUCTPOOL employ a dense mechanism that is not easily scalable to larger graphs [[4]], and G-U-NET uses convolution operations, which bring additional computation cost to distribute the received information to the graph.

![Fig. 3. Ablation study of Ego-network size $\lambda$ in terms of different tasks. NC task follow the supervised settings.](image)

### Table 11
Average one epoch running time (in seconds) for supervised node classification task. ($\times 2$) means accelerated by 2 GPUs [[14]].

| Models            | ACM  | Citeseer | Cora  |
|-------------------|------|----------|-------|
| GCN               | 0.008| 0.008    | 0.009 |
| CAT               | 0.010| 0.011    | 0.011 |
| GCNII             | 0.045| 0.040    | 0.041 |
| G-U-NET           | 0.076| 0.064    | 0.069 |
| AdamGNN ($\times 1$) | 0.087| 0.072    | 0.074 |
| AdamGNN ($\times 2$) | 0.059| 0.050    | 0.052 |
| AdamGNN ($\times 3$) | 0.050| 0.047    | 0.048 |

### Table 12
Average one epoch running time (in seconds) for graph classification task.

| Models       | NCI1 | NCI109 | PROTEINS |
|--------------|------|--------|----------|
| DIFFPOOL     | 6.23 | 3.22   | 3.65     |
| SAGPOOL      | 1.95 | 1.55   | 0.45     |
| G-U-NET      | 4.58 | 4.45   | 1.46     |
| EIGENPOOL    | 3.88 | 3.54   | 1.38     |
| ASAP         | 2.04 | 1.83   | 1.09     |
| STRUCTPOOL   | 6.31 | 6.04   | 1.34     |
| AdamGNN      | 3.62 | 3.24   | 1.03     |

### Messages from different levels
We aim to figure out the importance of received messages from different granularity levels since messages from different levels contain the meso/macro-level knowledge encoded by super nodes. Here we consider the node classification on the ACM and DBLP as an example. Specifically, ACM’s paper nodes are labelled with 3 topics: database (DB), data mining (DM) and wireless communication (WC); DBLP’s author nodes have 4 research areas: AI, DB, DM and computer vision. The node classification task on these two datasets is predicting the paper/scholar’s research area. The attention scores of nodes that highlight the importance of different levels’ messages are plotted in Figure 4. We can find different distributions of attention weights over different granularity levels for various areas’ classifications. The relatively general topics,
In the left, we can see that the number of nodes of each level stabilises after 3–4 times of our adaptive pooling which illustrates AdamGNN will maintain the graph size at a proper level to avoid dense super graph.

Visualisation of attention weight for messages at different granularity levels. Dark colours indicate higher weights.

i.e., AI and WC, receive messages from different levels with relatively indistinguishable weights, i.e., higher attention scores of nodes are distributed across levels. The DM topic in two datasets has different attention patterns: it receives messages from level 1 with the greatest attention in ACM but receives greatest attention messages from level 3 in DBLP. This is because DM is not closely related to the other two topics of ACM dataset, Scholars of DM-related papers are less possible collaborate with researchers from DB or WC. DM papers are close to each other in the network. Thus DM papers only need to receive level 1 granularity semantic information summarised from neighbouring nodes. In contrast, DBLP’s other 3 topics are close to DM. DM-related scholars may cite any other scholars’ papers, and information related to DM is scattered over author nodes in DBLP network. Therefore, DM researchers tend to be characterised by level 3 semantics from a wide range.

Visualisation of different granularity levels.

Visualisation of different granularity levels. To better understand the process of learning multi-grained semantics, we report the relative numbers of nodes (i.e., node ratio concerning the original graph) at different granularity levels generated by AdamGNN in 7 datasets, as shown in Figure 5. In particular, we set the max number of granularity levels as 5, and level 0 indicates the original graph. We train AdamGNN with semi-supervised node classification and report the relative number of nodes and selected ego-nodes at each level. In the right, we can find out that the number of ego-nodes can stay stable after 1–2 times of our adaptive pooling which indicates that AdamGNN can effectively find a compact structure that contains multi-grained semantics.

In Table 13, we report the relative number of nodes (i.e., node ratio) concerning the original graph) at different granularity levels generated by AdamGNN in 7 datasets, as shown in Figure 5. In particular, we set the max number of granularity levels as 5, and level 0 indicates the original graph. We train AdamGNN with semi-supervised node classification and report the relative number of nodes and selected ego-nodes at each level. In the right, we can find out that the number of ego-nodes can stay stable after 1–2 times of our adaptive pooling which indicates that AdamGNN can effectively find a compact structure that contains multi-grained semantics.

Visualisation of adjacency matrices at different granularity levels. One of the fundamental limitations of existing GNNs is the inability of capturing long-range node interactions in the graph [19]. We find that AdamGNN can provide a possible solution to overcome this limitation. AdamGNN allows nodes to receive messages from far-away nodes with the support of the adaptive multi-grained structure. That said, the learned multi-grained structure can be regarded as a kind of short-cuts to let far-away nodes be aware of each other. We visualise these short-cut connections at different levels on the Wiki dataset in Figure 6. Figure-level 0 plots the original adjacency matrix, Figure-level 1 exhibits the learned short-cuts by the first pooled super graph (in green). Similarly, figure-levels 2 and 3 present the derived short-cuts by further stacking the adjacency matrices of the second (in blue) and third (in yellow) pooled graphs, respectively. We can clearly see that the original graph of the Wiki dataset is very sparse, and AdamGNN adds short-cuts between nodes with the help of the learned multi-grained structure. In this way, AdamGNN allows nodes to capture global information with few adaptively pooled graphs.

| Model     | 1-Layer | 2-Layers |
|-----------|---------|----------|
| GCN       | 65.26   | 69.74    |
| AdamGNN (1-level) | 88.65 | 97.91    |

Exploration of short-cuts of AdamGNN. To explore the short-cuts derived by AdamGNN, we perform another empirical analysis on one additional network, i.e., the Karate [40] club network. We choose 1-shot NC as target task, where we randomly select one sample from each class as training set, an equal number nodes as validation set and the...
rest nodes for test. Network structure, experimental results is summarised in Table 13 and two adjacency matrices are shown in Figure 7. Short-cuts derived by the first pooled super graph are depicted in green in the level 1 adjacency matrix. We find that AdamGNN outperforms GCN on 1-shot NC task with up to 40.5% performance improvements. The two figures in the right part of Figure 7 clearly demonstrate that the short-cuts derived by AdamGNN make the example node aware of far-away nodes.

Comparing aggregation mechanisms. To demonstrate the internal aggregation mechanism of AdamGNN and figure out the reason that it leads to performance improvements as shown in Section 4.2, we give a toy example of the aggregation schemas in vanilla GNNs and AdamGNN. As shown in Figure 8, 1-layer vanilla GNN can only capture limited information as presented in the left rooted tree. Nevertheless, thanks to the adaptive hierarchical structure learned by AdamGNN, target nodes can receive multi-grained semantics as well as endowed with the ability to capture information from nodes from a long-range. For instance, node \( v_a \)'s message cannot be obtained by target node with a few-layers vanilla GNN, but AdamGNN allows the target node to receive \( v_a \)'s information with 1 granularity level. The level 1’s graph allows the super node to receive a message from node \( v_a \) and pass it to the target nodes by flyback aggregator.

5 CONCLUSION

To summarise, we proposed AdamGNN, a method that integrates multi-grained semantics into node representa-

ACKNOWLEDGMENTS

This work is supported by the Luxembourg National Research Fund through grant PRIDE15/1062168/SFSpquared, and supported by Ministry of Science and Technology (MOST) of Taiwan under grants 110-2221-E-006-136-MY3, 110-2221-E-006-001, and 110-2634-F-002-051.

REFERENCES

[1] P. Barceló, E. V. Kostylev, M. Monet, J. Pérez, J. L. Reutter, and J. P. Silva, “The logical expressiveness of graph neural networks,” in Proceedings of the 2020 International Conference on Learning Representations (ICLR). OpenReview.net, 2020.
[2] D. Bo, X. Wang, C. Shi, M. Zhu, E. Lu, and P. Cui, “Structural deep clustering network,” in Proceedings of the 2020 International Conference on World Wide Web (WWW). ACM, 2020, pp. 1400–1410.
[3] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and locally connected networks on graphs,” in Proceedings of the 2014 International Conference on Learning Representations (ICLR). OpenReview.net, 2014.
[4] C. Cangea, P. Velickovic, N. Jovanovic, T. Kipf, and P. Liò, “Towards sparse hierarchical graph classifiers,” CoRR, vol. abs/1811.01287, 2018.
[5] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, “Measuring and relieving the over-smoothing problem for graph neural networks from the topological view,” in Proceedings of the 2020 AAAI Conference on Artificial Intelligence (AAAI). AAAI, 2020.
[6] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, “Simple and deep graph convolutional networks,” in Proceedings of the 2020 International Conference on Machine Learning (ICML). JMLR, 2020.
[7] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Velickovic, “Principal neighbourhood aggregation for graph nets,” in Proceedings of the 2020 Annual Conference on Neural Information Processing Systems (NeurIPS). NeurIPS, 2020.
[8] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang, “Graph random neural networks for semi-supervised learning on graphs,” in Proceedings of the 2020 Annual Conference on Neural Information Processing Systems (NeurIPS). NeurIPS, 2020.
[9] H. Gao and S. Ji, “Graph u-nets,” in Proceedings of the 2019 International Conference on Machine Learning (ICML). JMLR, 2019.
[10] W. L. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in Proceedings of the 2017 Annual Conference on Neural Information Processing Systems (NIPS). NIPS, 2017, pp. 1025–1035.
[11] S. Haykin, “Neural networks: A comprehensive foundation,” Knowledge Engineering Review, 1999.
[12] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” Int. J. Uncertain. Fuzziness Knowl. Based Syst., vol. 6, no. 2, pp. 107–116, 1998.
[13] 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,” in Proceedings of the 2020 Annual Conference on Neural Information Processing Systems (NeurIPS). NeurIPS, 2020.
[14] T. Kaler, N. Statthas, A. Ouyang, A. Iliopoulos, T. B. Schardl, C. E. Leiserson, and J. Chen, “Accelerating training and inference of graph neural networks with fast sampling and pipelining,” CoRR, 2021, abs/2110.08450.
[15] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proceedings of the 2017 International Conference on Learning Representations (ICLR). OpenReview.net, 2017.

[16] J. Lee, I. Lee, and J. Kang, “Self-attention graph pooling,” in Proceedings of the 2019 International Conference on Machine Learning (ICML). JMLR, 2019.

[17] J. Leskovec and A. Krevl, “SNAP Datasets: Stanford large network dataset collection,” [http://snap.stanford.edu/data] Jun. 2014.

[18] B. Li, X. Fan, J. Pang, and J. Zhao, “A model to slice java programs hierarchically,” Journal of Computer Science and Technology, vol. 19, no. 6, pp. 848–858, 2004.

[19] M. Li, S. Chen, Y. Zhang, and I. W. Tsang, “Graph cross networks for semi-supervised learning,” in Proceedings of the 2020 Annual Conference on Neural Information Processing Systems (NeurIPS). NeurIPS, 2020.

[20] Q. Li, Z. Han, and X. Wu, “Deeper insights into graph convolutional networks for semi-supervised learning,” in Proceedings of the 2018 AAAI Conference on Artificial Intelligence (AAAI). AAAI, 2018, pp. 3538–3545.

[21] H. Liu, Y. Yang, and X. Wang, “Overcoming catastrophic forgetting in graph neural networks,” in Proceedings of the 2021 AAAI Conference on Artificial Intelligence (AAAI). AAAI, 2021.

[22] J. You, R. Ying, and J. Leskovec, “Hierarchical graph representation learning with differentiable pooling,” in Proceedings of the 2018 Annual Conference on Neural Information Processing Systems (NeurIPS). NeurIPS, 2018, pp. 4805–4815.

[23] J. You, R. Ying, and J. Leskovec, “Position-aware graph neural networks,” in Proceedings of the 2019 International Conference on Machine Learning (ICML). JMLR, 2019.

[24] H. Yuan and S. Ji, “Structpool: Structured graph pooling via conditional random fields,” in Proceedings of the 2020 International Conference on Learning Representations (ICLR). OpenReview.net, 2020.

[25] W. W. Zachary, “An information flow model for conflict and fission in small groups,” Journal of Anthropological Research, 1977.

[26] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang, “GCC: graph contrastive coding for graph neural network pre-training,” in Proceedings of the 2020 ACM Conference on Knowledge Discovery and Data Mining (KDD). ACM, 2020, pp. 723–731.

[27] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, “Provably powerful graph networks,” in Proceedings of the 2019 Annual Conference on Neural Information Processing Systems (NeurIPS). NeurIPS, 2019, pp. 2153–2164.

[28] J. Shi, M. Xu, Z. Zhu, W. Zhang, M. Zhang, and J. Tang, “Graphf: a flow-based autoregressive model for molecular graph generation,” in Proceedings of the 2021 International Conference on Learning Representations (ICLR). OpenReview.net, 2021.

[29] S. Teng, “Scalable algorithms for data and network analysis,” Foundations and Trends in Theoretical Computer Science, vol. 12, no. 1-2, pp. 1–274, 2016.

[30] V. Thost and J. Chen, “Directed acyclic graph neural networks,” in Proceedings of the 2021 International Conference on Learning Representations (ICLR). OpenReview.net, 2021.

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in Proceedings of the 2017 Annual Conference on Neural Information Processing Systems (NIPS). NIPS, 2017, pp. 5998–6008.

[32] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” in Proceedings of the 2018 International Conference on Learning Representations (ICLR). OpenReview.net, 2018.

[33] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks,” IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 1, pp. 4–24, 2020.

[34] J. Xie, R. B. Girshick, and A. Farhadi, “Unsupervised deep embedding for clustering analysis,” in Proceedings of the 2016 International Conference on Machine Learning (ICML). JMLR, 2016, pp. 478–487.

[35] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” in Proceedings of the 2019 International Conference on Machine Learning (ICML). JMLR, 2019.

[36] Z. Yang, M. Ding, C. Zhou, H. Yang, J. Zhou, and J. Tang, “Understanding negative sampling in graph representation learning,” in Proceedings of the 2020 ACM Conference on Knowledge Discovery and Data Mining (KDD). ACM, 2020, pp. 1666–1676.

[37] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, “Hierarchical graph representation learning with differentiable pooling,” in Proceedings of the 2018 Annual Conference on Neural Information Processing Systems (NeurIPS). NeurIPS, 2018, pp. 4805–4815.

Zhiquang Zhong is currently pursuing his Ph.D. degree at Faculty of Science, Technology and Medicine, University of Luxembourg, Luxembourg. His principal research interest is in graph neural networks and its applications in urban areas.

Cheng-Te Li is an Associate Professor at Institute of Data Science and Department of Statistics, National Cheng Kung University (NCKU), Tainan, Taiwan. He received my Ph.D. degree (2013) from Graduate Institute of Networking and Multimedia, National Taiwan University. Before joining NCKU, He was an Assistant Research Fellow (2014-2016) at CITI, Academia Sinica. Dr. Li’s research targets at Machine Learning, Deep Learning, Data Mining, Social Networks and Social Media Analysis, Recommender Systems, and Natural Language Processing. He has a number of papers published at top conferences, including KDD, WWW, ICDM, CIKM, SIGIR, IJCAI, ACL, ICDM, and ACM-MM. He leads Networked Artificial Intelligence Laboratory (NetAI Lab) at NCKU.

Jun Pang is currently a Senior Researcher at Faculty of Science, Technology and Medicine & Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg. He received his Ph.D. degree from Vrije Universiteit Amsterdam in 2004. His research interests include formal methods, social media mining, computational systems biology, security and privacy.