On Size-Oriented Long-Tailed Graph Classification of Graph Neural Networks

Zemin Liu[1*], Qiheng Mao[2,4*], Chenghao Liu[3‡], Yuan Fang[1], Jianling Sun[2,4‡]
[1]Singapore Management University; [2]Zhejiang University;
[3]Salesforce Research Asia; [4]Alibaba-Zhejiang University Joint Institute of Frontier Technologies
{zmliu, yfang}@smu.edu.sg, {22021184, sunjl}@zju.edu.cn, chenghao.liu@salesforce.com

ABSTRACT

The prevalence of graph structures attracts a surge of investigation on graph data, enabling several downstream tasks such as multi-graph classification. However, in the multi-graph setting, graphs usually follow a long-tailed distribution in terms of their sizes, *i.e.*, the number of nodes. In particular, a large fraction of tail graphs usually have small sizes. Though recent graph neural networks (GNNs) can learn powerful graph-level representations, they treat the graphs uniformly and marginalize the tail graphs which suffer from the lack of distinguishable structures, resulting in inferior performance on tail graphs. To alleviate this concern, in this paper we propose a novel graph neural network named SOLT-GNN, to close the representational gap between the head and tail graphs from the perspective of knowledge transfer. In particular, SOLT-GNN capitalizes on the *co-occurrence substructures exploitation* to extract the transferable patterns from head graphs. Furthermore, a novel *relevance prediction function* is proposed to memorize the pattern relevance derived from head graphs, in order to predict the complements for tail graphs to attain more comprehensive structures for enrichment. We conduct extensive experiments on five benchmark datasets, and demonstrate that our proposed model can outperform the state-of-the-art baselines.

CCS CONCEPTS

• **Computing methodologies** → **Learning latent representations**; • **Information systems** → *Data mining*.

KEYWORDS

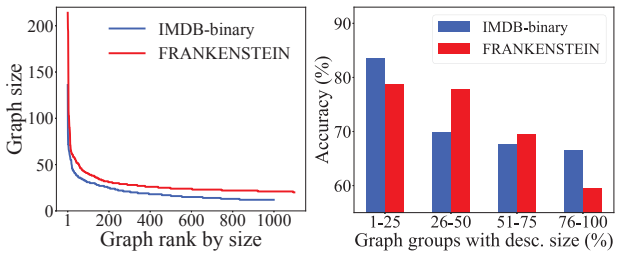
Size-oriented long-tailed distribution, graph neural networks, knowledge transfer

ACM Reference Format:
Zemin Liu[1*], Qiheng Mao[2,4*], Chenghao Liu[3‡], Yuan Fang[1], Jianling Sun[2,4‡]. 2022. On Size-Oriented Long-Tailed Graph Classification of Graph Neural Networks. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3485447.3512197

*Co-first authors with equal contribution.
‡Corresponding authors.

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512197



(a) Long-tailed graph distribution (b) Graph classification performance

Figure 1: Illustration of long-tailed distribution.

1 INTRODUCTION

Graph structures, as a typical form of Web data, have attracted a surge of investigation due to their prevalence in society. In particular, analysis on graphs benefits a wide range of tasks on the Web, such as search, recommendation, Web evolution, *etc.* Consequently, many efforts have been devoted to the community of graph analysis. In particular, graph representation learning [3] arises as an effective tool to map nodes into low-dimensional vectors by preserving the graph structures, opening a great opportunity for graph analysis. Recently, more attention has been shifted to graph neural networks (GNNs) [45], which capitalize on the key operation of neighborhood aggregation to recursively aggregate messages from neighboring nodes to form the node representations, thus preserving both the structure and content information simultaneously. These prior attempts accordingly lay the foundation for graph-level representation learning [46, 50], which essentially summarizes the underlying patterns of the entire graph, enabling several downstream tasks such as multi-graph classification [26, 51].

Problem. In the setting of multi-graph classification, the number of nodes in each graph, *i.e.*, the graph size, usually follows a long-tailed distribution across graphs. For example, as shown in Fig. 1(a), the graph sizes in two benchmark datasets (refer to Appendix B for more details) exhibit the power-law characteristic. In particular, a few *head* graphs occupy a handful of the largest graph sizes, while a significant fraction of *tail* graphs usually have small sizes. However, GNNs for graph-level representation learning depend heavily on the abundance of structural information in a graph. As shown in Fig. 1(b), applying GIN [46] for graph classification, the graphs with larger size usually achieve higher accuracy, while the performance gradually drops as the graph size decreases. While GNNs are capable of drawing expressive and discriminative representations of the head graphs hinging on their sufficient structures, they usually treat the graphs uniformly and marginalize the tail graphs which suffer from the lack of distinguishable structures. Unsurprisingly, this

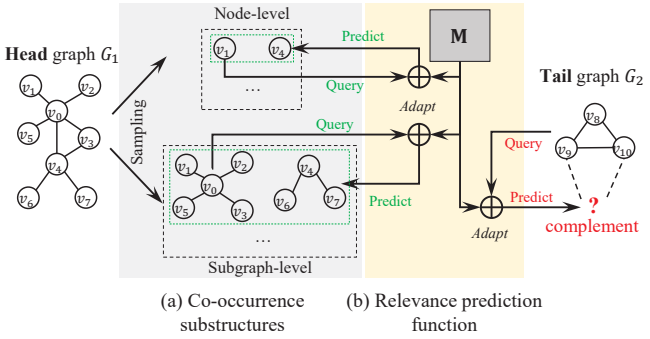


Figure 2: Knowledge transfer from head to tail graphs.

long-tailed distribution gives rise to the performance differentiation between head and tail graphs. The inferior performance of tail graphs further limits the overall performance. To bridge this gap, in this paper, we investigate the significant yet unexplored problem of size-oriented long-tailed graph classification, by particularly improving the performance of tail graphs.

Prior work. Recent studies [8, 32, 46, 50] on graph-level representation learning generally concentrate on improving the GNN architectures to yield more expressive representations. Despite the effectiveness, they usually treat all graphs uniformly and neglect to particularly improve the tail graphs which suffer from scarce structures. Several recent studies investigate the node-level long-tailed distribution phenomenon on graphs, such as the long-tailed node degrees [24, 25] and node classes [37, 52]. However, these are distinct problems and their approaches cannot be directly applied to the multi-graph setting. Several related studies investigate the effect of graph size on graph learning, such as difficulty estimation by graph size for curriculum learning [43], embedding dimension modulation w.r.t. graph size for representation learning [18], and model generalization from small to large graphs [48]. However, none of them is particularly designed to improve the performance of tail graphs, an especially challenging group due to the structural limitation.

Challenges and present work. Compared to the head, tail graphs usually lack inner structures due to their small sizes, which further restrict the expressiveness of their learnt representations. To promote the representational capacity of GNNs especially toward tail graphs, a possible solution is through the lens of knowledge transfer [19, 24, 49] from head to tail graphs to enrich the latter, by deriving more comprehensive graph structures on tail graphs for a better representation. However, this non-trivial solution presents us with two main challenges. First, *how to identify and exploit the transferable patterns on head graphs for tail graph enrichment?* Second, *how to transfer the necessary patterns for tail graphs in consideration of their individuality?*

To address these challenges, in this paper we propose a novel graph neural network named SOLT-GNN, toward Size-Oriented Long-Tailed graph representation learning especially for the promotion of tail graphs, in order to narrow the gap between head and tail graphs. In particular, SOLT-GNN tries to exploit the transferable patterns on the structure-abundant head graphs for the sake of enriching the structure-scarce tail graphs for more expressive graph representations, which can further boost the overall performance.

Generally, patterns on graphs can be associated together to form some higher-level groups. For example, on a protein network, the combination of an amidogen and a carboxyl can form some kind of amino acid. This association intrinsically reflects the relevance proximity between the graph patterns, which motivates us to resort to the *co-occurrence of substructures* for transferable patterns exploitation. Therefore, to deal with the first challenge, we distill the pattern relevance by exploiting the substructure co-occurrence on the structure-abundant head graphs, especially from two perspectives, *i.e.*, node- and subgraph-levels, as shown in Fig. 2(a). On the one hand, nodes are atomic elements on graph, and their representations fundamentally reflect the miniature of local patterns. Therefore, the co-occurrence of nodes on a graph (*e.g.*, v_1 and v_4 in Fig. 2(a)) essentially reveals the relevance between the fine local patterns. On the other hand, a sampled subgraph intrinsically preserves the partial structure information of the entire graph, which can be complemented by the rest of the graph to form a holistic overview of the whole graph. In light of this, we further rely on the co-occurrence of subgraphs to derive the complementary characteristic for pattern relevance exploitation.

To address the second challenge, as shown in Fig. 2(b), we devise a novel *relevance prediction function* to preserve the pattern relevance exploited on head graphs, and further predict the relevant patterns to enrich the tail graphs. In particular, the relevance prediction function capitalizes on a *pattern memory* $\mathbf{M}$ to memorize the underlying patterns across the graphs, which are distilled from the mined co-occurrence substructures shown in Fig. 2(a). More precisely, given the sampled co-occurrence substructures, the relevance prediction function is capable of learning to predict the relevant patterns for a given query by adapting the shared pattern memory toward the query, which can be further utilized to predict the relevant patterns as the complement to a given tail graph. As a result, the relevance prediction function can bridge the structure-abundant head and structure-scarce tail graphs for knowledge transfer, to subsequently boost the representational expressiveness of the latter.

Contributions. To the best of our knowledge, this is the first GNN model to investigate the problem of size-oriented long-tailed graph classification. In summary, our contributions are three-fold. (1) We identify the size-oriented long-tailed phenomenon and its impact on the graph classification performance, and investigate the problem through the lens of knowledge transfer. (2) We propose a novel graph neural network SOLT-GNN to close the gap between head and tail graphs for long-tailed graph classification. (3) Extensive experiments on five benchmark datasets demonstrate that our proposed model can outperform the state-of-the-art baselines.

2 RELATED WORK

Graph representation learning. The development of graph embedding approaches [11, 30, 38] opens great opportunities for graph analysis, which aim to map nodes or substructures into a low-dimensional space in which the connecting structures on the graph can be preserved. Recently, graph neural networks (GNNs) [13, 15, 21, 23, 41, 46] emerge as the state-of-the-art approaches for graph

representation learning, which usually capitalize on the key operation of neighborhood aggregation to recursively pass and receive messages from the neighbors for node representation learning, thus both the structure and content information can be preserved.

In addition, graph-level representation learning [17, 26, 46, 50, 51] focuses on the overall graph structure, thus they usually require an extra step—summarizing the local structure representations to generate the graph-level representation. Early studies usually resort to graph kernel methods [16, 35, 42, 47], but the heuristic characteristic and computational expense hinder their development on more complex networks. Recently, GNN-based graph pooling approaches [27] boost the investigation of graph-level representation, which can be further categorized into two aspects, *i.e.*, global pooling and hierarchical pooling. Global pooling [8, 10, 46, 51] is also known as the READOUT function, which utilizes a pooling layer to summarize all the node representations by connected layer [10], direct pooling [46], sorted pooling [51] or more [1, 8]. Hierarchical pooling [9, 17, 26, 32, 50] approaches try to gradually group nodes into clusters and coarsen the graph recursively, thus an overview of the whole graph structure can be obtained at last. However, these graph-level representation approaches are generally designed for improving the overall graph representations, yet still marginalizing the structure-scarce tail graphs.

Long-tailed problems on graph. Recently, increasing attention from diverse research areas has been paid to the problem of long-tailed distribution, such as recommendation [5, 20, 34, 49] and imbalanced classification [14, 19, 39] in computer vision. Furthermore, several recent studies have been devoted to address the long-tailed problems on graph. Approaches meta-tail2vec [25] and Tail-GNN [24] concentrate on the long-tailed distribution of node degrees. In addition, several studies [6, 22, 31, 37, 52, 53] are proposed to address the imbalanced class issue toward node classification. However, these approaches mainly address the long-tailed distribution issue lying in the node-level, which is distinct from our size-oriented long-tailed graph classification.

Investigation on graph size effect. A few recent studies consider the graph size effect on graph neural networks. CurGraph [43] tries to investigate the curriculum learning on graph neural networks, by regarding the graph size as a heuristic metric to evaluate the difficulty of diverse graphs. MxGNN [18] considers the relationship between graph size and the dimension size of GNN embeddings, by grouping graphs into several groups w.r.t. their sizes. Another graph size related problem is size generalization [48] on GNNs, which aims to train GNNs on small graphs and generalize the obtained GNN models onto large graphs for generalization ability investigation. While they distinguish graphs based on their sizes, they are not particularly devised to enhance the representational capacity of GNNs toward structure-scarce tail graphs.

3 PRELIMINARIES

3.1 Problem Formulation

Graph. A graph is denoted by $G = \{V, E, \mathbf{X}\}$, where V is the set of nodes, E is the set of edges, $\mathbf{X} \in \mathbb{R}^{|V| \times d_X}$ is the node feature matrix with $\mathbf{x}_v \in \mathbb{R}^{d_X}$ denoting the feature vector of node $v \in V$.

The problem. For multi-graph classification, given a set of graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$ associated with the corresponding graph labels $Y = \{y_1, y_2, \ldots, y_N\}$, where $G_i = \{V_i, E_i, \mathbf{X}_i\}$ is a graph with $i \in \{1, 2, \ldots, N\}$, graph-level representation learning aims to learn a mapping function $f : \mathcal{G} \to \mathbb{R}^d$ to map each graph $G_i \in \mathcal{G}$ into a low-dimensional vector $\mathbf{h}_{G_i} \in \mathbb{R}^d$, thus a classifier $\ell : \mathcal{G} \to Y$ can be employed to classify graph $G_i \in \mathcal{G}$ into its corresponding group $\ell(G_i) = y_i \in Y$ based on the representation $\mathbf{h}_{G_i}$.

In the multi-graph setting, the graph size, *i.e.*, the number of nodes $|V_i|$ in each graph G_i, usually follows a long-tailed distribution. For ease of discussion, we rank all graphs by their sizes in a descent order, *i.e.*, $|V_1| \geqslant |V_2| \geqslant \ldots \geqslant |V_N|$. In particular, the first K graphs are called *head* graphs, *i.e.*, $\mathcal{G}_{\text{head}} = \{G_1, G_2, \ldots, G_K\} \subset \mathcal{G}$; while the rest graphs with small sizes are called *tail* graphs, *i.e.*, $\mathcal{G}_{\text{tail}} = \{G_{K+1}, G_{K+2}, \ldots, G_N\} \subset \mathcal{G}$. It is obvious that $\mathcal{G} = \mathcal{G}_{\text{head}} \cup \mathcal{G}_{\text{tail}}$ and $\mathcal{G}_{\text{head}} \cap \mathcal{G}_{\text{tail}} = \emptyset$. Note that, K can be predetermined based on the specificality of each dataset. Let k denote the smallest size of head graphs, *i.e.*, $k = |V_K|$. Given the above setup, we aim to learn a powerful GNN which can achieve satisfactory graph classification performance for both head and tail graphs, especially the latter.

3.2 Graph Neural Networks

Graph neural networks (GNNs) [45] typically depend on the key operation of layer-wise neighborhood aggregation to recursively pass and transform messages from the neighboring nodes to form the representation of the target node. Formally, let $\phi_g(\cdot; \theta_g)$ denote a GNN architecture parameterized by θ_g. In the l-th layer, the representation of node v, *i.e.*, $\mathbf{h}_v^l \in \mathbb{R}^{d_l}$, can be calculated by

$$\mathbf{h}_v^l = \textsc{Aggr}(\mathbf{h}_v^{l-1}, \{\mathbf{h}_i^{l-1} : i \in \mathcal{N}_v\}; \theta_g^l), \tag{1}$$

where $\mathcal{N}_v$ is the neighbors set of node v, and $\textsc{Aggr}(\cdot; \theta_g^l)$ is the neighborhood aggregation function parameterized by θ_g^l in layer l. Given a total of L layers, $\theta_g = \{\theta_g^1, \theta_g^2, \ldots, \theta_g^L\}$ denotes all the parameters set in the GNN model. Note that, in the first layer, the node feature vector $\mathbf{x}_v$ is utilized as the initial representation $\mathbf{h}_v^0$, *i.e.*, $\mathbf{h}_v^0 = \mathbf{x}_v$. For simplicity, we directly use $\mathbf{h}_v \in \mathbb{R}^{d_L}$ to denote the output representation of node v. In addition, different GNN architectures may differ in the neighborhood aggregation function, *e.g.*, mean-pooling aggregation in GCN [15], attention-based aggregation in GAT [41]. In particular, our SOLT-GNN is agnostic to the base GNN models, and we aim to make it flexible to most neighborhood aggregation based GNNs.

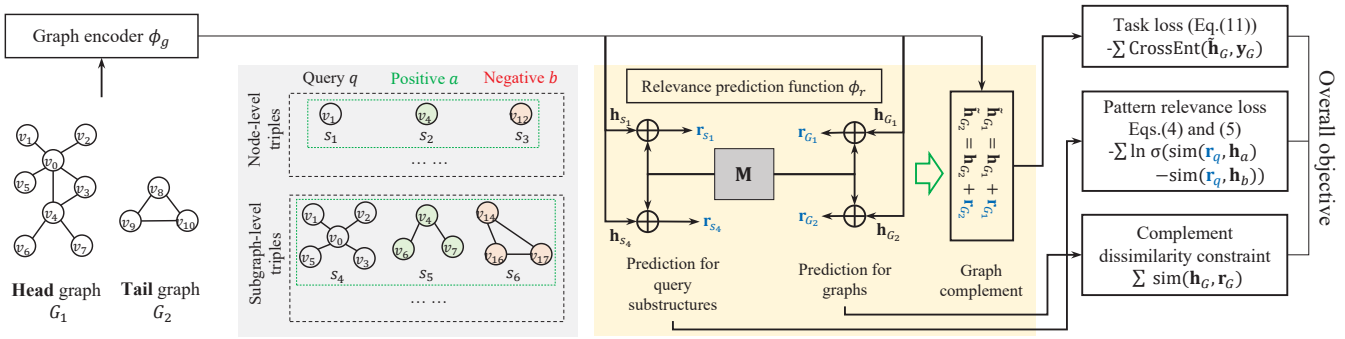
Given the node representations on a graph G_i, an additional summarizing function, *i.e.*, the READOUT function, can be utilized to aggregate the node representations to form the graph-level representation $\mathbf{h}_{G_i} \in \mathbb{R}^d$ as follows,

$$\mathbf{h}_{G_i} = \text{READOUT}(\{\mathbf{h}_v : v \in V_i\}), \tag{2}$$

where $\text{READOUT}(\cdot)$ is a READOUT function, which is usually permutation invariant [50, 51]. We simply apply SUM-pooling in our implementation (thus here dimension $d = d_L$), although more advanced alternatives can also be utilized [46, 50].

4 THE PROPOSED MODEL: SOLT-GNN

In this section, we present the concrete description of SOLT-GNN for size-oriented long-tailed graph classification through the lens of



(a) Toy networks (b) Co-occurrence substructures exploitation (c) Pattern relevance prediction (d) Overall objective

Figure 3: Overall framework of SOLT-GNN. Note that, substructures s_3 and s_6 in (b) are sampled from another random graph to serve as the negative instances.

knowledge transfer from head to tail graphs, especially to promote the performance of the latter. We illustrate the overall framework of SOLT-GNN in Fig. 3. Given a set of graphs including the head and tail, we first exploit the co-occurrence substructures on head graphs in Fig. 3(b) from two aspects, *i.e.*, node- and subgraph-levels (Sect. 4.1). Afterwards, in Fig. 3(c) we rely on a relevance prediction function to extract the pattern relevance hinging on the mined co-occurrence substructures, and predict the relevant patterns for graphs to serve as the graph complement (Sect. 4.2). Finally, in Fig. 3(d), the complemented graph representations are utilized to form the graph classification loss, by incorporating several auxiliary constraints to enhance the model training (Sect. 4.3). In the following part, we will elaborate the details of each part.

Relevance prediction function. The GNN architectures can draw expressive representations from head graphs by virtue of their abundant structures. On the contrary, the lack of structural information on tail graphs limits their representational capacity, exposing a bottleneck of ordinary GNNs on representation learning for tail graphs. To address this issue, we resort to knowledge transfer from head to tail graphs to enrich the latter for more comprehensive graph representations. In particular, the abundant structures on head graphs, open an opportunity for us to exploit the transferable patterns. As discussed in Sect. 1, the graph patterns usually associate with each other to form more advanced structural groups. In addition, patterns usually manifest as substructures on a graph, such as the amidogen on a protein network. In light of this, we exploit the co-occurrence of substructures, which instantiate the abstract patterns, to derive the relevance between patterns, thus proper structural information can be further predicted on tail graphs to complement their scarce structures.

In fulfillment of relevance distillation, we are required to seek a relevance prediction function $\phi_r(\cdot;\theta_r)$ parameterized by θ_r, to predict the relevant pattern for a query q as follows,

$$\mathbf{r}_q = \phi_r(\mathbf{h}_q;\theta_r), \tag{3}$$

where $\mathbf{h}_q \in \mathbb{R}^d$ is the representation of query q, and $\mathbf{r}_q \in \mathbb{R}^d$ is the predicted relevant pattern representation of q.

In particular, relevance prediction function provides us with the ability of preserving the relevance between patterns from head graphs, thus it further enables the relevant pattern prediction for the structure-scarce tail graphs. Therefore, it can bridge the head and tail graphs for knowledge transfer. Next, we will illustrate the co-occurrence substructures exploitation on head graphs, which lay the foundation for pattern relevance distillation.

4.1 Co-occurrence Substructures Exploitation

In essence, pattern is an abstract concept and can be arbitrary structures on graphs. To capture the relevance from different perspectives, we embody the patterns into two levels of substructures, *i.e.*, node- and subgraph-levels, as illustrated before. The co-occurrence of nodes on a graph essentially reveals the relevance between fine local structures, while the co-occurrence of subgraphs reflects the relevance between partial graph-level representation and its complementary representation. On account of this, the dual-level exploitation enables us to discern relevance for both the fine- and coarse-grained structures, respectively.

Node-level co-occurrence. Intrinsically, the co-occurrence of nodes on a graph reflects the relevance between their contextual structures they reside in, while the relevance would not remain for nodes distributing on different graphs. This motivates us to compose the positive and negative pairs for the training of $\phi_r(\cdot;\theta_r)$ in a *contrastive* manner. Formally, given a head graph $G_i = \{V_i, E_i, \mathbf{X}_i\} \in \mathcal{G}_{\text{head}}$, we randomly sample two nodes $v, u \in V_i$ to form the positive pair. Meanwhile, we further sample a node u' from another random graph $G_j \in \mathcal{G}_{\text{head}}$ as a negative node. Thus, we can instantiate a triplet (q, a, b) as (v, u, u'), in which (q, a) is a positive pair and (q, b) is a negative pair. For example, as shown in Fig. 3(b), substructures s_1, s_2 and s_3 (*i.e.*, nodes v_1, v_4 and v_{12}) form a node-level triplet. Finally, we assemble all the node-level triplets to form set $\mathcal{T}_{\text{node}}$.

Subgraph-level co-occurrence. For subgraph-level co-occurrence, given a head graph $G_i = \{V_i, E_i, \mathbf{X}_i\} \in \mathcal{G}_{\text{head}}$ as well as a predefined size t (usually smaller than k), we start from a random node $v \in V_i$ to conduct a BFS (Breadth First Search) based sampling to derive a connected subgraph $G_i^{\text{sub}} = \{V_i^{\text{sub}}, E_i^{\text{sub}}, \mathbf{X}_i^{\text{sub}}\}$ with size $|V_i^{\text{sub}}| = t$, to serve as the query. Subsequently, we regard the remaining structure on graph G_i, *i.e.*, $G_i^{\text{rem}} = \{V_i^{\text{rem}}, E_i^{\text{rem}}, \mathbf{X}_i^{\text{rem}}\}$ by subtracting G_i^{sub} from G_i, as the positive sample of G_i^{sub}. Note that, the combination of these two substructures can form a holistic

structure, *i.e.*, the graph G_i. Furthermore, given another random head graph G_j, we randomly sample a set of nodes to form subgraph G_j^{neg} which has the same size with G_i^{rem} and can be unconnected, to serve as the negative instance. Finally, a subgraph-level triplet (q, a, b) can be instantiated as $(G_i^{\text{sub}}, G_i^{\text{rem}}, G_j^{\text{neg}})$. For example, as shown in Fig. 3(b), substructures s_4, s_5 and s_6 form a subgraph-level triplet. We assemble all the subgraph-level triplets into set $\mathcal{T}_{\text{subg}}$. In particular, due to the limited structural information, subgraph G_i^{sub} can be regarded as a fake structure-scarce tail graph, which requires a complement to form a holistic structure.

Note that, we can conduct the co-occurrence exploitation offline to save time. The exploited triplets can reflect the pattern relevance from two views, *i.e.*, node- and subgraph-levels, which will be employed to train the relevance prediction function $\phi_r(\cdot;\theta_r)$.

Pattern relevance loss. We rely on the dual-level co-occurrence triplets to guide the training of relevance prediction function $\phi_r(\cdot;\theta_r)$. In particular, given a triplet (q, a, b) associated with the corresponding representation triplet $(\mathbf{h}_q, \mathbf{h}_a, \mathbf{h}_b)$, we predict q's relevant pattern representation $\mathbf{r}_q$ by Eq. (3), and drive $\mathbf{r}_q$ close to the positive instance $\mathbf{h}_a$, while away from negative instance $\mathbf{h}_b$. Note that, to calculate the representations for the subgraphs existing in subgraph-level triplets, we directly modify the READOUT function in Eq. (2) to merely aggregate the node representations within each subgraph. We resolve the pattern relevance objective into two parts, for node- and subgraph-levels, respectively, as they may contribute differently to the optimization of relevance prediction function, as follows.

$$\mathcal{L}_{\text{rel}}^{\text{node}} = -\sum_{(q,a,b)\in\mathcal{T}_{\text{node}}} \ln \sigma(\text{sim}(\mathbf{r}_q, \mathbf{h}_a) - \text{sim}(\mathbf{r}_q, \mathbf{h}_b)), \quad (4)$$

$$\mathcal{L}_{\text{rel}}^{\text{subg}} = -\sum_{(q,a,b)\in\mathcal{T}_{\text{subg}}} \ln \sigma(\text{sim}(\mathbf{r}_q, \mathbf{h}_a) - \text{sim}(\mathbf{r}_q, \mathbf{h}_b)). \quad (5)$$

Here $\text{sim}(\cdot,\cdot)$ is a similarity function, and we employ cosine similarity in our implementation; and $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function.

By the optimization w.r.t. the dual-level co-occurrence triplets, relevance prediction function $\phi_r(\cdot;\theta_r)$ can be driven to learn how to predict the relevant pattern representation for a query. We will elaborate the details of this function in Sect. 4.2.

4.2 Realizing Pattern Relevance Prediction

The dual-level co-occurrence triplets exploited on head graphs empower the optimization of relevance prediction function $\phi_r(\cdot;\theta_r)$. In this section, we concretely illustrate this function.

Relevance prediction function. Generally, in a particular domain, typical patterns are commonly shared across graphs. For example, patterns such as hydroxy groups and amidogens are usually shared across protein networks. On the other hand, each graph usually possesses particular patterns forming its individual frame. For example, a protein molecule may have a hydroxy group, while another may has a phenyl group forming its own holistic function.

To memorize the typical patterns across graphs while holding the capacity to adapt to each specific query for prediction, we devise a novel module named *pattern memory* for pattern memorization. In particular, pattern memory $\mathbf{M} \in \mathbb{R}^{d_m \times d_m}$ is a globally shared learnable parameter matrix to record the typical patterns across graphs. Meanwhile, it can also be adapted to each query q (*e.g.*, a tail graph) w.r.t. its representation $\mathbf{h}_q$ to predict its relevant pattern representation. Again, we aim to extract the abstract patterns based on the co-occurrence triples $\mathcal{T}_{\text{node}}$ and $\mathcal{T}_{\text{subg}}$ exploited from head graphs and preserve them into the pattern memory, so that it can be further utilized to predict the relevant patterns for a query (*e.g.*, a tail graph) to serve as its structural complement. Thus, the structural knowledge can be transferred from head to tail graphs.

Formally, given the representation of a query q, *i.e.*, $\mathbf{h}_q \in \mathbb{R}^d$, we consider the scaling and shifting transformation [21, 29] to modulate the pattern memory $\mathbf{M}$ w.r.t. $\mathbf{h}_q$ into an adapted version,

$$\mathbf{M}_q = (\mathbf{A}_q + \mathbf{1}) \odot \mathbf{M} + \mathbf{B}_q, \quad (6)$$

where $\mathbf{M}_q \in \mathbb{R}^{d_m \times d_m}$ is the adapted pattern memory w.r.t. query q, $\odot$ denotes element-wise product, variables $\mathbf{A}_q$ and $\mathbf{B}_q \in \mathbb{R}^{d_m \times d_m}$ are modulating factors to scale and shift pattern memory $\mathbf{M}$ w.r.t. the query q, and $\mathbf{1}$ is a vector filled with ones to ensure the scaling factor surrounding around ones. Note that, here q can be a node in Eq. (4), a subgraph in Eq. (5), or a graph which needs structural complement. In addition, $\mathbf{A}_q$ and $\mathbf{B}_q$ are not learnable parameter matrices, but are generated by a secondary neural network [12] as,

$$\mathbf{A}_q = \text{LeakyReLU}(\mathbf{w}_A \mathbf{h}_q^\top \mathbf{U}_A), \quad (7)$$

$$\mathbf{B}_q = \text{LeakyReLU}(\mathbf{w}_B \mathbf{h}_q^\top \mathbf{U}_B), \quad (8)$$

where $\mathbf{w}_A$ and $\mathbf{w}_B \in \mathbb{R}^{d_m}$ are learnable parameter vectors, and $\mathbf{U}_A$ and $\mathbf{U}_B \in \mathbb{R}^{d \times d_m}$ are learnable parameter matrices.

The relevant pattern representation of query q, *i.e.*, $\mathbf{r}_q$, can be further distilled from the modulated pattern memory $\mathbf{M}_q$ as

$$\mathbf{r}_q = \phi_r(\mathbf{h}_q;\theta_r) = (\text{LeakyReLU}(\mathbf{M}_q \mathbf{W}_r))^\top \mathbf{w}_r, \quad (9)$$

where $\mathbf{w}_r \in \mathbb{R}^{d_m}$ and $\mathbf{W}_r \in \mathbb{R}^{d_m \times d}$ are learnable parameter vector and matrix, respectively. Here, $\theta_r = \{\mathbf{M}, \mathbf{W}_r, \mathbf{U}_A, \mathbf{U}_B, \mathbf{w}_A, \mathbf{w}_B, \mathbf{w}_r\}$ is the parameters set in the relevance prediction function.

Given a query q, relevance prediction function $\phi_r(\cdot;\theta_r)$ is capable of inferring the relevant pattern representation for q. Consequently, given a tail graph $G_i \in \mathcal{G}_{\text{tail}}$ associated with the representation $\mathbf{h}_{G_i}$, we can capitalize on this function to infer its relevant pattern representation by $\mathbf{r}_{G_i} = \phi_r(\mathbf{h}_{G_i};\theta_r)$. In particular, as $\mathbf{r}_{G_i}$ reflects the relevant patterns which may co-occur with the query tail graph G_i with high probability, it can act as the complement to enrich the structure-scarce tail graph G_i for a more comprehensive structure.

Enriched graph representation. For each graph G_i, we enrich its original graph-level representation $\mathbf{h}_{G_i}$ by incorporating the predicted relevant pattern representation $\mathbf{r}_{G_i}$ (*i.e.*, the complement) generated by relevance prediction function $\phi_r(\cdot;\theta_r)$ as follows,

$$\tilde{\mathbf{h}}_{G_i} = \mathbf{h}_{G_i} + \phi_r(\mathbf{h}_{G_i};\theta_r) = \mathbf{h}_{G_i} + \mathbf{r}_{G_i}. \quad (10)$$

We directly utilize an add operation to combine the two representations, while it is also possible to employ more advanced neural networks to make it learnable. The enriched graph representation $\tilde{\mathbf{h}}_{G_i}$ can be further employed for graph classification.

4.3 Training Constraints and Objective

While the graph representations can be fed into the task loss for end-to-end learning, we also incorporate several auxiliary constraints into the overall objective to facilitate the model training.

Task loss. The enriched graph representations can be employed to minimize the graph classification loss. In particular, as our goal is to improve the graph classification especially for the tail graphs, we resolve the task loss into two parts, *i.e.*, task loss for head and tail graphs, since they may contribute differently to the model optimization for promoting the performance of tail graphs. Formally, given a set of training graphs $\mathcal{G}_{\text{tr}}$ associated with the graph labels Y_{tr}, the task loss $\mathcal{L}_{\text{task}}$ can be formulated as

$$\mathcal{L}_{\text{task}} = \alpha \cdot \sum_{G_i \in \mathcal{G}_{\text{tr}} \cap \mathcal{G}_{\text{head}}} \textsc{CrossEnt}(\tilde{\mathbf{h}}_{G_i}, \mathbf{y}_i) + (1-\alpha) \cdot \sum_{G_i \in \mathcal{G}_{\text{tr}} \cap \mathcal{G}_{\text{tail}}} \textsc{CrossEnt}(\tilde{\mathbf{h}}_{G_i}, \mathbf{y}_i), \quad (11)$$

where $\mathbf{y}_i$ is the one-hot encoding of the corresponding label $y_i \in Y_{\text{tr}}$, $\textsc{CrossEnt}(\cdot,\cdot)$ is the cross-entropy function, and α is a hyper-parameter to modulate the weight of loss on head and tail graphs. Note that, in Eq. (11) we incorporate the inferred complement for both head and tail graphs for model training, and this would drive the model to learn more expressive representations for tail graphs which need the complement. For inference on testing set, we only incorporate the predicted complement for tail graphs to form their final representations; while for head graphs, we directly resort to the representations calculated by the base GNN model for usage, since their abundant structures are already representative enough.

Constraint for dissimilarity of inferred complement. Given a query graph G_i, the relevance prediction function $\phi_r(\cdot;\theta_r)$ can infer the relevant patterns which may co-occur with the structures on G_i with a high probability. In particular, the predicted complement should represent the highly relevant patterns which may co-occur with G_i, but not those similar to G_i. Thus, to stress more on the relevance yet less on similarity between them, we import an additional constraint to make them dissimilar, as follows.

$$\mathcal{L}_{\text{dis}} = \sum_{G_i \in \mathcal{G}_{\text{tr}}} \text{sim}(\mathbf{h}_{G_i}, \mathbf{r}_{G_i}). \quad (12)$$

Overall objective. By incorporating all the loss functions and constraints, we formulate the overall objective as follows,

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda \cdot \mathcal{L}_{\text{dis}} + \mu_1 \cdot \mathcal{L}_{\text{rel}}^{\text{node}} + \mu_2 \cdot \mathcal{L}_{\text{rel}}^{\text{subg}} + \Omega(\Theta), \quad (13)$$

where $\Theta = \theta_g \cup \theta_r$ is the parameters set, $\Omega(\cdot)$ is the L_2 regularization function, and λ, μ_1 and μ_2 are hyperparameters to modulate the weight of each part. We further present the algorithm for model training as well as the complexity analysis in Appendix A.

5 EXPERIMENTS

In this section, we conduct extensive experiments on tail graph classification to evaluate the performance of the proposed SOLT-GNN, and further give detailed model analysis from several aspects.

5.1 Experimental Setups

Datasets. We employ a total of five benchmark datasets, including four bioinformatics datasets and one social network dataset. The bioinformatics datasets include *PTC* [40], *PROTEINS* [2], *D&D* [7] and *FRANKENSTEIN* [28], while the social network dataset includes *IMDB-binary* [47]. The statistic of the datasets is summarized in Table 1. We provide further details for these datasets in Appendix B.

Table 1: Summary of datasets.

	# Graphs	Avg.(nodes)	Avg.(edges)	# Classes	# Features	K
PTC	344	25.5	25.96	2	19	72
PROTEINS	1,113	39.06	72.82	2	3	251
D&D	1,178	284.32	715.66	2	82	228
FRANKENSTEIN	4,337	16.90	17.88	2	4	922
IMDB-binary	1,000	19.77	96.53	2	65	205

Base GNN models. Our SOLT-GNN is agnostic of the base GNN models and can flexibly work with most of the neighborhood aggregation based GNN architectures for graph representation learning. By default, we employ GIN [46] as the base GNN model in our experiments due to its effectiveness, thus forming the variant of SOLT-GIN. To evaluate the flexibility of SOLT-GNN with different GNNs, we further employ another two popular GNNs as the base models, *i.e.*, GCN [15] and GraphSAGE [13], thus forming another two variants, SOLT-GCN and SOLT-SAGE, respectively. Further details and settings for base GNN models are in Appendix C.

Baselines. To comprehensively evaluate the proposed SOLT-GNN against the state-of-the-art approaches, we consider a series of baselines from three main categories, *graph kernel based approaches*, *graph neural neworks* and *graph pooling approaches*. (1) *Graph kernel approaches*: GK (Graphlet Kernel) [36] and WL (Weisfeiler-Lehman subtree kernel) [35]. They are classical graph representation learning methods that generating graph representations directly by exploiting graph substructures with handcrafted kernel functions. (2) *Graph neural networks*: Mixup [44] and CurGraph [43]. They follow the architecture of layer-wise neighborhood aggregation to achieve expressive node representations and use the READOUT function to summarize the node representations to generate graph representations. In particular, they capitalize on advanced training techniques over the basic GNNs to improve the performance, *i.e.*, Mixup and curriculum learning, respectively. (3) *Graph pooling approaches*: SortPool [51], DiffPool [50] and MxGNN [18]. They develop diverse graph pooling mechanisms to generate more expressive graph representations in order to achieve promotion for graph classification. More details of the baselines are in Appendix D.

Settings and parameters. For each dataset, we divide graphs into head and tail with predefined K. In particular, for each dataset, K is determined based on the Pareto principle (also known as 20/80 rule) [33] to employ the 20% largest graphs as head graphs, and the rest 80% as tail graphs. Therefore, K is different across datasets, as shown in Table 1. Note that, K is an important hyper-parameter to decide the partition of knowledge transfer source (*i.e.*, head graphs) and target (*i.e.*, tail graphs). We also conduct experiments with different K's in Sect. 5.3, and conclude that our model can persistently empower the knowledge transfer to enhance the tail graph classification performance. We adopt accuracy as the metric to evaluate the performance, which is widely employed in the task of graph classification [46, 50, 51]. For each dataset, we randomly split the graphs with proportion of 7:1:2 as training, validation and testing set respectively, thus the graphs in all sets follow the long-tailed distribution. All experiments are repeated for five times, and we report the averaged results with standard deviations. Note that, in evaluation, we mainly focus on the performance promotion of tail

Table 2: Long-tailed graph classification using GIN as the base model.

Henceforth, tabular results are in percent; the best result is **bolded** and the runner-up is underlined.

Methods	PTC		PROTEINS		D&D		FRANKENSTEIN		IMDB-binary	
	Test	Tail	Test	Tail	Test	Tail	Test	Tail	Test	Tail
GK	52.3 ± 2.5	50.6 ± 3.6	68.9 ± 1.9	66.4 ± 1.8	75.1 ± 2.4	72.0 ± 2.8	63.1 ± 1.3	62.3 ± 1.5	52.7 ± 3.3	57.0 ± 3.2
WL	57.8 ± 4.8	54.8 ± 2.7	74.0 ± 3.6	71.1 ± 4.4	78.2 ± 1.2	76.0 ± 1.5	72.4 ± 1.2	72.1 ± 1.3	71.4 ± 2.6	68.4 ± 2.9
GIN	56.2 ± 6.1	54.3 ± 5.5	74.9 ± 1.1	73.1 ± 2.2	76.3 ± 3.4	72.7 ± 4.2	72.4 ± 1.4	72.4 ± 1.2	76.9 ± 1.1	73.6 ± 1.2
CurGraph	54.1 ± 4.6	58.8 ± 5.8	74.3 ± 1.6	74.5 ± 0.7	69.6 ± 2.2	68.1 ± 2.8	71.3 ± 1.4	70.8 ± 1.2	73.5 ± 1.7	70.5 ± 1.5
Mixup	58.8 ± 4.3	56.3 ± 4.9	72.9 ± 0.6	71.5 ± 1.6	73.6 ± 3.4	72.1 ± 4.4	70.2 ± 0.7	70.1 ± 0.9	76.5 ± 2.5	72.9 ± 2.4
SortPool	58.8 ± 2.1	59.1 ± 2.7	72.0 ± 1.2	70.4 ± 1.9	74.4 ± 2.0	74.7 ± 4.0	71.3 ± 1.4	70.2 ± 1.2	68.4 ± 5.4	66.2 ± 4.2
DiffPool	58.8 ± 4.4	58.4 ± 4.3	72.0 ± 1.0	69.4 ± 2.0	74.3 ± 1.1	72.2 ± 2.3	69.8 ± 1.5	69.2 ± 1.3	72.5 ± 0.8	70.3 ± 0.8
MxGNN	60.3 ± 3.7	60.6 ± 4.3	74.7 ± 1.5	71.6 ± 2.1	78.2 ± 0.6	76.7 ± 0.8	71.7 ± 1.7	71.0 ± 2.0	74.5 ± 1.8	73.2 ± 2.5
SOLT-GIN	**61.8** ± 6.8	**62.0** ± 6.4	**76.4** ± 1.8	**75.1** ± 2.3	**78.3** ± 1.5	**77.0** ± 2.5	**73.0** ± 1.4	**73.1** ± 1.3	**77.5** ± 1.3	**74.3** ± 1.5

Table 3: Long-tailed graph classification using other GNNs as the base models.

Methods	PTC		PROTEINS		D&D		FRANKENSTEIN		IMDB-binary	
	Test	Tail	Test	Tail	Test	Tail	Test	Tail	Test	Tail
GCN	53.5 ± 2.5	53.3 ± 2.7	74.2 ± 0.6	71.9 ± 1.0	78.5 ± 1.4	76.6 ± 1.7	71.1 ± 0.5	70.4 ± 0.9	75.7 ± 1.5	71.5 ± 1.5
SOLT-GCN	**57.9** ± 4.5	**61.4** ± 3.3	**76.7** ± 1.6	**75.1** ± 2.0	**79.5** ± 2.4	**78.1** ± 2.4	**71.7** ± 0.4	**70.7** ± 0.4	**77.1** ± 1.6	**73.4** ± 1.9
GraphSAGE	56.5 ± 4.5	55.1 ± 4.3	74.1 ± 1.2	72.9 ± 1.1	77.7 ± 1.7	76.1 ± 1.6	69.0 ± 3.5	67.8 ± 4.3	77.0 ± 3.0	73.4 ± 2.5
SOLT-SAGE	**63.2 ± 5.6**	**65.3** ± 6.4	**75.2** ± 0.9	**74.6** ± 1.6	**79.0** ± 0.6	**77.6** ± 0.8	**70.7** ± 1.0	**69.6** ± 1.0	**77.5** ± 1.7	**74.5** ± 2.1

graphs by SOLT-GNN; while for head graphs, we directly employ the representations produced by the base GNN model for prediction, since their abundant structures are already representative enough.

5.2 Long-Tailed Graph Classification

We conduct long-tailed graph classification on the five benchmark datasets for comparison. In particular, as we mainly focus on the performance promotion of tail graphs, we report the graph classification accuracy from two perspectives, *i.e.*, Test (the entire testing set) and Tail (the tail graph set in testing set).

For fair comparison, we employ GIN as the base model for all the GNN-based approaches. To evaluate the flexibility of SOLT-GNN on diverse GNN architectures, we further adopt GCN and GraphSAGE as the base GNN models for extension.

Using GIN as the base model. We report the graph classification comparison with state-of-the-art approaches in Table 2, and observe that SOLT-GIN can generally outperform all the baselines.

In particular, we make the following observations. First, the performance gap between entire testing set and the tail graph set generally emerges on all approaches across the five datasets, which demonstrates that the size-oriented long-tailed problem is universal and significant. Second, SOLT-GIN outperforms its base model GIN with a large margin. This proves the feasibility of pattern relevance extraction on head graphs, as well as the transferability of structural information across graphs. Furthermore, by virtue of the knowledge transfer from head to tail graphs, SOLT-GNN can alleviate the shortage of structural information on tail graphs to form a more comprehensive representation. Third, approaches CurGraph and MxGNN, which investigate the graph size effect on the performance, fail to generate expressive representations for tail graphs. Though they treat the head and tail graphs differently, the structure-scarce tail graphs still suffer from the lack of structural information, which hinders the promotion of these approaches on the tail graph representations. Fourth, graph pooling approaches such as SortPool, DiffPool and MxGNN, have approaching performance on tail graph classification compared to GIN. Though they capitalize on the purposely designed pooling mechanisms to enhance the graph representations, they are unable to address the tail graph problem, thus produce inferior results.

Using other GNNs as base models. To demonstrate the flexibility of SOLT-GNN, we further utilize GCN and GraphSAGE as the base GNN models for evaluation. The performance results are reported in Table 3. We can observe that, SOLT-GNN can outperform its base GNN models in each case, showing the flexibility of SOLT-GNN to apply with different GNN architectures for knowledge transfer to enhance the representational capacity of tail graphs.

5.3 Model Analysis

We further analyse SOLT-GNN from several aspects for tail graph classification, by resorting to GIN as the base GNN model.

Ablation study. To evaluate the contribution of each component in SOLT-GNN, we conduct an ablation study by comparing with several degenerate variants: (1) *no head*: we remove the graph classification loss of head graphs, *i.e.*, $\alpha = 0$ in Eq. (11); (2) *no node-level*: we remove the node-level pattern relevance loss, *i.e.*, $\mu_1 = 0$ in Eq. (13); (3) *no subg-level*: we remove the subgraph-level pattern relevance loss, *i.e.*, $\mu_2 = 0$ in Eq. (13); (4) *no scale & shift*: we remove the scaling and shifting factors of pattern memory in Eq. (6).

We show the results of ablation study in Fig. 4, and make the following observations. First, as the core design of SOLT-GNN is to transfer knowledge from head to tail graphs, when we remove the

Table 4: Impact of threshold K (*i.e.*, the proportion of head) on graph classification.

Methods	10%						30%					
	PROTEINS		D&D		IMDB-binary		PROTEINS		D&D		IMDB-binary	
	Test	Tail	Test	Tail	Test	Tail	Test	Tail	Test	Tail	Test	Tail
GIN	74.9 ± 1.1	73.7 ± 2.6	76.3 ± 3.4	74.6 ± 3.8	76.9 ± 1.1	74.8 ± 1.1	74.9 ± 1.1	71.1 ± 3.5	76.3 ± 3.4	71.9 ± 4.8	76.9 ± 1.1	74.7 ± 1.4
CurGIN	74.3 ± 1.6	73.5 ± 2.5	69.6 ± 2.2	68.6 ± 3.5	73.5 ± 1.7	72.0 ± 2.0	74.3 ± 1.6	72.8 ± 2.6	69.6 ± 2.2	67.7 ± 4.3	73.5 ± 1.7	69.7 ± 0.7
DiffPool	72.0 ± 1.0	70.6 ± 1.1	74.3 ± 1.1	73.9 ± 1.3	72.5 ± 0.8	70.9 ± 0.9	72.0 ± 1.0	67.2 ± 1.2	74.3 ± 1.1	70.8 ± 1.5	72.5 ± 0.8	70.5 ± 1.0
MxGNN	74.7 ± 1.5	72.7 ± 1.8	78.2 ± 0.6	77.7 ± 0.7	74.5 ± 1.8	74.0 ± 2.4	74.7 ± 1.5	72.8 ± 2.8	78.2 ± 0.6	76.2 ± 0.9	74.5 ± 1.8	74.0 ± 2.3
SOLT-GIN	**77.8** ± 2.7	**76.1** ± 3.0	**79.5** ± 1.7	**78.2** ± 1.9	**78.1** ± 1.9	**76.2** ± 2.3	**77.1** ± 1.5	**73.2** ± 2.4	**80.9** ± 1.3	**77.9** ± 1.5	**77.1** ± 1.5	**75.0** ± 1.8

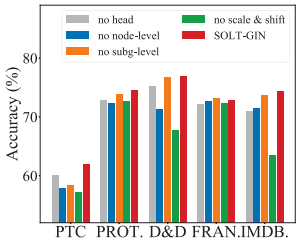


Figure 4: Ablation study.

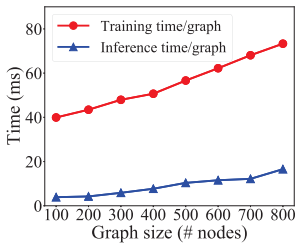
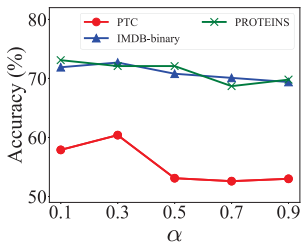
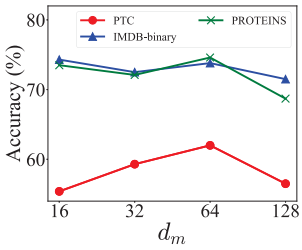


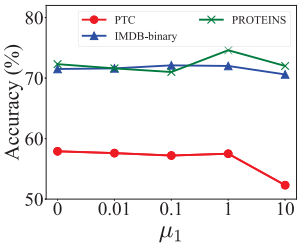
Figure 5: Scalability study.



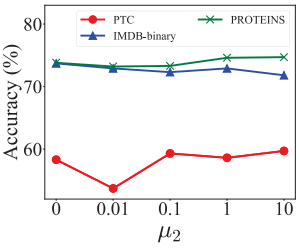
(a) Weight of head graph loss α



(b) Pattern memory dimension d_m



(c) Node-level co-occurrence μ_1



(d) Subgraph-level co-occurrence μ_2

Figure 6: Parameters sensitivity.

graph classification loss of head graphs, the underlying transferable patterns cannot be effectively extracted from head graphs, resulting in inferior performance. Second, without node- or subgraph-levels pattern relevance loss, the performance also generally decreases, showing that both of them can contribute to the pattern relevance extraction. Third, without the scaling and shifting factors, SOLT-GNN cannot well adapt the globally shared pattern memory towards each query, which impairs the relevance prediction. Finally, the whole model SOLT-GNN can achieve the best performance, demonstrating its power for tail graph classification.

Impact of threshold K. Though we follow the Pareto principle [33] to determine the threshold K, in Table 4 we also resort to different proportions for data division, *i.e.*, considering the 10% and 30% largest graphs as head graphs, to compare with several representative baselines on three datasets. We first observe that with more graphs marked as head graphs, the performance of baselines on tail graph classification decreases, exposing the bottleneck of them in coping with the challenging tail graphs. Second, SOLT-GIN can outperform the baselines with different threshold K, which further proves the feasibility of transferring knowledge from head graphs to tail graphs. Third, for Test accuracy, SOLT-GIN performs differently with varying K's on different datasets, showing that the optimal division for knowledge transfer differs across datasets.

Scalability. We also evaluate the scalability of SOLT-GIN on dataset D&D which has the largest averaged graph size. In particular, we construct eight graph groups with increasing graph size from 100 to 800, and each group contains ten graphs with approximate sizes. We show the training and inference time per graph in Fig. 5, and observe that both of them generally increase linearly as the graph size increases. The linear growth demonstrates that the proposed SOLT-GNN can scale to very large graphs in real-world scenarios.

Parameters Sensitivity. We evaluate the sensitivity of several important hyperparameters in SOLT-GNN, and show their impact in Fig. 6. For head graph loss weight α, too large values may decrease the contribution of tail graphs in the model training, thus impairing the performance. In particular, relative small values such as [0.1,0.3] can bring more benefits to the model training. For pattern memory dimension d_m, too small values may limit the storage capacity of pattern memory, while too large values may result in overfitting. Moderate values such as 32 or 64 would benefit the performance. For the weight of node- and subgraph-levels co-occurrence μ_1 and μ_2, moderate values such as $\mu_1 = 1$ and $\mu_2 = 1$ or 10 can facilitate the representation learning of tail graphs.

6 CONCLUSION

In this paper, we investigate a significant yet unexploited problem, size-oriented long-tailed graph classification with graph neural networks. To cope with this issue, we propose a novel model SOLT-GNN, to enrich the structure-scarce tail graphs from the lens of knowledge transfer. Two novel modules, *i.e.*, co-occurrence substructure exploitation and relevance prediction function, are employed in SOLT-GNN. Extensive experiments on five benchmark datasets demonstrate the effectiveness of our proposed SOLT-GNN.

ACKNOWLEDGMENTS

This research is supported by the Agency for Science, Technology and Research (A*STAR) under its AME Programmatic Funds (Grant No. A20H6b0151).

REFERENCES

[1] Daniel Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-Sequence Learning using Gated Graph Neural Networks. In *ACL*. 273–283.

[2] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl_1 (2005), i47–i56.

[3] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE TKDE* 30, 9 (2018), 1616–1637.

[4] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: a library for support vector machines. *ACM TIST* 2, 3 (2011), 1–27.

[5] Zhihong Chen, Rong Xiao, Chenliang Li, Gangfeng Ye, Haochuan Sun, and Hongbo Deng. 2020. ESAM: Discriminative Domain Adaptation with Non-Displayed Items to Improve Long-Tail Performance. In *SIGIR*. 579–588.

[6] Kaize Ding, Jianling Wang, Jundong Li, Kai Shu, Chenghao Liu, and Huan Liu. 2020. Graph prototypical networks for few-shot learning on attributed networks. In *CIKM*. 295–304.

[7] Paul D Dobson and Andrew J Doig. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology* 330, 4 (2003), 771–783.

[8] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *NeurIPS*. 2224–2232.

[9] Hongyang Gao and Shuiwang Ji. 2019. Graph u-nets. In *ICML*. 2083–2092.

[10] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *ICML*. 1263–1272.

[11] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. 855–864.

[12] David Ha, Andrew Dai, and Quoc V Le. 2017. Hypernetworks. In *ICLR*.

[13] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1025–1035.

[14] Bingyi Kang, Saining Xie, Marcus Rohrbach, Zhicheng Yan, Albert Gordo, Jiashi Feng, and Yannis Kalantidis. 2020. Decoupling representation and classifier for long-tailed recognition. In *ICLR*.

[15] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[16] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. 2020. A survey on graph kernels. *Applied Network Science* 5, 1 (2020), 1–42.

[17] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-attention graph pooling. In *ICML*. 3734–3743.

[18] Yanyan Liang, Yanfeng Zhang, Dechao Gao, and Qian Xu. 2021. An End-to-End Multiplex Graph Neural Network for Graph Representation Learning. *IEEE Access* 9 (2021), 58861–58869.

[19] Jialun Liu, Yifan Sun, Chuchu Han, Zhaopeng Dou, and Wenhui Li. 2020. Deep Representation Learning on Long-tailed Data: A Learnable Embedding Augmentation Perspective. In *CVPR*. 2970–2979.

[20] Siyi Liu and Yujia Zheng. 2020. Long-tail session-based recommendation. In *RecSys*. 509–514.

[21] Zemin Liu, Yuan Fang, Chenghao Liu, and Steven C.H. Hoi. 2021. Node-wise Localization of Graph Neural Networks. In *IJCAI*. 1520–1526.

[22] Zemin Liu, Yuan Fang, Chenghao Liu, and Steven CH Hoi. 2021. Relative and Absolute Location Embedding for Few-Shot Node Classification on Graph. In *AAAI*.

[23] Zemin Liu, Yuan Fang, Yong Liu, and Vincent W. Zheng. 2021. Neighbor-Anchoring Adversarial Graph Neural Networks. *IEEE TKDE* Early Access (2021).

[24] Zemin Liu, Trung-Kien Nguyen, and Yuan Fang. 2021. Tail-GNN: Tail-Node Graph Neural Networks. In *KDD*. 1109–1119.

[25] Zemin Liu, Wentao Zhang, Yuan Fang, Xinming Zhang, and Steven CH Hoi. 2020. Towards locality-aware meta-learning of tail node embeddings on networks. In *CIKM*. 975–984.

[26] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. 2019. Graph convolutional networks with eigenpooling. In *KDD*. 723–731.

[27] Diego Mesquita, Amauri Souza, and Samuel Kaski. 2020. Rethinking pooling in graph neural networks. In *NeurIPS*.

[28] Francesco Orsini, Paolo Frasconi, and Luc De Raedt. 2015. Graph invariant kernels. In *IJCAI*. 3756–3762.

[29] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. 2018. FiLM: Visual reasoning with a general conditioning layer. In *AAAI*. 3942–3951.

[30] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online learning of social representations. In *KDD*. 701–710.

[31] Liang Qu, Huaisheng Zhu, Ruiqi Zheng, Yuhui Shi, and Hongzhi Yin. 2021. ImGAGN: Imbalanced Network Embedding via Generative Adversarial Graph Networks. In *KDD*. 1390–1398.

[32] Ekagra Ranjan, Soumya Sanyal, and Partha Talukdar. 2020. Asap: Adaptive structure aware pooling for learning hierarchical graph representations. In *AAAI*. 5470–5477.

[33] Robert Sanders. 1987. The Pareto principle: its use and abuse. *Journal of Services Marketing* 1, 2 (1987), 37–40.

[34] Aravind Sankar, Junting Wang, Adit Krishnan, and Hari Sundaram. 2021. ProtoCF: Prototypical Collaborative Filtering for Few-shot Recommendation. In *RecSys*. 166–175.

[35] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research* 12, 9 (2011).

[36] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *AISTATS*. 488–495.

[37] Min Shi, Yufei Tang, Xingquan Zhu, David Wilson, and Jianxun Liu. 2020. Multi-class imbalanced graph convolutional network learning. In *IJCAI*.

[38] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale information network embedding. In *WWW*. 1067–1077.

[39] Kaihua Tang, Jianqiang Huang, and Hanwang Zhang. 2020. Long-Tailed Classification by Keeping the Good and Removing the Bad Momentum Causal Effect. In *NeurIPS*.

[40] Hannu Toivonen, Ashwin Srinivasan, Ross D King, Stefan Kramer, and Christoph Helma. 2003. Statistical evaluation of the predictive toxicology challenge 2000–2001. *Bioinformatics* 19, 10 (2003), 1183–1193.

[41] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.

[42] Nikil Wale, Ian A Watson, and George Karypis. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems* 14, 3 (2008), 347–375.

[43] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. 2021. CurGraph: Curriculum Learning for Graph Classification. In *TheWebConf*. 1238–1248.

[44] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. 2021. Mixup for Node and Graph Classification. In *TheWebConf*. 3663–3674.

[45] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE TNNLS* 32, 1 (2020), 4–24.

[46] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks?. In *ICLR*.

[47] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *KDD*. 1365–1374.

[48] Gilad Yehudai, Ethan Fetaya, Eli Meirom, Gal Chechik, and Haggai Maron. 2021. On Size Generalization in Graph Neural Networks. https://openreview.net/forum?id=9p2CltauWEY

[49] Jianwen Yin, Chenghao Liu, Weiqing Wang, Jianling Sun, and Steven CH Hoi. 2020. Learning transferrable parameters for long-tailed sequential user behavior modeling. In *KDD*. 359–367.

[50] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *NeurIPS*. 4800–4810.

[51] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *AAAI*.

[52] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. 2021. GraphSMOTE: Imbalanced Node Classification on Graphs with Graph Neural Networks. In *WSDM*. 833–841.

[53] Dawei Zhou, Jingrui He, Hongxia Yang, and Wei Fan. 2018. Sparc: Self-paced network representation for few-shot rare category characterization. In *KDD*. 2807–2816.

APPENDICES

A Algorithm and Complexity Analysis

Algorithm. We illustrate the model training of SOLT-GNN in Alg. 1. In line 1, we initialize all the parameters. In lines 3-9, we exploit the co-occurrence substructures from each head graph and accumulate the pattern relevance loss. In particular, in lines 5 and 6, we exploit the co-occurrence substructures from node- and subgraph-levels, respectively. In lines 7 and 8, we accumulate the pattern relevance loss for each of them. In line 9, we complement the graph representation for each graph. In line 10, we form the graph classification loss. In line 11, we form the complement dissimilarity constraint. In line 12, we summarize all the loss and constraints to generate the overall objective. Finally, we optimize the objective in line 13.

Complexity analysis. The proposed SOLT-GNN needs to exploit the co-occurrence substructures from the head graphs, which can be done offline. Thus, we analyse its complexity from two aspects, *i.e.*, offline and online. **(1) Offline.** Given a graph $G = \{V, E, \mathbf{X}\}$, for node-level substructures exploitation, we randomly sample a positive and a negative sample for each node $v \in V$, thus involving complexity of $O(|V|)$ in all; for subgraph-level substructures exploitation, we only generate one triplet for each graph, and conduct BFS to sample the query G^{sub} on G with complexity of $O(|V|)$, while the generation of positive and negative samples, *i.e.*, G^{rem} and G^{neg}, involves complexity of $O(1)$ and $O(|V|)$, respectively. Thus, in total, the offline data preparation involves complexity of $O(|V|)$ for each graph. **(2) Online.** Given a graph $G = \{V, E, \mathbf{X}\}$ with averaged node degree $\bar{d}$, the complexity of a neighborhood aggregation based GNN (*e.g.*, GIN) is $O(|V| \cdot \bar{d} \cdot L)$, where L is the total number of GNN layers. Compared to the base GNN, in the online period, SOLT-GNN only needs an extra complement to enrich each graph, and this complement can be calculated directly by virtue of the relevance prediction function, with complexity of $O(1)$. Thus, the online complexity of SOLT-GNN belongs to same complexity class with its base GNN model, *i.e.*, $O(|V| \cdot \bar{d} \cdot L)$.

B Further Details of Datasets

We employ a total of five benchmark datasets, including four bioinformatics datasets and one social network dataset. (1) *PTC* [40] is a collection of 344 chemical compounds represented in the form of graphs, which report the carcinogenicity for rats. There are 19 node labels, and we utilize them as the node features. (2) *PROTEINS* [2] is a protein dataset, in which nodes are secondary structure elements and edges are neighboring relations in the amino-acid sequence or in 3D space. The graphs are labelled into two groups, *i.e.*, enzymes and non-enzymes. (3) *D&D* [7] is another protein dataset with larger sizes. Nodes represent the amino acids, and two nodes are connected by an edge if they are less than 6 Angstroms apart. (4) *Frankenstein* [28] is a biological molecule dataset, where each graph is a molecule with or without mutagenicity, and nodes and edges represent atoms and chemical bonds, respectively. We employ the node degrees as the node labels (features). (5) *IMDB-binary* [47] is a movie collaboration dataset. Each graph corresponds to an ego-network for each actor/actress, where nodes correspond to actors/actresses and each edge indicates that two actors/actresses appear in the same movie. Each graph is derived from a pre-specified genre of movies, and is labelled by the genre of derived graph. Since there are no node labels in IMDB-binary, we employ the node degrees as the node labels (features).

C Further Details and Settings of Base GNN Models

Descriptions. We describe the three base GNN models below.

- GCN [15]: GCN depends on the key operation of neighborhood aggregation to aggregate messages from the neighboring nodes to form the node representations. In particular, it employs a mean-pooling to aggregate the neighborhood information.
- GraphSAGE [13]: Similar to GCN, GraphSAGE also resorts to the neighborhood aggregation to generate the node representations of the target nodes. Differently, it pays more attention to the information from the target node itself.
- GIN [46]: GIN also relies on the neighborhood aggregation for node representation learning. Furthermore, it employs a SUM-based aggregation to replace the mean-pooling to aggregate the messages from neighbors, which is more expressive to capture the local structures.

Settings. We follow the model configurations in their original papers for base GNNs' implementation. Based on the recommendation from their original papers, we further tune the parameters for each base GNN model to attain the optimal performance for graph classification task. In detail, for each base GNN model, we resort to a five-layer architecture followed with a SUM-pooling layer, which is proposed by GIN [46] and can benefit the graph representation learning. We set the number of hidden units as 32 for all of them on bioinformatics datasets and 64 on *IMDB-binary*. And we set the dropout rate as 0.5. For GraphSAGE, we employ the mean-pooling as its aggregator.

D Further Details of Baselines

To comprehensively evaluate the proposed SOLT-GNN against the state-of-the-art approaches, we consider a series of baselines from three main categories, *graph kernel approaches*, *graph neural networks* and *graph pooling approaches*.

(1) *Graph kernel approaches.*

- GK: Graphlet Kernel [36] counts graphlets on graphs and use graphlet distribution to learn graph representations.
- WL: Weisfeiler-Lehman Subtree Kernel [35] iteratively updates a node's feature vector based on its neighbors' features to encode the structural information of graphs.

(2) *Graph Neural Networks.*

- Mixup: Mixup [44] applies both node-level and graph-level Mixup methods on graph neural networks to combat the over-fitting of graph neural networks, and we consider graph-level Mixup for graph classification.
- CurGraph: CurGraph [43] utilizes curriculum learning to help the training process of GNNs for effective representations. To sufficiently investigate the performance of head graphs and tail graphs, we use graph size as the heuristic difficulty metric for graph data.

Algorithm 1 Model Training for SOLT-GNN

Input: Training graphs set $\mathcal{G}_{\text{tr}}$ with labels Y_{tr}, hyper-parameters K, α, λ, μ_1, and μ_2.
Output: Model parameters Θ.

initialize parameters Θ;
while not converged **do**
  for each graph $G_i \in \mathcal{G}_{\text{tr}}$ **do**
    if $G_i \in \mathcal{G}_{\text{head}}$ **then** ▷ Co-occurrence substructures exploitation
      sample node-level triplets $\mathcal{T}_{\text{node}}$;
      sample subgraph-level triplets $\mathcal{T}_{\text{subg}}$;
      $\mathcal{L}_{\text{rel}}^{\text{node}} \leftarrow \mathcal{L}_{\text{rel}}^{\text{node}} - \sum_{(q,a,b)\in\mathcal{T}_{\text{node}}} \ln \sigma(\text{sim}(\mathbf{r}_q, \mathbf{h}_a) - \text{sim}(\mathbf{r}_q, \mathbf{h}_b))$; ▷ Node-level pattern relevance loss, Eq. (4)
      $\mathcal{L}_{\text{rel}}^{\text{subg}} \leftarrow \mathcal{L}_{\text{rel}}^{\text{subg}} - \sum_{(q,a,b)\in\mathcal{T}_{\text{subg}}} \ln \sigma(\text{sim}(\mathbf{r}_q, \mathbf{h}_a) - \text{sim}(\mathbf{r}_q, \mathbf{h}_b))$; ▷ Subgraph-level pattern relevance loss, Eq. (5)
    $\tilde{\mathbf{h}}_{G_i} \leftarrow \mathbf{h}_{G_i} + \phi_r(\mathbf{h}_{G_i}; \theta_r) = \mathbf{h}_{G_i} + \mathbf{r}_{G_i}$; ▷ Enrich graph representation, Eq. (10)
  $\mathcal{L}_{\text{task}} \leftarrow \alpha \cdot \sum_{G_i\in\mathcal{G}_{\text{tr}}\cap\mathcal{G}_{\text{head}}} \text{CrossEnt}(\tilde{\mathbf{h}}_{G_i}, \mathbf{y}_i) + (1-\alpha) \cdot \sum_{G_i\in\mathcal{G}_{\text{tr}}\cap\mathcal{G}_{\text{tail}}} \text{CrossEnt}(\tilde{\mathbf{h}}_{G_i}, \mathbf{y}_i)$; ▷ Task loss, Eq. (11)
  $\mathcal{L}_{\text{dis}} \leftarrow \sum_{G_i\in\mathcal{G}_{\text{tr}}} \text{sim}(\mathbf{h}_{G_i}, \mathbf{r}_{G_i})$; ▷ Complement dissimilarity constraint, Eq. (12)
  $\mathcal{L} \leftarrow \mathcal{L}_{\text{task}} + \lambda \cdot \mathcal{L}_{\text{dis}} + \mu_1 \cdot \mathcal{L}_{\text{rel}}^{\text{node}} + \mu_2 \cdot \mathcal{L}_{\text{rel}}^{\text{subg}} + \Omega(\Theta)$; ▷ Overall objective, Eq. (13)
  update Θ by minimizing $\mathcal{L}$;
return Θ.

(3) *Graph pooling approaches.*

- SortPool: SortPool [51] applies a GNN architecture and re-sorts to a top-k pooling mechanism which sorts nodes within the graph according to their structural roles. The sorted node features are then fed into the convolutional and dense layers to learn graph representations.
- DiffPool: DiffPool [50] is a differentiable graph pooling method that can learn hierarchical representations in an end-to-end fashion, and a clustering assignment matrix is proposed to coarsen the graphs.
- MxGNN: MxGNN [18] comprises multiple graph convolution networks with different hyperparameters to learn effective graph representations for graphs with different properties, and DiffPool layers are applied as the pooling mechanism.

E Hyperparameters Settings

Baselines. For baseline GK, we set the size of graphlets as 3, and use C-SVM [4] with linear kernel to implement the graph classification. For WL [35], we search the height parameter of WL in $\{0, 1, 2, 3, 4, 5\}$ and also use C-SVM with linear kernel to implement the graph classification. For Mixup [44], we employ GIN [46] as the base model to implement Mixup, and the configuration of Mixup is the same to GIN. For CurGraph [43], to investigate the effect of graph size, we use graph size as the difficulty metric for graph data which is treated as a heuristics metric in their original paper. We divide graphs into four groups according to their sizes, and treat large graphs as difficult data and train GIN in an easy-to-difficult order. For SortPool [51], according to the original paper, we set the pooling ratio k for each dataset such that 60% graphs have nodes more than k. The hidden dimension of graph convolution layers are set as 32 for all datasets. For DiffPool [50], we use a total of two DIFFPOOL layers, and three graph convolution layers are employed after each DIFFPOOL layer. And we set the hidden dimension of convolution layers as 32 and the number of clusters as 10. For MxGNN [18], we keep the based configuration of MxGNN consisitent with DiffPool and run two graph convolution networks separately to deal with head and tail graphs, respectively. For the hidden dimensions of the two graph convolution networks, we set them as 80 and 30 for head and tail graphs, respectively.

Our model. For co-occurrence substructures exploitation, on each head graph, we randomly generate one node-level triplet for each node, and one subgraph-level triplet for this graph. The size of sampled query subgraphs, *i.e.*, t, is set to $min(k, |V|/2)$ in order to mimic the tail graphs, as well as to avoid the left structure which serves as the positive sample being too small. For the hyperparameters of SOLT-GNN, we set weight $\alpha = 0.05$ for *D&D*, 0.1 for *FRANKENSTEIN*, 0.15 for *PROTEINS* and *IMDB-binary*, and 0.3 for *PTC*. For weight μ_1, we set weight 0.5 for *IMDB-binary*, 1.5 for *PTC*, and 2 for the other datasets. For weight μ_2, we set weight 0 for *FRANKENSTEIN*, 1.5 for *PTC*, and 2 for the other datasets. We set $\lambda = 0$ for *FRANKENSTEIN* and $\lambda = 1e-4$ for the other datasets, and set dimension of pattern memory $d_m = 16$ for *IMDB-binary* and 64 for the other datasets.