Graph Few-shot Learning with Task-specific Structures

Song Wang
University of Virginia
sw3wv@virginia.edu

Chen Chen
University of Virginia
zrh6du@virginia.edu

Jundong Li
University of Virginia
jundong@virginia.edu

Abstract

Graph few-shot learning is of great importance among various graph learning tasks. Under the few-shot scenario, models are required to conduct classification given limited labeled samples. Existing graph few-shot learning methods typically leverage Graph Neural Networks (GNNs) and perform classification across a series of meta-tasks. Nevertheless, these methods generally rely on the original graph (i.e., the graph that the meta-task is sampled from) to learn node representations. Consequently, the graph structure used in each meta-task is identical. Since the class sets are different across meta-tasks, node representations should be learned in a task-specific manner to promote classification performance. Therefore, to adaptively learn node representations across meta-tasks, we propose a novel framework that learns a task-specific structure for each meta-task. To handle the variety of nodes across meta-tasks, we extract relevant nodes and learn task-specific structures based on node influence and mutual information. In this way, we can learn node representations with the task-specific structure tailored for each meta-task. We further conduct extensive experiments on five node classification datasets under both single- and multiple-graph settings to validate the superiority of our framework over the state-of-the-art baselines. Our code is provided at https://github.com/SongW-SW/GLITTER.

1 Introduction

Nowadays, graph-structured data is playing a crucial role in various real-world applications, such as molecular property prediction [1], knowledge graph completion [2], and recommender systems [3]. More recently, Graph Neural Networks (GNNs) [4, 5, 6, 7] have been proposed to learn node representations via information aggregation based on the given graph structure. Generally, these methods adopt a semi-supervised learning strategy to train models on a graph with abundant labeled samples [8]. However, in practice, it is often difficult to obtain sufficient labeled samples due to the laborious labeling process [9]. Hence, there is a surge of research interests aiming at performing graph learning with limited labeled samples as references, known as *graph few-shot learning* [10, 11, 12].

Among various types of graph few-shot learning tasks, *few-shot node classification* is essential in real-world scenarios, including protein classification [13] and document categorization [14]. To deal with the label deficiency issue in node classification, many recent works [15, 9, 1, 11] incorporate existing few-shot learning frameworks from other domains [16, 17] into GNNs. Specifically, few-shot classification during evaluation is conducted on a specific number of meta-test tasks. Each meta-test task contains a small number of labeled nodes as references (i.e., support nodes) and several unlabeled

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

nodes for classification (i.e., query nodes). To extract transferable knowledge from classes with abundant labeled nodes, the model is trained on a series of meta-training tasks that are sampled from these disjoint classes but share similar structures with meta-test tasks. We refer to meta-training and meta-test tasks as meta-tasks. Note that few-shot node classification can be conducted on a single graph (e.g., a citation network for author classification) or across multiple graphs (e.g., a set of protein-protein interaction networks for drug discovery). Here each meta-task is sampled from one single graph in both single-graph and multiple-graph settings, since each meta-test task is conducted on one graph. Despite the success of recent studies on few-shot node classification, they mainly learn node representations from the original graph (i.e., the graph that the meta-task is sampled from). However, the whole graph can be redundant and uninformative for a specific meta-task as each meta-task only contains a small number of nodes. As a result, the learned node representations are not tailored for the meta-task (i.e., task-specific), which increases the difficulties of few-shot learning. Thus, instead of leveraging the same original graph for all meta-tasks, it is crucial to learn a task-specific structure for each meta-task.

Intuitively, the task-specific structure should contain nodes in the meta-task along with other relevant nodes from the original graph. Moreover, the edge weights among these nodes should also be learned in a task-specific manner. Nevertheless, it remains a daunting problem to learn a task-specific structure for each meta-task due to two challenges: (1) It is non-trivial to select relevant nodes for the task-specific structure. Particularly, this structure should contain nodes that are maximally relevant to the support nodes in the meta-task. Nevertheless, since each meta-task consists of multiple support nodes, it is difficult to select nodes that are relevant to the entire support node set. (2) It is challenging to learn edge weights for the task-specific structure. The task-specific structure should maintain strong correlations for nodes in the same class, so that the learned node representations will be similar. Nonetheless, the support nodes in the same class could be distributed across the original graph, which increases the difficulty of enhancing such correlations for the task-specific structure.

To address these challenges, we propose a novel **G**raph few-shot **L**earning framework w**IT**h **T**ask-sp**E**cific st**R**uctures - GLITTER, which aims at effectively learning a task-specific structure for each meta-task in graph few-shot learning. Specifically, to reduce the irrelevant information from the original graph, we propose to select nodes via two strategies according to their overall node influence on support nodes in each meta-task. Moreover, we learn edge weights in the task-specific structure based on node influence within classes and mutual information between query nodes and labels. With the learned task-specific structures, our framework can effectively learn node representations that are tailored for each meta-task. In summary, the main contributions of our framework are as follows: (1) We selectively extract relevant nodes from the original graph and learn a task-specific structure for each meta-task based on node influence and mutual information. (2) The proposed framework can handle graph few-shot learning under both single-graph and multiple-graph settings. Differently, most existing works only focus on the single-graph setting. (3) We conduct extensive experiments on five real-world datasets under single-graph and multiple-graph settings. The superior performance over the state-of-the-art methods further validates the effectiveness of our framework.

2 Problem Formulation

Denote the set of input graphs as $\mathcal{G} = \{G_1, \ldots, G_M\}$ (for the single-graph setting, $|\mathcal{G}| = 1$), where M is the number of graphs. Here each graph can be represented as $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges, and $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$ is a feature matrix with the i-th row vector (d-dimensional) representing the attribute of the i-th node. Under the prevalent meta-learning framework, the training process is conducted on a series of meta-training tasks $\{\mathcal{T}_1, \ldots, \mathcal{T}_T\}$, where T is the number of meta-training tasks. More specifically, $\mathcal{T}_i = \{\mathcal{S}_i, \mathcal{Q}_i\}$, where $\mathcal{S}_i$ is the *support set* of $\mathcal{T}_i$ and consists of K labeled nodes for each of N classes (i.e., $|\mathcal{S}_i| = NK$). The corresponding label set of $\mathcal{T}_i$ is $\mathcal{Y}_i$, where $|\mathcal{Y}_i| = N$. $\mathcal{Y}_i$ is sampled from the whole training label set $\mathcal{Y}_{train}$. With $\mathcal{S}_i$ as references, the model is required to classify nodes in the *query set* $\mathcal{Q}_i$, which contains Q unlabeled samples. Note that the actual labels of query nodes are from $\mathcal{Y}_i$. After training, the model will be evaluated on a series of meta-test tasks, which follow a similar setting as meta-training tasks, except that the label set in each meta-test task is sampled from a distinct label set $\mathcal{Y}_{test}$ (i.e., $\mathcal{Y}_{test} \cap \mathcal{Y}_{train} = \emptyset$). It is noteworthy that under the multiple-graph setting, meta-training and meta-test tasks can be sampled from different graphs, while each meta-task is sampled from one single graph.

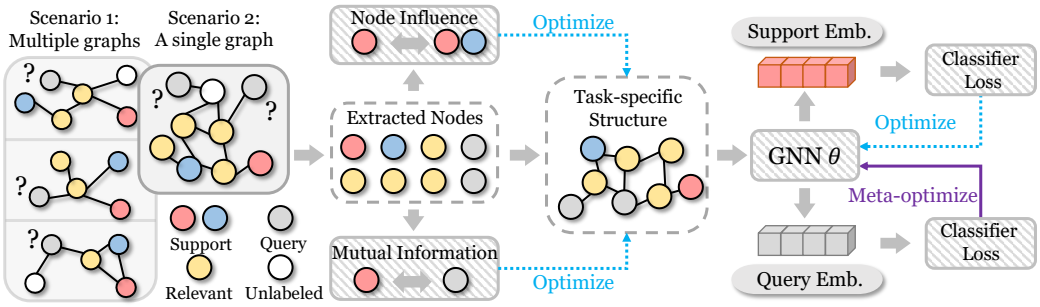

Figure 1: The overall framework of GLITTER. We first extract relevant nodes based on two strategies: local sampling and common sampling. Then we learn the task-specific structure with the extracted nodes along with support and query nodes based on node influence and mutual information. The learned structure will be used to generate node representations with a GNN. We further classify support nodes with a classifier, and the classification loss is used to optimize the GNN and the classifier. Finally, we meta-optimize the GNN and the classifier with the loss on query nodes.

3 Methodology

In this section, we introduce our framework that explores task-specific structures for different meta-tasks in graph few-shot learning. The detailed framework is illustrated in Figure 1. We first elaborate on the process of selecting relevant nodes based on node influence to construct the task-specific structure in each meta-task. Then we provide the detailed process of learning task-specific edge weights via maximizing node influence within classes and mutual information between query nodes and labels. Finally, we describe the meta-learning strategy used to optimize model parameters.

3.1 Selecting Nodes for Task-specific Structures

Given a meta-task $\mathcal{T} = \{\mathcal{S}, \mathcal{Q}\}$, we first aim to extract relevant nodes that are helpful for $\mathcal{T}$ and construct a task-specific structure $G_{\mathcal{T}}$ based on these nodes. In this way, we can avoid redundant information on the original graph and focus on meta-task $\mathcal{T}$. Nevertheless, it remains difficult to determine which nodes are useful for classification in $\mathcal{T}$. The reason is that the support nodes in $\mathcal{T}$ can be distributed across the original graph, which increases the difficulty of selecting nodes that are relevant to all these support nodes. Therefore, we propose to leverage the concept of node influence to select relevant nodes. Here we first define node influence based on [1, 5, 18] as follows:

Definition 1 (Node Influence)**.** *The node influence from node v_i to node v_j is defined as $I(v_i, v_j) = \|\partial \mathbf{h}_i / \partial \mathbf{h}_j\|$, where $\mathbf{h}_i$ and $\mathbf{h}_j$ are the output representations of v_i and v_j in a GNN, respectively. $\partial \mathbf{h}_i / \partial \mathbf{h}_j$ is a Jacobian matrix, and the norm can be any specific subordinate norm.*

According to Definition 1, large node influence denotes that the representation of a node can be easily impacted by another node, thus rendering stronger correlations. Intuitively, we need to incorporate more nodes with large influence on the support nodes into $G_{\mathcal{T}}$. In this way, $G_{\mathcal{T}}$ can maintain the most crucial information that is useful for classification based on support nodes. To effectively select nodes with larger influence on the support nodes, we consider important factors that affect node influence. The following theorem provides a universal pattern for node influence on support nodes:

Theorem 3.1. *Consider the node influence from node v_k to the i-th class (i.e., C_i) in a meta-task $\mathcal{T}$. Denote the geometric mean of the node influence values to all support nodes in C_i as $I_{C_i}(v_k) = \sqrt[K]{\prod_{j=1}^{K} I(v_k, s_{i,j})}$, where $s_{i,j}$ is the j-th support node in C_i. Assume the node degrees are randomly distributed with the mean value as $\bar{d}$. Then, $\mathbb{E}(\log I_{C_i}(v_k)) \geq -\log \bar{d} \cdot \sum_{j=1}^{K} SPD(v_k, s_{i,j})/K$, where $SPD(v_k, s_{i,j})$ denotes the shortest path distance between v_k and $s_{i,j}$.*

The proof is provided in Appendix A. Theorem 3.1 indicates that the lower bound of the node influence on a specific class is measured by its shortest path distances to all support nodes in this class. Therefore, to effectively select nodes with large influence on a specific class, we can choose nodes with small average shortest path distances to support nodes of this class. Based on this theorem, we propose two strategies, namely *local sampling* and *common sampling*, to select nodes for the task-specific structure $G_{\mathcal{T}}$. In particular, we combine the selected nodes with support and query

nodes in the meta-task (i.e., $\mathcal{S}$ and $\mathcal{Q}$) to obtain the final node set $\mathcal{V}_\mathcal{T} = \mathcal{V}_l \cup \mathcal{V}_c \cup \mathcal{S} \cup \mathcal{Q}$. Here $\mathcal{V}_l$ and $\mathcal{V}_c$ are the node sets extracted based on local sampling and common sampling, respectively. $\mathcal{S}$ and $\mathcal{Q}$ are the support set and the query set of $\mathcal{T}$, respectively. Then we introduce the two strategies in detail.

- **Local Sampling.** In this strategy, we extract the local neighbor nodes of support nodes within a specific distance (i.e., neighborhood size). Intuitively, the neighbor nodes can maintain a small shortest path distance to a specific support node. Therefore, by combining neighbor nodes of all support nodes in a class, we can obtain nodes with considerable node influence on this class without calculating the shortest path distances. Specifically, the extracted node set is denoted as $\mathcal{V}_l = \cup_{v_i \in \mathcal{S}} \mathcal{N}_l(v_i)$, where $\mathcal{N}_l(v_i) = \{u | d(u, v_i) \leq h\}$, and h is the pre-defined neighborhood size.
- **Common Sampling.** In this strategy, we select nodes that maintain a small average distance to all nodes in the same class. In this way, the node influence on an entire class can be considered. Specifically, for each of N classes in $\mathcal{T}$, we extract nodes with the smallest average distances to nodes in this support class. The overall extracted node set $\mathcal{V}_c$ can be presented as follows:

$$\mathcal{V}_c = \cup_{i=1}^{N} \underset{\mathcal{V}' \subset \mathcal{V}, |\mathcal{V}'|=C}{\operatorname{argmin}} \sum_{v \in \mathcal{V}'} \sum_{j=1}^{K} d(v, s_{i,j}), \tag{1}$$

  where $s_{i,j}$ is the j-th node of the i-th class in $\mathcal{T}$. Here we extract C nodes with the smallest sum of shortest path distances to nodes in each class. Then we aggregate these nodes into the final node set $\mathcal{V}_c$. In this way, we can select nodes with large influence on an entire class. As a result, the selected nodes will bear more crucial information for classifying a specific class. Note that since there are only N classes in $\mathcal{T}$, the maximum size of $\mathcal{V}_c$ is NC, i.e., $|\mathcal{V}_c| \leq NC$.

Edge Weight Functions. With the extracted node set $\mathcal{V}_\mathcal{T}$, we intend to learn task-specific edge weights for $G_\mathcal{T}$. Intuitively, although the original structural information is crucial for classification, it can also be redundant for meta-task $\mathcal{T}$. Therefore, we propose to construct the edges based on both node representations and the shortest path distance between two nodes. In this way, the model will learn to maintain and learn beneficial edges for $\mathcal{T}$. Particularly, the edge weight starting from node v_i to node v_j is denoted as $a_{i,j} = (a^r_{i,j} + a^s_{i,j})/2$, where $a^r_{i,j}$ and $a^s_{i,j}$ are learned by two functions that utilize node representations and structures as input, respectively.

- **Node representations as input.**

$$a^r_{i,j} = \exp\left(- \left\| \frac{\phi(\mathbf{W}_1 \mathbf{x}_i)}{\|\phi(\mathbf{W}_1 \mathbf{x}_i)\|_2} - \frac{\phi(\mathbf{W}_2 \mathbf{x}_j)}{\|\phi(\mathbf{W}_2 \mathbf{x}_j)\|_2} \right\|_2 \right), \tag{2}$$

  where ϕ is a non-linear activation function, and $\mathbf{x}_i$ denotes the input feature vector of node v_i. $\mathbf{W}_1 \in \mathbb{R}^{d_a \times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{d_a \times d}$ are learnable parameters, where d_a is the dimension size of $\mathbf{W}_1 \mathbf{x}_i$ and $\mathbf{W}_2 \mathbf{x}_j$. $a^r_{i,j}$ is the edge weight between node v_i and v_j learned from node representations. Such a design naturally satisfies that $a^r_{i,j} \in (0, 1]$. Moreover, by introducing two weight matrices $\mathbf{W}_1$ and $\mathbf{W}_2$, the learned task-specific structured will be a directed graph to encode more information.
- **Structures as input.**

$$a^s_{i,j} = \text{Sigmoid}\left(\psi\left(\text{SPD}(v_i, v_j)\right)\right), \tag{3}$$

  where ψ is a learned function that outputs a scalar while utilizing the shortest path distance between v_i and v_j (i.e., $\text{SPD}(v_i, v_j)$) on the original graph. In this way, we can preserve the structural information on the original graph by mapping the distance to a scalar. For example, if ψ is learned as a decreasing function regarding the input $\text{SPD}(v_i, v_j)$, the obtained task-specific structure will result in stronger correlations among nodes that are close to each other on the original graph.

3.2 Learning Task-specific Structures from Labeled Nodes

With the proposed functions for edge weights, we still need to optimize these weights to obtain the task-specific structure for $\mathcal{T}$. In particular, we can leverage the label information inside labeled nodes (i.e., support nodes in each meta-task). Intuitively, the task-specific structure should ensure that the learned representations of nodes in the same class are similar, so that the classification of this class will be easier. According to Definition 1, larger node influence represents stronger correlations between nodes, which will increase the similarity between the learned representations. Therefore, we propose to enhance the node influence between support nodes in each class to optimize the

task-specific structure. However, directly calculating the Jacobian matrix $\|\partial \mathbf{h}_i / \partial \mathbf{h}_j\|$ can result in an excessively large computational cost, especially when the representations are high-dimensional [18]. Hence, instead of directly computing the node influence, we propose to enhance the node influence based on the following theorem:

Theorem 3.2 (Node Influence within Classes)**.** *Consider the expectation of node influence from the j-th node v_j to all nodes in its same class $\mathcal{C}$ on G, where $v_j \in \mathcal{C}$. Assume that $|\mathcal{C}| = K > 2$. The overall node influence is $\mathbb{E}\left(\sum_{v_i \in \mathcal{C} \setminus \{v_j\}} I(v_i, v_j)\right) = \sum_{v_i \in \mathcal{C} \setminus \{v_j\}} \frac{K-2}{n-1} \mathbf{h}_j^\top \cdot \mathbf{h}_i / \|\mathbf{h}_j\|^2 - \sum_{v_k \in \mathcal{V} \setminus \mathcal{C}} \frac{K-1}{n-1} \mathbf{h}_j^\top \cdot \mathbf{h}_k / \|\mathbf{h}_j\|^2$, where n is the number of nodes in G.*

The proof is deferred to Appendix B. Theorem 3.2 states that maximizing the node influence within one class is equivalent to maximizing the similarity within classes while minimizing similarity between classes. Specifically, the similarity is measured by the dot product between representations of v_j and other nodes. In this way, we can effectively optimize the edge weights by increasing the node influence within each class without a significant computational cost. The overall loss in meta-task $\mathcal{T}$ can be formulated as follows:

$$\mathcal{L}_N = -\frac{1}{N} \sum_{l=1}^{N} \sum_{v_j \in \mathcal{C}_l} \left(\sum_{v_i \in \mathcal{C}_l \setminus \{j\}} \frac{K-2}{|\mathcal{V}_\mathcal{T}| - 1} \mathbf{h}_j^\top \cdot \mathbf{h}_i / \|\mathbf{h}_j\|^2 - \sum_{v_k \in \mathcal{V} \setminus \mathcal{C}} \frac{K-1}{|\mathcal{V}_\mathcal{T}| - 1} \mathbf{h}_j^\top \cdot \mathbf{h}_k / \|\mathbf{h}_j\|^2 \right), \tag{4}$$

where $\mathcal{C}_l$ denotes the node set of the l-th support class in meta-task $\mathcal{T}$, and $|\mathcal{C}_l| = K$.

3.3 Learning Task-specific Structures from Unlabeled Nodes

Mutual Information. Although we have leveraged the information in support nodes by enhancing node influence, the unlabeled nodes (i.e., query nodes) in $G_\mathcal{T}$ remain unused. Thus, we further propose to utilize the query nodes to improve the learning of task-specific structures. Specifically, we leverage the concept of mutual information (MI) [19] by maximizing the MI between query nodes and their potential labels. Formally, the MI can be represented as follows:

$$\begin{aligned} \mathcal{L}_M := -\mathcal{I}(X_Q; Y_Q) &= -\mathcal{H}(Y_Q) + \mathcal{H}(Y_Q | X_Q) \\ &= \sum_{j=1}^{N} \bar{p}_j \log \bar{p}_j - \frac{1}{|\mathcal{Q}|} \sum_{i \in \mathcal{Q}} \sum_{j=1}^{N} p_{ij} \log p_{ij}, \end{aligned} \tag{5}$$

where $p_{ij} := P(y_i = j | \mathbf{x}_i)$ and $\bar{p}_j = \sum_{i \in \mathcal{Q}} p_{ij} / |\mathcal{Q}|$. $\mathcal{Q}$ is the query node set in meta-task $\mathcal{T}$. Here X_Q and Y_Q are the features and labels of query nodes, respectively. In particular, by maximizing the mutual information between query nodes and their labels, the learned task-specific structure can effectively leverage the information from the query set.

Transformed Markov Chain. Although the strategy of maximizing MI between X_Q and Y_Q has proven to be effective in few-shot learning, it remains non-trivial to utilize such a strategy for learning task-specific structures. Specifically, the crucial part is to model the dependency of classification probabilities (i.e., p_{ij}) on the learned task-specific structure. In this way, we can leverage the mutual information to help optimize the task-specific structure. To estimate the classification probability, we propose to utilize node influence inferred from the task-specific structure in each meta-task. Intuitively, we can assume that the higher node influence between two nodes can represent a larger probability that they share the same class. As a result, by estimating the node influence between a query node and a support node, we can optimize the task-specific structure based on Eq. (5). However, it still remains challenging to obtain an exact value of the node influence due to the potential computation cost of calculating the Jacobian matrix. Therefore, we propose to estimate node influence via the absorbing probability in a Markov chain [20]. Particularly, we create a Markov chain with absorbing states, where each state corresponds to a node on $G_\mathcal{T}$. Here the support nodes in $G_\mathcal{T}$ will be transformed into the absorbing states. Denoting the current adjacency matrix for $G_\mathcal{T}$ as $\mathbf{A}$, we obtain the transition matrix of the Markov chain from $\mathbf{A}$ as follows: (1) The transition probability from a support node to other nodes is 0 (i.e., the absorbing state). (2) Each entry is row-wise normalized to ensure that the sum of each row equals 1 (i.e., the transition probabilities starting from one node should sum up to 1). Denoting the obtained transition probability matrix as $\tilde{\mathbf{A}}$. Here we theoretically validate that the node influence between a query node and a support node can be effectively and efficiently estimated by the absorbing probability.

Theorem 3.3 (Absorbing Probabilities and Node Influence)**.** *Consider the Markov chain with a transition matrix $\tilde{\mathbf{A}}$ derived from a graph G. Denote the probability of being absorbed in the j-th state (absorbing state) when starting from the i-th state as $b_{i,j}$. Then, $I(v_i, v_j) = b_{i,j}$, where $I(v_i, v_j)$ is the node influence from the i-th node v_i to the j-th node v_j on G.*

The proof is provided in Appendix C. Theorem 3.3 provides that with the learned task-specific structure in meta-task $\mathcal{T}$, the node influence of a support node on a query node equals the absorbing probability starting from the query node to the support node in the Markov chain. In consequence, we can leverage the property of Markov chains to provide an efficient approximation for node influence.

Theorem 3.4 (Approximation of Absorbing Probabilities)**.** *Denote t as the number of non-absorbing states in the Markov chain, i.e., $t = |\mathcal{V}_\mathcal{T}| - |\mathcal{S}|$, where $|\mathcal{V}_\mathcal{T}|$ is the node set in $G_\mathcal{T}$. Denote $\mathbf{Q} \in \mathbb{R}^{t \times t}$ as the transition probability matrix for non-absorbing states in the Markov chain. Estimating $b_{i,j}$ with $\tilde{b}_{i,j} = \sum_{k=1}^{t} \tilde{\mathbf{A}}_{k,j} \cdot \sum_{h=0}^{m} \left(\mathbf{Q}^h\right)_{i,k}$, then the absolute error is upper bounded as $|b_{i,j} - \tilde{b}_{i,j}| \leq C/t^{m-1}(t-1)$, where m controls the upper bound, and C is a constant.*

The proof is provided in Appendix D. Theorem 3.4 indicates that the estimation error of node influence is upper bounded by $C/t^{m-1}(t-1)$, which is controlled by m. Therefore, we can adjust the value of m to fit in different application scenarios. With Theorem 3.3 and Theorem 3.4, we can estimate the value of the classification probability as follows:

$$P(y_i = j|\mathbf{x}_i) = \sum_{k=1}^{t} \tilde{\mathbf{A}}_{k,j} \cdot \sum_{h=0}^{m} (\mathbf{Q}^h)_{i,k} \tag{6}$$

As a result, we can optimize the objective in Eq. (5) according to the estimated node influence. In this way, we can adjust edge weights to reach a state that is tailored for this meta-task $\mathcal{T}$ and beneficial for the following classification of query nodes.

3.4 Meta Optimization

To effectively accumulate learned meta-knowledge across a variety of meta-tasks, we adopt the prevalent meta-optimization strategy based on MAML [21]. Generally, MAML aims at learning the unique update scheme based on each meta-task for fast adaptation to novel meta-tasks. Specifically, for each meta-task, we first perform several update steps to learn task-specific structures based on the two structure losses: $\theta_S^{(i)} = \theta_S^{(i-1)} - \alpha \nabla \mathcal{L}_S^{(i)}$, where θ_S denotes the parameters used to learn task-specific structures, and $\mathcal{L}_S^{(i)} = \mathcal{L}_N^{(i)} + \mathcal{L}_M^{(i)}$. During each update step, we leverage a GNN to learn node representations from the current task-specific structure, followed by an MLP layer to classify the support nodes: $\theta_G^{(i)} = \theta_G^{(i-1)} - \alpha \nabla \mathcal{L}_{support}^{(i)}$, where $\mathcal{L}_{support}^{(i)} = -\sum_{k \in \mathcal{S}} \sum_j y_{k,j} \log p_{k,j}^{(i)}$ is the cross-entropy loss, and α is the base learning rate. Here $y_{k,j} \in \{0, 1\}$ and $y_{k,j} = 1$ if the k-th node in $\mathcal{S}$ (i.e., the support set) belongs to the j-th support class in meta-task $\mathcal{T}$; otherwise $y_{k,j} = 0$. $p_{k,j}^{(i)}$ is the j-the entry of the class distribution vector $\mathbf{p}_k^{(i)}$ calculated by $\mathbf{p}_k^{(i)} = \text{Softmax}(\text{MLP}(\mathbf{h}_k^{(i)}))$, where $\mathbf{h}_k^{(i)}$ is extracted from the learned node representations $\mathbf{H}^{(i)} = \text{GNN}(\mathbf{A}^{(i)}, \mathbf{X})$. Here $\mathbf{A}^{(i)}$ is the edge weights during the i-th step. θ_G denotes the parameters of the GNN and the MLP classifier. After repeating these steps for η times, the loss on the query set will be used for the meta-update: $\theta_G = \theta_G^{(\eta)} - \beta_1 \nabla \mathcal{L}_{query}^{(\eta)}$ and $\theta_S = \theta_S^{(\eta)} - \beta_2 \nabla \mathcal{L}_S^{(\eta)}$, where β_1 and β_2 are the meta-learning rates, and $\mathcal{L}_{query}^{(\eta)}$ is the cross-entropy loss calculated on query nodes. For the meta-test tasks, we will use the final meta-updated parameters to evaluate the model.

4 Experiments

Experimental settings. The training process of our framework is conducted on a series of meta-training tasks. After training, the model will be evaluated on a specific number of meta-test tasks. Here we introduce four few-shot node classification settings for the experiments. (1) Shared graphs with disjoint labels. Under this setting, the meta-training and meta-test tasks are sampled from the same graph set. The labels in meta-test tasks are disjoint from those in meta-training tasks. (2) Disjoint graphs with shared labels. Under this setting, the meta-training tasks and meta-test

tasks are from disjoint graph sets. In other words, the graphs in meta-test tasks will be new graphs unseen during training. The labels are shared during training and test, which slightly decrease the difficulty. (3) Disjoint graphs with disjoint labels. Under this setting, both the graph set and the label set of meta-test tasks are disjoint from those in meta-training tasks. That being said, the model is required to handle new labels on graphs unseen during training, which renders the highest difficulty. (4) Single graph with disjoint labels. This setting is widely adopted in existing graph few-shot learning methods [9, 22, 10]. Under this setting, all nodes will be sampled from the same graph. Note that setting (1) and setting (2) are proposed in G-Meta [1], while setting (3) is a new setting introduced in this paper. Detailed experimental and parameter settings are provided in Appendix E.

Datasets. To evaluate the performance of GLITTER on the few-shot node classification task, we conduct experiments on five prevalent real-world graph datasets, two for the multiple-graph setting and three for the single-graph setting. The detailed statistics are provided in Table 1. For the multiple-graph setting, we use the following two datasets: (1) Tissue-PPI [23] consists of 24 protein-protein interaction (PPI) networks obtained from different types of tissues. Specifically, node features are gene signatures, and labels are gene ontology functions. This dataset consists of ten binary classification tasks. (2) Fold-PPI [1] contains of 144 tissue PPI networks, where the node labels are protein structures in SCOP database [24]. Moreover, the node features are conjoint triad protein descriptors [25]. There are 29 classes in this dataset. For the single-graph setting, we leverage three datasets: (1) DBLP [14] is a citation network with nodes and edges representing papers and citation relations, respectively. The node features are based on the paper abstracts, and the labels are determined by the paper venues. This dataset consists of 137 classes. (2) Cora-full [26] is also a citation network which contains 70 classes. (3) ogbn-arxiv [27] is a citation network that consists of 20 classes and significantly more nodes than other datasets.

Table 1: Statistics of five node classification datasets.

Dataset	# Graphs	# Nodes	# Edges	# Features	# Class
`Tissue-PPI`	24	51,194	1,350,412	50	10
`Fold-PPI`	144	274,606	3,666,563	512	29
`DBLP`	1	40,672	288,270	7,202	137
`Cora-full`	1	19,793	65,311	8,710	70
`ogbn-arxiv`	1	169,343	1,166,243	128	40

Baseline methods. (1) KNN [28] trains a GNN based on all samples in training graphs to learn node representations. Then it classifies query samples according to the labels in the support set of each meta-task. (2) ProtoNet [17] learns the prototype of each class by averaging its node representations. Then it classifies query samples based on their distances to the prototypes. (3) MAML [21] adopts the meta-learning strategy and learns node representations via GNNs. (4) Meta-GNN combines MAML with Simple Graph Convolution (SGC) [29]. (5) G-Meta [1] learns node representations from extracted local subgraphs and further adopts MAML for meta-learning. (6) GPN [9] learns prototypes based on node importance and classifies query nodes according to their distances to the learned prototypes. (7) RALE [11] learns node dependencies based on node locations on the graph. It is noteworthy that Meta-GNN, GPN, and RALE only work for the single-graph setting, while other baseline methods can be applied to both single-graph and multiple-graph settings.

4.1 Main Results

Table 2 and Table 3 present the performance comparison of our framework GLITTER and all baselines on few-shot node classification under the multiple-graph setting and the single-graph setting, respectively. Specifically, for the multiple-graph task, we conduct experiments under three settings (i.e., shared graphs with disjoint labels, disjoint graphs with shared labels, and disjoint graphs with disjoint labels). For the single-graph task, we choose two different few-shot settings to obtain a more comprehensive comparison: 5-way 3-shot (i.e., $N = 5$ and $K = 3$) and 10-way 3-shot (i.e., $N = 10$ and $K = 3$). For the evaluation metric, we utilize the average classification accuracy over 10 repetitions. From Table 2 and Table 3, we can obtain the follow observations: (1) Our framework GLITTER outperforms all other baselines in all multiple-graph and single-graph datasets. GLITTER also consistently achieves the best results under different settings, which validates the superiority of GLITTER on few-shot node classification problems. (2) When the multiple-graph setting is conducted with disjoint graphs, GLITTER has the least performance drop compared with

other baselines. This is because the learned task-specific structure is tailored for each meta-task and is thus more capable in the multiple-graph setting, where the structures in meta-tasks across graphs can vary greatly. (3) The performance improvement of GLITTER over other baselines is relatively larger on `Fold-PPI` and `DBLP`. This is because these datasets consist of more classes than other datasets under the same setting. Since the classes in different meta-tasks are more diversely distributed, the task-specific structures learned by GLITTER will provide better performance in this situation. (4) When the value of N increases (i.e., larger class set in each meta-task), the performance of all baselines drops significantly due to the higher classification difficulty. Nevertheless, GLITTER consistently outperforms the other baselines. The reason is that a larger class set in a meta-task will require more specific representations for classification due to the variety of classes.

Table 2: The few-shot node classification results (accuracy in %) under the multiple-graph setting.

Dataset	`Tissue-PPI`			`Fold-PPI`		
Graph Setting	Shared	Disjoint	Disjoint	Shared	Disjoint	Disjoint
Label Setting	Disjoint	Shared	Disjoint	Disjoint	Shared	Disjoint
KNN [28]	62.3±3.1	63.6±2.3	56.6±3.2	40.0±3.4	50.2±2.2	38.6±2.8
ProtoNet [17]	59.0±3.3	61.8±2.7	55.7±3.2	37.4±2.0	46.5±2.0	34.7±3.1
MAML [21]	63.8±4.3	68.4±3.7	56.0±4.0	41.0±3.0	49.3±2.8	42.5±4.4
G-Meta [1]	64.6±4.9	69.9±4.1	58.1±2.4	46.1±5.0	54.7±4.3	50.3±5.2
GLITTER	**69.7**±3.9	**73.1**±4.8	**60.3**±3.1	**53.3**±3.6	**61.3**±2.9	**54.7**±4.0

Table 3: The few-shot node classification results (accuracy in %) under the single-graph setting.

Dataset	`DBLP`		`Cora-full`		`ogbn-arxiv`	
Setting	5-way 3-shot	10-way 3-shot	5-way 3-shot	10-way 3-shot	5-way 3-shot	10-way 3-shot
KNN [28]	63.4±3.3	54.1±2.5	56.9±2.1	42.4±2.7	45.4±3.3	35.7±2.4
ProtoNet [17]	58.5±2.6	51.3±3.3	49.6±2.2	41.4±1.9	42.9±3.0	34.6±3.0
MAML [21]	60.4±4.5	52.3±3.5	54.0±4.4	38.1±3.7	43.0±3.1	34.1±3.4
Meta-GNN [10]	66.4±2.8	58.0±2.6	59.7±2.7	42.5±3.6	48.8±2.4	35.0±4.8
G-Meta [1]	73.1±3.4	57.5±5.0	63.7±4.0	47.6±4.9	50.8±4.8	38.2±4.9
GPN [9]	75.4±2.3	64.0±3.8	62.1±3.4	47.3±4.4	55.3±3.1	36.3±4.6
RALE [11]	74.9±3.2	65.1±5.7	63.4±4.0	45.2±3.6	54.7±3.9	38.4±5.1
GLITTER	**79.3**±3.1	**69.5**±5.2	**66.2**±4.4	**52.0**±2.6	**58.5**±4.7	**44.2**±3.8

4.2 Ablation Study

In this section, we perform a series of ablations studies to evaluate the effectiveness of different components in our framework GLITTER. The results are presented in Figure 2. Specifically, we compare our proposed framework GLITTER with three degenerate versions: (1) GLITTER without sampling relevant nodes (i.e., the task-specific structure only consists of support nodes and query nodes), denoted as GLITTER\S; (2) GLITTER without enhancing the node influence within classes, denoted as GLITTER\N; (3) GLITTER without maximizing the mutual information between query nodes and their potential labels, denoted as GLITTER\M. From Figure 2, we can obtain several meaningful observations. First, our framework consistently outperforms all variants with certain components removed, which demonstrates that each module in GLITTER plays a crucial role in learning task-specific structures. Second, removing the relevant nodes deteriorates the performance under the multiple-graph setting with disjoint graphs more than that on shared graphs. The reason is meta-knowledge from training graphs can be uninformative for a novel graph during evaluation. As a result, the relevant nodes on the novel graph become more crucial for learning task-specific structures. Third, the performance decreases differently by removing the node influence module or the mutual information module. Specifically, removing node influence generally causes a more significant drop when N increases. This is due to that a larger class set in each meta-task requires more distinct node representations in different classes. Therefore, enhancing the node influence can provide better performance by enlarging correlations within classes.

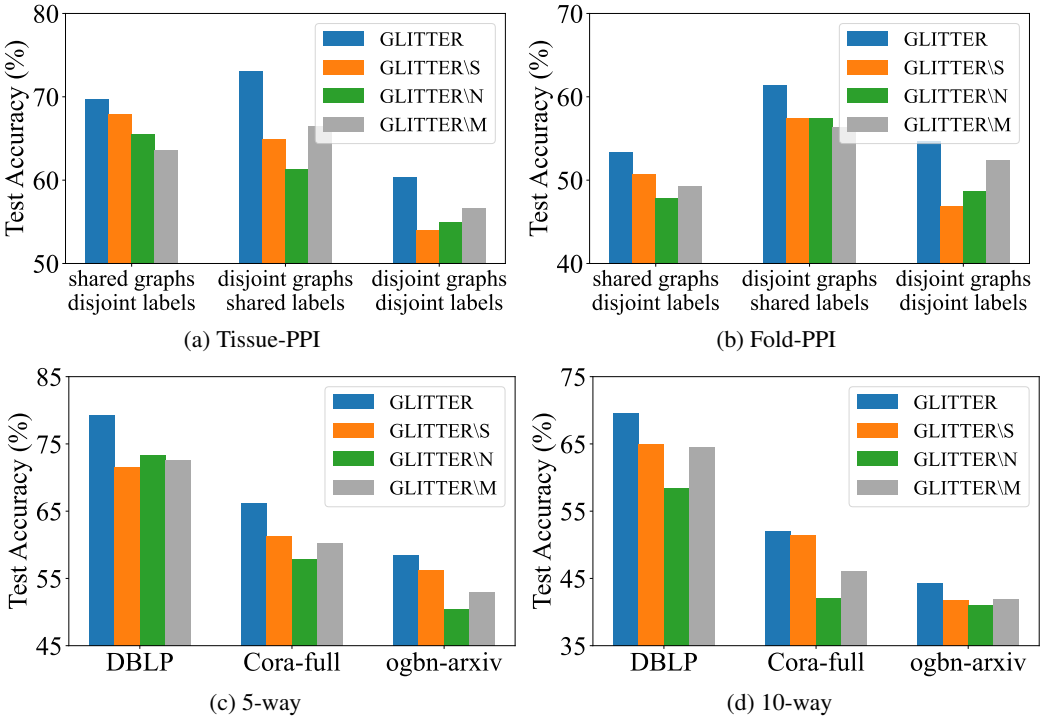

Figure 2: The ablation study of GLITTER on both single-graph and mutiple-graph settings.

5 Related Work

Few-shot Learning on Graphs. Few-shot learning aims at effectively solving tasks with scarce labeled samples as references. The general approach is to learn meta-knowledge from tasks with abundant labeled samples and generalize it to a new task with few labeled samples. Few-shot learning frameworks generally adopt the meta-learning strategy, which means both training and test processes will be conducted on a series of learning tasks. Existing few-shot learning methods can be broadly divided into two categories: (1) *Metric-based* approaches, which aim at learning a metric function between the query set and the support set for classification [16, 30, 31, 17]. (2) *Optimization-based* approaches, which focus on updating model parameters according to gradients calculated on the support set in each meta-task [32, 33, 21, 34]. Recently, many works propose to incorporate existing few-shot learning frameworks into graph few-shot learning problems [11, 22, 12, 35, 36, 37, 38]. For example, GPN [9] leverages the importance of nodes and combines Prototypical Networks [17] to solve few-shot node classification problem. G-Meta [1] extracts local subgraphs for nodes for fast adaptations across multiple graphs.

Graph Neural Networks. Recently, many researchers have focused on using Graph Neural Networks (GNNs) to learn effective representation for nodes on graphs [39, 5, 40, 41, 42, 43, 40, 44]. With GNNs, models will gradually learn comprehensive node representations based on the specific downstream tasks [45]. In particular, GNNs learn node representations from neighbor nodes via an information aggregation mechanism. As a result, GNNs can incorporate both structural information and node feature information into the representation learning process. For example, Graph Convolutional Networks (GCNs) [8] leverage a first-order approximation to learn node representations with graph convolution layers in a simplified manner. Graph Attention Networks (GATs) [6] learn adjustable weights for neighbors nodes via the attention mechanism to aggregate features of neighboring nodes in an adaptive manner.

6 Conclusion

In this paper, we propose to learn task-specific structures for each meta-task in graph few-shot learning, so that the learned node representations will be tailored for each meta-task. To handle the variety of nodes among meta-tasks, we propose an innovative framework, GLITTER, to extract important

nodes and learn task-specific structures based on node influence and mutual information. As a result, our framework can adaptively learn node representations to promote classification performance in each meta-task. We conduct extensive experiments on five node classification datasets under both the single-graph and multiple-graph settings, and the results validate the superiority of our framework over other state-of-the-art baselines.

Acknowledgement

This work is supported by the National Science Foundation under grants IIS-2006844, IIS-2144209, IIS-2223769, CNS-2154962, and BCS-2228534, the JP Morgan Chase Faculty Research Award, the Cisco Faculty Research Award, and the subcontract JSA-22-D0311 from Jefferson Lab.

References

[1] Kexin Huang and Marinka Zitnik. Graph meta learning via local subgraphs. In *NeurIPS*, 2020.

[2] Chuxu Zhang, Huaxiu Yao, Chao Huang, Meng Jiang, Zhenhui Li, and Nitesh V Chawla. Few-shot knowledge graph completion. In *AAAI*, 2019.

[3] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*, 2018.

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

[5] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, 2018.

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

[7] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 2017.

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

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

[10] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. Meta-gnn: On few-shot node classification in graph meta-learning. In *CIKM*, 2019.

[11] Zemin Liu, Yuan Fang, Chenghao Liu, and Steven CH Hoi. Relative and absolute location embedding for few-shot node classification on graph. In *AAAI*, 2021.

[12] Kaize Ding, Jundong Li, Nitin Agarwal, and Huan Liu. Inductive anomaly detection on attributed networks. In *IJCAI*, 2020.

[13] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 2005.

[14] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: extraction and mining of academic social networks. In *SIGKDD*, 2008.

[15] Jatin Chauhan, Deepak Nathani, and Manohar Kaul. Few-shot learning on graphs via super-classes based on graph spectral measures. In *ICLR*, 2020.

[16] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *NeurIPS*, 2016.

[17] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, 2017.

[18] Hongwei Wang and Jure Leskovec. Unifying graph convolutional neural networks and label propagation. *arXiv:2002.06755*, 2020.

[19] José Dolz, Pablo Piantanida, and Ismail Ben Ayed. Transductive information maximization for few-shot learning. In *NeurIPS*, 2020.

[20] Luke Tierney. Markov chains for exploring posterior distributions. *the Annals of Statistics*, pages 1701–1728, 1994.

[21] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[22] Ning Wang, Minnan Luo, Kaize Ding, Lingling Zhang, Jundong Li, and Qinghua Zheng. Graph few-shot learning with attribute matching. In *CIKM*, 2020.

[23] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 2017.

[24] Antonina Andreeva, Eugene Kulesha, Julian Gough, and Alexey G Murzin. The scop database in 2020: expanded classification of representative family and superfamily domains of known protein structures. *Nucleic acids research*, 2020.

[25] Juwen Shen, Jian Zhang, Xiaomin Luo, Weiliang Zhu, Kunqian Yu, Kaixian Chen, Yixue Li, and Hualiang Jiang. Predicting protein–protein interactions based only on sequences information. *PNAS*, 2007.

[26] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *ICLR*, 2018.

[27] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*, 2020.

[28] Sahibsingh A Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 1976.

[29] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

[30] Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. Learning to propagate for graph meta-learning. *NeurIPS*, 2019.

[31] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: relation network for few-shot learning. In *CVPR*, 2018.

[32] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *ICLR*, 2018.

[33] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017.

[34] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv:1803.02999*, 2018.

[35] Zhen Tan, Kaize Ding, Ruocheng Guo, and Huan Liu. Graph few-shot class-incremental learning. In *WSDM*, 2022.

[36] Wenhan Xiong, Mo Yu, Shiyu Chang, Xiaoxiao Guo, and William Yang Wang. One-shot relational learning for knowledge graphs. In *EMNLP*, 2018.

[37] Mingyang Chen, Wen Zhang, Wei Zhang, Qiang Chen, and Huajun Chen. Meta relational learning for few-shot link prediction in knowledge graphs. In *EMNLP*, 2019.

[38] Song Wang, Xiao Huang, Chen Chen, Liang Wu, and Jundong Li. Reform: Error-aware few-shot knowledge graph completion. In *CIKM*, 2021.

[39] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. Heterogeneous network embedding via deep architectures. In *SIGKDD*, 2015.

[40] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE TNNLS*, 2020.

[41] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, 2016.

[42] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *ICLR*, 2019.

[43] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *SIGKDD*, 2019.

[44] Kaize Ding, Zhe Xu, Hanghang Tong, and Huan Liu. Data augmentation for deep graph learning: A survey. *arXiv:2202.08235*, 2022.

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

[46] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 2020.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Appendix F.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Appendix F.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Appendix A.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A, B, C, and D.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See the supplemental materials.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix E
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Table 2 and Table 3.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix E

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 4.
   (b) Did you mention the license of the assets? [Yes] See Appendix E.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] The source code of our proposed framework is included in the supplemental materials.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] The datasets are developed based on public data by the corresponding authors.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The data used in this paper does not contain identifiable information or offensive content.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Theorem 3.1 and Proof

Before proving Theorem 3.1, we first provide a lemma that demonstrate the lower bound of node influence between two nodes. In the following proof, we will follow [1] and [5] to use GCNs [8] as the exemplar GNN for simplicity. Moreover, it is noteworthy that our proof can be naturally generalized to different types of GNNs (e.g., GAT [6] and GraphSAGE [45]) by assigning different values for edge weights. Specifically, the l-th layer propagation process can be represented as $\mathbf{H}^{(l+1)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)})$. Here $\mathbf{H}^{(l)}$ and $\mathbf{W}^{(l)}$ denote the node representation and weight parameter matrices, respectively. $\hat{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$ is the adjacency matrix after row normalization, which means each row of $\hat{\mathbf{A}}$ sums up to 1. Following [1], [18], and [5], we set σ as an identity function and $\mathbf{W}$ an identity matrix. Moreover, we assume that the propagation process is conducted in a sufficient number of steps. As a result, the output representation of one node can be represented by representations of its neighbor nodes.

Lemma A.1. *Consider the expectation of node influence between node v_i and node v_j, i.e., $\mathbb{E}\left(I(v_i, v_j)\right)$. Assume that the node degrees are distributed uniformly for each node with the mean value $\bar{d}$. Then, $\mathbb{E}\left(I(v_i, v_j)\right) \geq d^{(SPD(v_i, v_j))}$, where $SPD(v_i, v_j)$ is the shortest path dsitance between v_i and v_j.*

Proof. Based on the GCN propagation strategy, we know that on G, the representation of node v_i can be represented as

$$\mathbf{h}_i = \frac{1}{D_{ii}} \sum_{k \in \mathcal{N}(i)} a_{ik} \mathbf{h}_k,$$

where $\mathcal{N}(i)$ denotes the set of neighbor nodes of node v_i. Then we can expand the equation by incorporating 2-hop neighbors of node v_i:

$$\mathbf{h}_i = \frac{1}{D_{ii}} \sum_{k \in \mathcal{N}(i)} a_{ik} \frac{1}{D_{kk}} \sum_{l \in \mathcal{N}(k)} a_{kl} \mathbf{h}_l.$$

In this way, the expectation of node influence $I_{i,j} = \|\partial \mathbf{h}_i / \partial \mathbf{h}_j\|$ can be represented as:

$$
\begin{aligned}
\left\|\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_j}\right\| &= \left\|\frac{\partial}{\partial \mathbf{h}_j}\left(\frac{1}{D_{ii}}\sum_{k\in\mathcal{N}(i)} a_{ik}\frac{1}{D_{kk}}\sum_{l\in\mathcal{N}(k)} a_{kl}\cdots\frac{1}{D_{mm}}\sum_{o\in\mathcal{N}(m)} a_{mo}\mathbf{h}_o\right)\right\| \\
&= \left\|\frac{\partial}{\partial \mathbf{h}_j}\left(\left(\frac{1}{D_{ii}}a_{ik_1^1}\frac{1}{D_{k_1^1k_1^1}}a_{k_1^1k_2^1}\cdots\frac{1}{D_{k_{n_1}^1k_{n_1}^1}}a_{k_{n_1}^1 j}\mathbf{h}_j\right)\right. \\
&\quad \left.+\cdots+\left(\frac{1}{D_{ii}}a_{ik_1^m}\frac{1}{D_{k_1^mk_1^m}}a_{k_1^mk_2^m}\cdots\frac{1}{D_{k_{nm}^mk_{nm}^m}}a_{k_{nm}^m j}\mathbf{h}_j\right)\right)\right\|.
\end{aligned}
\tag{7}
$$

In the above equation, we first substitute the term $\mathbf{h}_i$ by the iterative expansion of neighbors. In this expansion, we only keep m paths that start from v_i to v_j, where n_i is the number of intermediate nodes on the i-th path. The reason is that since we are considering the gradient between v_i and v_j, the derivative on paths that do not contain v_j will be 0 and thus can be ignored. Then we can further extract the common term $\|\partial\mathbf{h}_j/\partial\mathbf{h}_j\|$:

$$
\begin{aligned}
\left\|\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_j}\right\| &= \left\|\frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_j}\right\|\cdot\left(\left(\frac{1}{D_{ii}}a_{ik_1^1}\frac{1}{D_{k_1^1k_1^1}}a_{k_1^1k_2^1}\cdots\frac{1}{D_{k_{n_1}^1k_{n_1}^1}}a_{k_{n_1}^1 j}\right)\right. \\
&\quad \left.+\cdots+\left(\frac{1}{D_{ii}}a_{ik_1^m}\frac{1}{D_{k_1^mk_1^m}}a_{k_1^mk_2^m}\cdots\frac{1}{D_{k_{nm}^mk_{nm}^m}}a_{k_{nm}^m j}\right)\right) \\
&= \left(\frac{1}{D_{ii}}a_{ik_1^1}\frac{1}{D_{k_1^1k_1^1}}a_{k_1^1k_2^1}\cdots\frac{1}{D_{k_{n_1}^1k_{n_1}^1}}a_{k_{n_1}^1 j}\right) \\
&\quad +\cdots+\left(\frac{1}{D_{ii}}a_{ik_1^m}\frac{1}{D_{k_1^mk_1^m}}a_{k_1^mk_2^m}\cdots\frac{1}{D_{k_{nm}^mk_{nm}^m}}a_{k_{nm}^m j}\right).
\end{aligned}
\tag{8}
$$

In this equation, we first utilize $\|partial\mathbf{h}_j/\partial\mathbf{h}_j\| = 1$. This is because $\|\partial\mathbf{h}_j/\partial\mathbf{h}_j\| = \|\mathbf{I}\| = \sup_{\|\mathbf{h}\|=1}\{\|\mathbf{I}\mathbf{h}\|\} = 1$. The resulting term is the expectation that sums up the node degree products of all paths between v_i and v_j. Therefore, it is larger than the value on the path with the maximum node degree product:

$$
\begin{aligned}
\left\|\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_j}\right\| &\geq \max\left(\left(\frac{1}{D_{ii}}a_{ik_1^1}\frac{1}{D_{k_1^1k_1^1}}a_{k_1^1k_2^1}\cdots\frac{1}{D_{k_{n_1}^1k_{n_1}^1}}a_{k_{n_1}^1 j}\right)\right. \\
&\quad \left.,\cdots,\left(\frac{1}{D_{ii}}a_{ik_1^m}\frac{1}{D_{k_1^mk_1^m}}a_{k_1^mk_2^m}\cdots\frac{1}{D_{k_{nm}^mk_{nm}^m}}a_{k_{nm}^m j}\right)\right). \\
&= \frac{1}{D_{ii}}a_{ik_1^*}\frac{1}{D_{k_1^*k_1^*}}a_{k_1^*k_2^*}\cdots\frac{1}{D_{k_{n_*}^*k_{n_*}^*}}a_{k_{n_*}^* j}.
\end{aligned}
\tag{9}
$$

If we assume the node degrees are uniformly distributed, then the expectation of node degree products on this path p_* is $d^{(n_*+1)}$, where n_*+1 is the length, and $\bar{d}$ is the expectation of node degrees. Moreover, we know p_* is exactly the shortest path between v_i and v_j. Therefore,

$$
\mathbb{E}\left(\left\|\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_j}\right\|\right) \geq \left(1/\bar{d}\right)^{(n_*+1)} = \bar{d}^{-(\mathrm{SPD}(v_i,v_j))},
\tag{10}
$$

where $\mathrm{SPD}(v_i, v_j)$ denotes the shortest path distance between node v_i and node v_j. □

Now with Lemma A.1, we can prove Theorem 3.1.

Theorem 3.1. *Consider the node influence from node v_k to the i-th class (i.e., C_i) in a meta-task $\mathcal{T}$. Denote the geometric mean of the node influence values to all support nodes in C_i as $I_{C_i}(v_k) = \sqrt[K]{\prod_{j=1}^{K} I(v_k, s_{i,j})}$, where $s_{i,j}$ is the j-th support node in C_i. Assume the node degrees are randomly distributed with the mean value as $\bar{d}$. Then, $\mathbb{E}(\log I_{C_i}(v_k)) \geq -\log\bar{d}\cdot\sum_{j=1}^{K} SPD(v_k, s_{i,j})/K$, where $SPD(v_k, s_{i,j})$ denotes the shortest path distance between v_k and $s_{i,j}$.*

Proof. We know $\log I_C(v_k)$ can be represented as follows:

$$\log I_C(v_k) = \frac{1}{K}\sum_{j=1}^{K} \log I(v_k, s_{i,j}) \tag{11}$$

Based on Lemma A.1, we know:

$$\mathbb{E}\left(\log I_C(v_k)\right) = \frac{1}{K}\sum_{j=1}^{K} \log \mathbb{E}\left(I(v_k, s_{i,j})\right) \geq \frac{1}{K}\sum_{j=1}^{K} -\text{SPD}(v_k, s_{i,j}) \cdot \log \bar{d}, \tag{12}$$

where $\text{SPD}(v_i, v_j)$ denotes the shortest path distance between node v_i and node v_j. By rearranging the term, we can obtain the final inequality:

$$\mathbb{E}\left(\log I_C(v_k)\right) \geq -\frac{\log \bar{d}}{K}\sum_{j=1}^{K} \text{SPD}(v_k, s_{i,j}). \tag{13}$$

□

B Theorem 3.2 and Proof

Theorem 3.2 (Node Influence within Classes)**.** *Consider the expectation of node influence from the j-th node v_j to all nodes in its same class $\mathcal{C}$ on G, where $v_j \in \mathcal{C}$. Assume that $|\mathcal{C}| = K > 2$. The overall node influence is $\mathbb{E}\left(\sum_{v_i \in \mathcal{C}\setminus\{v_j\}} I(v_i, v_j)\right) = \sum_{v_i \in \mathcal{C}\setminus\{v_j\}} \frac{K-2}{n-1}\mathbf{h}_j^\top \cdot \mathbf{h}_i/\|\mathbf{h}_j\|^2 - \sum_{v_k \in \mathcal{V}\setminus\mathcal{C}} \frac{K-1}{n-1}\mathbf{h}_j^\top \cdot \mathbf{h}_k/\|\mathbf{h}_j\|^2$, where n is the number of nodes in G.*

Proof. Following the expansion of node representations based on the neighbors in Lemma A.1, we can represent $\mathbf{h}_i$ as follows:

$$\mathbf{h}_i = \sum_{k=1}^{n} I_{ik}\mathbf{h}_k, \tag{14}$$

where I_{ik} is node influence from v_i to v_k. Then, we have

$$\begin{aligned}
\mathbf{h}_j^\top \cdot \mathbf{h}_i/\|\mathbf{h}_j\|^2 &= \sum_{k=1}^{n} I_{ik}\mathbf{h}_j^\top \cdot \mathbf{h}_k/\|\mathbf{h}_j\|^2 \\
&= I_{ij}\mathbf{h}_j^\top \cdot \mathbf{h}_j/\|\mathbf{h}_j\| + \sum_{k=1,k\neq j}^{n} I_{ik}\mathbf{h}_j^\top \cdot \mathbf{h}_k/\|\mathbf{h}_j\|^2 \\
&= I_{ij} + \sum_{k=1,k\neq j}^{n} I_{ik}\mathbf{h}_j^\top \cdot \mathbf{h}_k/\|\mathbf{h}_j\|^2.
\end{aligned} \tag{15}$$

We can further sum up the node influence from nodes in the same class of v_j to it since we aim to calculate the total node influence of a set of support nodes. Here we denote the support nodes set of v_j class as $\mathcal{C}$, where $|\mathcal{C}| = K$. Therefore, we can obtain:

$$\sum_{i\in\mathcal{C}\setminus\{j\}} \mathbf{h}_j^\top \cdot \mathbf{h}_i/\|\mathbf{h}_j\|^2 = \sum_{i\in\mathcal{C}\setminus\{j\}} I_{ij} + \sum_{i\in\mathcal{C}\setminus\{j\}} \sum_{k\in\mathcal{V}\setminus(\{i\}\cup\{j\})} I_{ik}\mathbf{h}_j^\top \cdot \mathbf{h}_k/\|\mathbf{h}_j\|^2 \tag{16}$$

We can move the same terms from the RHS to the LHS:

$$\begin{aligned}
&\sum_{i\in\mathcal{C}\setminus\{j\}} \mathbf{h}_j^\top \cdot \mathbf{h}_i/\|\mathbf{h}_j\|^2 - \sum_{i\in\mathcal{C}\setminus\{j\}} \sum_{k\in\mathcal{C}\setminus\{i\}} I_{ik}\mathbf{h}_j^\top \cdot \mathbf{h}_k/\|\mathbf{h}_j\|^2 \\
&= \sum_{i\in\mathcal{C}\setminus\{j\}} I_{ij} + \sum_{i\in\mathcal{C}\setminus\{j\}} \sum_{k\in\mathcal{V}\setminus\mathcal{C}} I_{ik}\mathbf{h}_j^\top \cdot \mathbf{h}_k/\|\mathbf{h}_j\|^2
\end{aligned} \tag{17}$$

Rearranging the LHS, we can obtain:

$$LHS = \sum_{i \in \mathcal{C} \setminus \{j\}} \left(1 - \sum_{k \in \mathcal{C} \setminus (\{i\} \cup \{j\})} a_{ki} \right) \mathbf{h}_j^\top \cdot \mathbf{h}_i / \|\mathbf{h}_j\|^2 \tag{18}$$

Rearranging the RHS, we can obtain:

$$RHS = \sum_{i \in \mathcal{C} \setminus \{j\}} I_{ij} + \sum_{k \in \mathcal{V} \setminus \mathcal{C}} \sum_{i \in \mathcal{C} \setminus \{j\}} I_{ik} \mathbf{h}_j^\top \cdot \mathbf{h}_k / \|\mathbf{h}_j\|^2 \tag{19}$$

We further assume that the expectation of each I_{ij} is $1/(n-1)$ since the sum of probabilities of all paths starting from v_i equals 1. Therefore,

$$\begin{aligned} \mathbb{E}\left(\sum_{i \in \mathcal{C} \setminus \{j\}} I_{ij} \right) &= \mathbb{E}\left(\sum_{i \in \mathcal{C} \setminus \{j\}} \left(1 - \sum_{k \in \mathcal{C} \setminus (\{i\} \cup \{j\})} a_{ki} \right) \mathbf{h}_j^\top \cdot \mathbf{h}_i / \|\mathbf{h}_j\|^2 \right. \\ &\quad \left. - \sum_{k \in \mathcal{V} \setminus \mathcal{C}} \sum_{i \in \mathcal{C} \setminus \{j\}} I_{ik} \mathbf{h}_j^\top \cdot \mathbf{h}_k / \|\mathbf{h}_j\|^2 \right) \\ &= \sum_{i \in \mathcal{C} \setminus \{j\}} \frac{K-2}{n-1} \mathbf{h}_j^\top \cdot \mathbf{h}_i / \|\mathbf{h}_j\|^2 - \sum_{k \in \mathcal{V} \setminus \mathcal{C}} \frac{K-1}{n-1} \mathbf{h}_j^\top \cdot \mathbf{h}_k / \|\mathbf{h}_j\|^2 \end{aligned} \tag{20}$$

□

C Theorem 3.3 and Proof

Theorem 3.3 (Absorbing Probabilities and Node Influence)**.** *Consider the Markov chain with a transition matrix $\tilde{\mathbf{A}}$ derived from a graph G. Denote the probability of being absorbed in the j-th state (absorbing state) when starting from the i-th state as $b_{i,j}$. Then, $I(v_i, v_j) = b_{i,j}$, where $I(v_i, v_j)$ is the node influence from the j-th node v_i to the i-th node v_j on G.*

Proof. Following Lemma A.1, we can obtain:

$$\begin{aligned} \left\| \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_j} \right\| &= \left\| \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_j} \right\| \cdot \left(\left(\frac{1}{D_{ii}} a_{ik_1^1} \frac{1}{D_{k_1^1 k_1^1}} a_{k_1^1 k_2^1} \cdots \frac{1}{D_{k_{n_1}^1 k_{n_1}^1}} a_{k_{n_1}^1 j} \right) \right. \\ &\quad \left. + \cdots + \left(\frac{1}{D_{ii}} a_{ik_1^m} \frac{1}{D_{k_1^m k_1^m}} a_{k_1^m k_2^m} \cdots \frac{1}{D_{k_{n_m}^m k_{n_m}^m}} a_{k_{n_m}^m j} \right) \right) \\ &= \left(\frac{1}{D_{ii}} a_{ik_1^1} \frac{1}{D_{k_1^1 k_1^1}} a_{k_1^1 k_2^1} \cdots \frac{1}{D_{k_{n_1}^1 k_{n_1}^1}} a_{k_{n_1}^1 j} \right) \\ &\quad + \cdots + \left(\frac{1}{D_{ii}} a_{ik_1^m} \frac{1}{D_{k_1^m k_1^m}} a_{k_1^m k_2^m} \cdots \frac{1}{D_{k_{n_m}^m k_{n_m}^m}} a_{k_{n_m}^m j} \right). \end{aligned} \tag{21}$$

As illustrated in Lemma A.1, $\|\partial \mathbf{h}_j / \partial \mathbf{h}_j\| = 1$ because $\|\partial \mathbf{h}_j / \partial \mathbf{h}_j\| = \|\mathbf{I}\| = \sup_{\|\mathbf{h}\|=1} \{\|\mathbf{I}\mathbf{h}\|\} = 1$. On the other hand, utilizing the total probability law and the properties of Markov chains, we know

$$b_{i,j} = \sum_{k=1}^{n} b_{k,j} p_{ik},$$

where p_{ik} is the (i, j)-entry of $\mathbf{P}$. The reason is that assuming the current state is i, we know the next state will be k with probability p_{ik}. Then we can iteratively expand the expression as follows:

$$
\begin{aligned}
b_{i,j} &= \sum_{k=1}^{n} p_{ik} \sum_{l=1}^{n} p_{kl} \cdots \sum_{o=1}^{n} p_{mo} b_{o,j} \\
&= \Big(\Big(p_{ik_1^1} p_{k_1^1 k_2^1} \cdots p_{k_{n_1}^1 j} b_{j,j} \Big) \\
&+ \cdots + \Big(p_{ik_1^m} p_{k_1^m k_2^1} \cdots p_{k_{n_m}^m j} b_{j,j} \Big) \Big) \\
&= \Big(\Big(p_{ik_1^1} p_{k_1^1 k_2^1} \cdots p_{k_{n_1}^1 j} \Big) \\
&+ \cdots + \Big(p_{ik_1^m} p_{k_1^m k_2^1} \cdots p_{k_{n_m}^m j} \Big) \Big) .
\end{aligned} \tag{22}
$$

In the above equation, we apply the similar expansion idea as in Eq. (7). The result is obtained by accumulating all products of transition probabilities on all possible paths from state i to state j. It is noteworthy that there are multiple absorbing states on this Markov chain. However, we ignore the paths absorbed in other absorbing states, since the corresponding absorbing probability $b_{k,j}$ is 0, where k and j are two absorbing states. In this way, we further incorporate $b_{j,j} = 1$ and obtain the final result. Considering Eq. (21) and Eq. (22), we can find that by setting $p_{ik} = a_{ik}/D_{ii}$, we can obtain $\|\partial \mathbf{h}_i / \partial \mathbf{h}_j\| = b_{i,j}$. Moreover, we know $\sum_{k=1}^{n} p_{ik} = \sum_{k=1}^{n} a_{ik}/D_{ii} = 1, i = 1, 2, \ldots, t$. Therefore, these transition probabilities satisfy the requirements of Markove chains, which completes the proof. □

D Theorem 3.4 and Proof

Theorem 3.4 (Approximation of Absorbing Probabilities)**.** *Denote t as the number of non-absorbing states in the Markov chain, i.e., $t = |\mathcal{V}_\mathcal{T}| - |\mathcal{S}|$, where $|\mathcal{V}_\mathcal{T}|$ is the node set in $G_\mathcal{T}$. Denote $\mathbf{Q} \in \mathbb{R}^{t \times t}$ as the transition probability matrix for non-absorbing states in the Markov chain. Estimating $b_{i,j}$ with $\tilde{b}_{i,j} = \sum_{k=1}^{t} \tilde{\mathbf{A}}_{k,j} \cdot \sum_{h=0}^{m} \left(\mathbf{Q}^h\right)_{i,k}$, then the absolute error is upper bounded as $|b_{i,j} - \tilde{b}_{i,j}| \leq C/t^{m-1}(t-1)$, where m controls the upper bound, and C is a constant.*

Proof. Considering the transition probability matrix $\mathbf{P}$ as

$$
\mathbf{P} = \begin{pmatrix} \mathbf{Q} & \mathbf{R} \\ \mathbf{0} & \mathbf{I}_r \end{pmatrix},
$$

where $\mathbf{Q} \in \mathbb{R}^{t \times t}$ and $\mathbf{R} \in \mathbb{R}^{t \times r}$. $\mathbf{0}$ is a $t \times r$ zero matrix, and $\mathbf{I}_t$ is an $r \times r$ identity matrix. Basically, the absorbing probability $b_{i,j}$ (i.e., the probability of being absorbed in the j-th state when starting from the i-th state) can be represented as

$$
\begin{aligned}
b_{i,j} &= \sum_{k=1}^{t} P(X_{t+1} = j | X_t = k) \cdot \mathbb{E}(N(k) | X_0 = i) \\
&= \sum_{k=1}^{t} p_{kj} \cdot \sum_{h=0}^{\infty} \left(\mathbf{Q}^h\right)_{i,k},
\end{aligned} \tag{23}
$$

where $N(k)$ is the umber of visits to state k. Nevertheless, directly calculating $\sum\limits_{h=0}^{\infty} \mathbf{Q}^h$ can be inefficient. Thus, we propose to estimate it by the sum of the first h values. Specifically, we know $\left(\mathbf{Q}^h\right)_{ik}$ is the probability that state i transitions to state j in exactly h steps. Therefore,

$$
\begin{aligned}
\left(\mathbf{Q}^h\right)_{ik} &= \sum_{k_1=1}^{t} p_{ik_1} \sum_{k_2=1}^{t} p_{k_1 k_2} \cdots \sum_{k_h=1}^{t} p_{k_h k} \\
&= \Big(\Big(p_{ik_1^1} p_{k_1^1 k_2^1} \cdots p_{k_h^1 j} \Big) + \cdots + \Big(p_{ik_1^m} p_{k_1^m k_2^1} \cdots p_{k_h^m j} \Big) \Big),
\end{aligned}
$$

which is the sum of transition probability products on all possible paths of length h from state i to state j. Furthermore,

$$\begin{aligned}\left(\mathbf{Q}^h\right)_{ik} &\leq C * \max\left(\left(p_{ik_1^1}p_{k_1^1k_2^1}\cdots p_{k_h^1k}\right)\right.\\ &\left.,\cdots,\left(p_{ik_1^m}p_{k_1^mk_2^1}\cdots p_{k_h^mk}\right)\right)\\ &= C * \left(p_{ik_1^*}p_{k_1^*k_2^*}\cdots p_{k_h^*k}\right)\\ &= C * \left(\sqrt[h]{p_{ik_1^*}p_{k_1^*k_2^*}\cdots p_{k_h^*k}}\right)^h\\ &= C * \left(\mathrm{GM}(p_{ik^*}^{(h)})\right)^h,\end{aligned} \tag{24}$$

where C is the extracted constant, and $\mathrm{GM}(p_{ik^*}^{(h)})$ denotes the geometric mean of the path with the transition probability products. Based on the inequality of arithmetic and geometric means (i.e., the AM-GM inequality), we know the geometric mean is always less than or equal to the arithmetic mean. Therefore, $\mathrm{GM}(p_{ik^*}^{(h)}) \leq \mathrm{AM}(p_{ik^*}^{(h)}) = 1/t$. This inequality holds because there are totally t non-absorbing states in the Markov chain, and the transition probabilities to all states sum up to 1. As a result, we can obtain:

$$\left(\mathbf{Q}^h\right)_{ik} \leq C * \left(\mathrm{GM}(p_{ik}^{(h)})\right)^h \leq C/t^h. \tag{25}$$

Recalling Eq. (23), if we only keep the terms with power less than or equal to m, the estimation error can be presented as follows:

$$|b_{i,j} - \tilde{b}_{i,j}| = \sum_{k=1}^{t} p_{kj} \cdot \sum_{h=m+1}^{\infty} \left(\mathbf{Q}^h\right)_{i,k} \leq \sum_{k=1}^{t} p_{kj} \cdot \sum_{h=m+1}^{\infty} C/t^h. \tag{26}$$

Since $p_{kj} < 1$ is a transition probability, we know $\sum_{k=1}^{t} p_{kj} < t$. Therefore,

$$\sum_{k=1}^{t} p_{kj} \cdot \sum_{h=m+1}^{\infty} C/t^h \leq tC \sum_{h=m+1}^{\infty} 1/t^h = tC \cdot \frac{1/t^{m+1}}{1-1/t} = \frac{C}{t^{m-1}(t-1)}. \tag{27}$$

In this way, we can obtain the final inequality:

$$|b_{i,j} - \tilde{b}_{i,j}| \leq \sum_{k=1}^{t} p_{kj} \cdot \sum_{h=m+1}^{\infty} C/t^h \leq \frac{C}{t^{m-1}(t-1)} \tag{28}$$

□

E Details on Experiments

In this section, we introduce the datasets, parameter settings for our framework and baselines, and training and evaluation details.

E.1 Datasets

In this section, we provide further details on the five datasets used in the experiemnts. (1) Tissue-PPI [23] consists of 24 protein-protein interaction (PPI) networks from different tissues. The node features are obtained based on gene signatures, and node labels are gene ontology functions [45]. Each label corresponds to a binary classification task, where the total number of such labels is 10. (2) Fold-PPI [1] is a dataset provided by G-Meta, constructed from 144 tissue networks. The labels are assigned based on the corresponding protein structures defined in the SCOP database. Specifically, fold groups with more than nine unique proteins are selected, resulting in 29 unique labels. Node features are conjoint triad protein descriptors [25]. Different from Tissue-PPI where all nodes are assigned labels, the labeled nodes in Fold-PPI are relatively scarce, which better fits into the few-shot scenario. (3) DBLP [14] is a citation network, where each node represents a paper, and links between nodes are created based on the citation relationship. The node features are generated

from the paper abstracts, and the class labels denote the paper venues. (4) Cora-full [26] is a citation network with node labels assigned based on the paper topic. This dataset extends the prevalent small dataset via extracting original data from the entire network. (5) ogbn-arxiv [27] is a directed citation network which consists of all CS arXiv papers indexed by MAG [46], where nodes represent arXiv papers, and edges indicate citation relationships. The node labels are assigned based on 40 subject areas of arXiv CS papers.

E.2 Parameter Settings

In this section, we introduce the detailed parameter settings for our experiments. For the ogbn-arxiv dataset, the number of update steps is 40 with a meta-learning rate of 0.005 and a base learning rate of 0.1. For other single-graph datasets, the number of update steps is 20 with a meta-learning rate of 0.005 and a base learning rate of 0.1. For the Tissue-PPI dataset, the number of update steps is 20 with a meta-learning rate of 0.005 and a base learning rate of 0.01. For the Fold-PPI dataset, the number of update steps is 20 with a meta-learning rate of 0.005 and a base learning rate of 0.1. The hidden dimension sizes of GNNs are set as 16. The dropout rate is set as 0.5. The weight decay rate is set as 10^{-4}. For the approximation of absorbing probabilities, we set the value of m as 2. For the common sampling, we set the value of C as 10. For the local sampling, we set the value of h as 2 (i.e., 2-hop neighbors). The activation functions are all set as the ReLU function.

E.3 Baseline Settings

In this section, we introduce the detailed settings for baselines used in the experiments. (1) KNN [28]: We follow the settings in G-Meta to first train a GNN on all training data as an embedding function. During test, we assign a label for each query node based on the voted K-closest node in the support set. (2) ProtoNet [17]: This baseline classifies query nodes based on their distances to the learned prototypes (i.e., the average embedding of support nodes in a specific class). We set the learning rate as 0.005 with a weight decay of 0.0005. (3) MAML [21]: This baseline performs several update steps within each meta-task and meta-update the parameter based on query loss. The meta-learning rate is set as 0.001 and the number of update step is 10 with a learning rate of 0.01. (4) Meta-GNN [10]: This baseline combines MAML with Simple Graph Convolution (SGC) [29]. Following the original work, we set the learning rate and meta-learning rate as 0.5 and 0.003, respectively. (5) G-Meta [1]: This baseline leverages the local subgraphs to learn node embeddings while combining ProtoNet and MAML. Following the original work, we set the numbers of update steps for meta-training and meta-test as 10 and 20, respectively. The inner learning rate is 0.01 while the outer learning rates is 0.005. The hidden dimension size is set as 128. (6) GPN [9]: This baseline learns node importance and utilizes the ProtoNet to classify query nodes. We follow the setting in the source code and set the learning rate as 0.005 with a weight decay of 0.0005. The dimension sizes of two GNNs used in GPN are set as 32 and 16, respectively. (7) RALE [11]: This baseline learns node embeddings based on the relative and absolute locations of nodes. We set the learning rates for training and fine-tuning as 0.001 and 0.01, respectively. The hidden size of used GNNs is 32.

E.4 Details on Training and Evaluation

We train our model on a single 16GB Nvidia V100 GPU. The GNNs used in our experiments are implemented with Pytorch [47], which is under a BSD-style license. The required packages are listed as below.

- Python == 3.7.10
- torch == 1.8.1
- torch-cluster == 1.5.9
- torch-scatter == 2.0.6
- torch-sparse == 0.6.9
- torch-geometric == 1.4.1
- numpy == 1.18.5
- scipy == 1.5.3

For the training and evaluation setting, we adopt different choices for single-graph and multiple-graph settings. Specifically, for the single-graph datasets, we adopt two settings: 5-way 3-shot (i.e., $N = 5$ and $K = 3$) and 10-way 3-shot (i.e., $N = 10$ and $K = 3$). For the multiple-graph datasets, we adopt 3-way 3-shot (i.e., $N = 3$ and $K = 3$) for Fold-PPI and 2-way 5-shot (i.e., $N = 2$ and $K = 5$) for Tissue-PPI. This is because Tissue-PPI consists of 10 binary classification tasks. The number of training epochs is set as 500. Furthermore, to keep consistency, the meta-testing tasks are identical for all baselines. For the class split setting in the single-graph datasets, we set training/validation/test classes as 15/5/20 for ogbn-arxiv, 25/20/25 for Cora-full, and 80/27/30 for DBLP. For the class split setting in the multiple-graph datasets, the class split setting on the disjoint label task is 21/4/4 for Fold-PPi and 8/1/1 for Tissue-PPI (each label in Tissue-PPI corresponds to a binary classification task). The graph split setting is 80%/10%/10%, which follows the same setting as G-Meta.

F Supplementary Discussion

F.1 Limitations

This paper aims at learning task-specific structures for each meta-task to promote few-shot node classification performance. Nevertheless, certain disadvantages exist in our specific design. First, the learned task-specific structures is tailored for one meta-task (including the support set and the query set) and cannot be easily generalized to other meta-tasks. In consequence, when a scenario setting requires a significantly larger query set than the support set, the generalization to all these query nodes can be difficult due to the large query set size. Second, when the graph size is relatively small, the original graph can be sufficient for conducting few-shot tasks. As a result, the learned task-specific structures can potentially lead to loss of useful information when the graph size is excessively small.

F.2 Negative Impacts

This paper studies the problem of graph few-shot learning, which exists widely in real-world applications. For example, certain protein structures maintain scarce labeled proteins, which increase the difficulty of classification on such protein structures [1]. Moreover, the technique is novel and suitable for various scenarios. Therefore, we currently do not foresee any negative impacts in our proposed framework.

F.3 Potential Improvements

Preserving the Original Graph. In our design, we select a specific number of nodes to construct the task-specific structure for each meta-task. Nevertheless, during this process, the useful information in other nodes can be lost. Although through our design, we can maintain the majority of useful information in the selected nodes, the nodes that are not selected can still be helpful. Therefore, a potential strategy for improvements can be keeping the original graph while learning flexible edge weights among all the nodes. In this way, the information inside all nodes will be preserved for better performance. This strategy is especially helpful for the datasets with small graphs, where all the nodes can be potentially informative. Nevertheless, when the graph size becomes larger, such a strategy can lead to scalablility problems.

Introducing Multiple GNNs. Although the learned task-specific structure is tailored for the meta-task, the GNNs are only applied to this structure. As a result, the information propagation process is restricted in the task-specific structure. A possible improvement strategy is to leverage another GNN that propagates messages on the original graph. Meanwhile, the learned features can be incorporated into the task-specific structure. In this way, the incorporated features are task-agnostic and can help the learning in each meta-task in a comprehensive manner. Nevertheless, such a strategy is not suitable for the multiple-graph setting, since learning GNNs across different graphs can lead to suboptimal performance.