A. Minimal and Sufficient Meta-Knowledge

A.1. Motivation

we provide a strategy for extracting the minimal and sufficient meta-knowledge in model training phase based on the two propositions proposed.

Proposition 1. Given a graph G with base classes Y_b, the minimal and sufficient meta-knowledge Z for FSNC on G could be obtained by solving the following problem:

$$\max_{Z} [I(Z; G) - I(Z; N) - \beta I(Z; Y_b)] \tag{1}$$

where N denotes the irrelevant information or noises in G and $\beta(\beta > 0)$ is a hyper-parameter. The first term $I(Z; G)$ is the embedding term that encourages to preserve the essential structural information of G. The second term $-I(Z; N)$ is the denoising term that encourages to eliminate the noises in G. The third term $-I(Z; Y_b)$ is the compression term that encourages to remove the label information that irrelevant to the novel classes from Y_b.

Proposition 2. Given a noisy graph G with novel classes Y_n, G^* is the essential structure of G if and only if $I(G^*; Y_n) = I(G; Y_n)$ and $H^1(G) > H^1(G^*)$. When G^* is obtained, we have the following

$$\max_{Z} [I(Z; G) - I(Z; N)] \Leftrightarrow \max_{Z} I(Z; G^*) \tag{2}$$

We provide an illustration of our motivation in Figure 1. When Z contains the information from the graph structure G and base class Y_b that is irrelevant to the novel class Y_n , it could result in overfitting. Figure 1 (a) illustrates traditional meta-learning method without constraints on information from G or Y_b. In contrast, Figure 1 (b) illustrates the graph information bottleneck (GIB) method (Wu et al., 2020) that constrains information from G. Figure 1 (c) describes our training objective, which imposes constraints on both G and Y_b. For further comparison between GIB and our method, please refer to Experiment C.1.

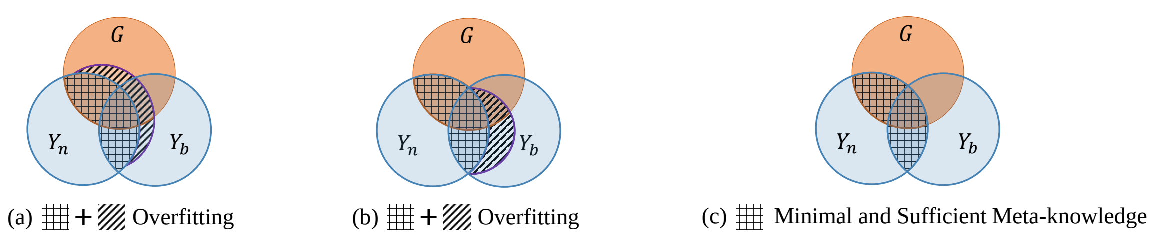

Figure 1. An illustration of the minimal and sufficient meta-knowledge Z.

A.2. Why Encoding Tree?

Graph structure learning (GSL) is an effective approach to eliminate noise in graph and improves node classification accuracy (Zhu et al., 2021). Here, we present a lemma to illustrate that GSL for node classification tasks to reduce the uncertainty of the original graph structure.

Lemma 1. Graph structure learning reduces the uncertainty of the original graph structure.

Proof. GSL models for node classification aim to enhance the intra-class connections but reduce the inter-class connections to improve the accuracy of classification. Let G be an undirected connected graph with n nodes, m edges and k equally-sized node categories. Suppose there is a GSL model that can generate the optimal graph structure $G^\star$ by eliminating all inter-class connections, i.e., $G^\star = \cup_{j=1}^{k} G_j^\star$, where $G_j^\star$ is a connected component of $G^\star$ with $\frac{n}{k}$ nodes, in which the nodes belong to the same category. The one-dimensional structural entropy $H^1(G^\star)$ measures the uncertainty of the one-dimensional structures of $G^\star$ (Li & Pan, 2016):

$$H^1(G^\star) = \frac{1}{vol(G^\star)} \sum_{j=1}^{k} vol(G_j^\star) H^1(G_j^\star) \tag{3}$$

According to the structural information theory, the lower bound and upper bound of $H^1(G)$ are determined by n and m respectively, following the inequalities:

$$\frac{1}{2}(\log_2 m - 1) \leqslant H^1(G) \leqslant \varepsilon \log_2 n \tag{4}$$

where $\varepsilon \in [0, 1]$ is related to the sparsity of G. Specifically, if G is sparse, then $0 \leqslant \varepsilon \leqslant 1/2$ (Li & Pan, 2016).

Since the real-world graphs are usually sparse and GSL models contain a regularization term to keep the sparsity in the optimal graph (Zhu et al., 2021), we have $H^1(G_j^\star) \leqslant \frac{1}{2}\log_2\frac{n}{k}$. According to Eq. 3, we obtain

$$H^1(G^\star) \leqslant \frac{1}{vol(G^\star)}\sum_{j=1}^{k} vol(G_j^\star)\frac{1}{2}\log_2\frac{n}{k} = \frac{1}{2}\log_2\frac{n}{k} \tag{5}$$

Since G is connected, we have $m \geqslant n-1$ and

$$H^1(G) \geqslant \frac{1}{2}[\log_2(n-1) - 1] \tag{6}$$

Combining Eq. 5 and Eq. 6, the difference of the one-dimensional structural entropy between G and $G^\star$ satisfies

$$\begin{aligned} H^1(G) - H^1(G^\star) &\geqslant \frac{1}{2}[\log_2(n-1) - 1] - \frac{1}{2}\log_2\frac{n}{k} \\ &= \frac{1}{2}\log_2\left[\frac{k(n-1)}{2n}\right] \end{aligned} \tag{7}$$

For the node classification tasks satisfying $k \geqslant \frac{2n}{n-1} \approx 2$, we can obtain

$$H^1(G) - H^1(G^\star) \geqslant 0 \tag{8}$$

Thus, we conclude that the uncertainty of $G^\star$ is less than that of G after GSL.

$\square$

According to Lemma 1, we can apply the encoding tree generated by minimizing the uncertainty of graph structure to implement Proposition 2.

B. Encoding Tree

B.1. Encoding Tree Construction

An efficient encoding tree construction algorithm is described here. Specifically, given a graph $G(V, E)$, let $\mathcal{P} = \{P_1, ..., P_c\}$ be a partition of V, where each $P_i \subset V$ is called a community. There are three basic operators as follows:

Definition 1. (Merging operator) Given any two communities P_i and P_j ($1 \leq i < j \leq c$), merging operator $op_m(P_i, P_j)$ merges P_i and P_j into a new community P_x, i.e., $P_x = P_i \cup P_j$, and then removes P_i and P_j from $\mathcal{P}$. After merging, $\mathcal{P} = \{P_1, ..., P_{i-1}, P_{i+1}, ..., P_{j-1}, P_{j+1}, P_c, P_x\}$.

The difference of K-dimensional structural entropy $\Delta SE_{i,j}^{\mathcal{P}}(G)$ before and after merging could be calculated by

$$\begin{aligned} \Delta SE_{i,j}^{\mathcal{P}}(G) = \frac{1}{vol(G)}[&(\mathcal{V}_i - g_i)\log_2\mathcal{V}_i + (\mathcal{V}_j - g_j)\log_2\mathcal{V}_j \\ &- (\mathcal{V}_x - g_x)\log_2\mathcal{V}_x + (g_i + g_j - g_x)\log_2 vol(G)] \end{aligned} \tag{9}$$

Definition 2. (Compressing operator) Given a graph G and a corresponding partition $\mathcal{P}$, compressing operator $op_c(\mathcal{P})$ compresses G into a smaller graph by transferring each community $P_i \in \mathcal{P}$ to a node v_i', and assigning the weight of edge between v_i' and v_j' to the sum of the weights of the edges from P_i to P_j.

Definition 3. (Updating operator) Given an encoding tree $\mathcal{T}$ and a graph G with partition $\mathcal{P}$, updating operator $op_u(\mathcal{T}, \mathcal{P})$ is to update the encoding tree by taking all communities in $\mathcal{P}$ as the leaf nodes of $\mathcal{T}$, i.e., inserting $\mathcal{P}$ into $\mathcal{T}$ and increasing the height of $\mathcal{T}$.

Initially, we adopt each node in the graph as a single community, and then iteratively execute the merging and compressing operators until the updating operator could construct a K-dimensional encoding tree. Actually, in the merging operation, we merge the communities with the maximal $\Delta SE^{\mathcal{P}}_{i,j}(G)$ greedily until there are no communities satisfying $\Delta SE^{\mathcal{P}}_{i,j}(G) > 0$, which can achieve the minimal structural entropy. The complete procedure is shown in Algorithm 1.

Algorithm 1 Encoding Tree Construction

Input: a graph G, an integer $K > 1$
Output: an encoding tree $\mathcal{T}$

$G_1 \leftarrow G$, $\mathcal{T} \leftarrow$ an encoding tree with height 1
for $h = 1$ to K **do**
  $\mathcal{P}_h \leftarrow$ initialize each node in G_h as a community
  while $True$ **do**
    $P'_i, P'_j \leftarrow \arg\max \Delta SE^{\mathcal{P}_h}_{i,j}(G_h)$ by Eq. 9
    if $\Delta SE^{\mathcal{P}_h}_{i,j}(G_h) > 0$ **then**
      $\mathcal{P}_h \leftarrow op_m(P'_i, P'_j)$ // Definition 1
      continue
    else
      $G_h \leftarrow op_c(\mathcal{P}_h)$ // Definition 2
      break
    end if
  end while
end for
for $h = K - 1$ down to 0 **do**
  $\mathcal{T} \leftarrow op_u(\mathcal{T}, \mathcal{P}_h)$ // Definition 3
end for
return $\mathcal{T}$

We then present an example for building a three-dimensional encoding tree.

Example B.1. The construction of the encoding tree consists of two stages.

(1) The first stage iteratively constructs the two-dimensional encoding trees by using the merging and compressing operators in a bottom-up manner, shown in Figure 2. Note that a two-dimensional encoding tree is naturally constructed after the merging operation.

(2) The second stage utilizes the updating operator in a top-down manner to combine the two-dimensional encoding trees and generate a higher-dimensional encoding tree, shown in Figure 3.

B.2. Time Complexity

Given a graph G with n nodes, the execution of merging and compressing operators takes $O(n\log^2 n + n + q)$ time, where q is the number of communities generated by the merging operator. Note that each compressing operator will gradually reduce the number of nodes in the compressed graph, which is significantly smaller than n. According to structural information theory (Li & Pan, 2016), the time complexity of the merging operator is $O(n\log^2 n)$. We need to traverse each node to compute the edge weights of the compressed graph, has a time complexity of $O(n)$. Additionally, the time complexity of the updating operator depends on the number of communities in the updating layer, i.e., $O(q)$. Therefore, Algorithm 1 is efficient for constructing encoding trees.

C. Experimental Results

We have supplemented experiments on few-shot node classification and efficiency.

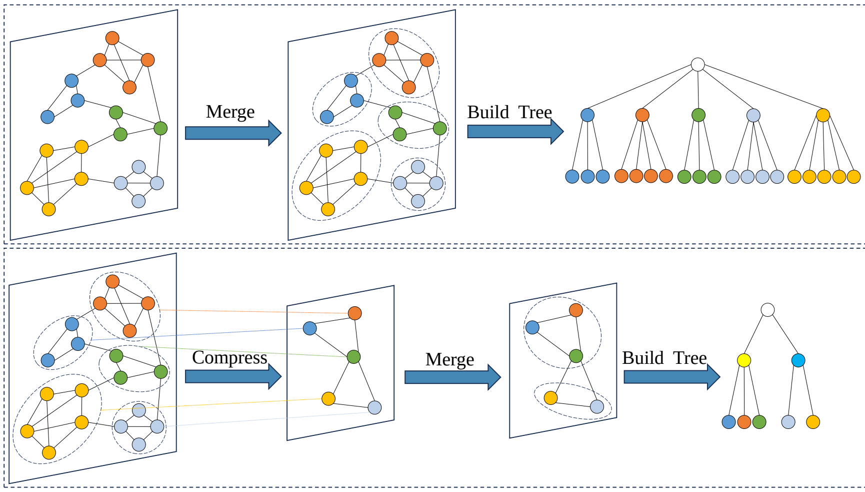

Figure 2. Bottom-up construction of the two-dimensional encoding tree

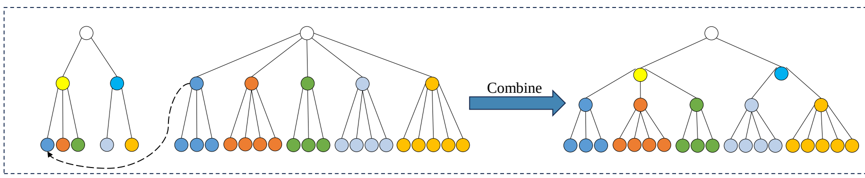

Figure 3. Top-down construction of the higher-dimensional encoding tree

C.1. Comparison on Node Classification

Exp-1: Comparison with GIB loss. We provide a two-stage training objective for GIB-based few-shot node classification. Initially, we train a mutual information estimator with the InfoNCE loss (Oord et al., 2018) to approximate the true mutual information value. Then, we employ the optimization objective of GIB to train the encoder and classifier. The results on all datasets under different few-shot settings are reported in Table 1. These results tell us that Our method outperforms traditional GIB method.

Table 1. Effectiveness comparison with GIB-based method on node classification under different few-shot settings. Accuracy ($\uparrow$) and confidence interval ($\downarrow$) are in $\%$. The best results are bold.

Dataset	Method	5-way 1-shot	5-way 5-shot	10-way 1-shot	10-way 5-shot
CoraFull	GIB	61.46 $\pm$ 2.32	78.05 $\pm$ 1.66	47.93 $\pm$ 1.50	72.48 $\pm$ 1.18
	ours	**77.95 $\pm$ 2.20**	**89.24 $\pm$ 1.30**	**66.11 $\pm$ 1.59**	**82.03 $\pm$ 1.08**
Amazon-Clothing	GIB	69.28 $\pm$ 2.50	85.58 $\pm$ 1.74	60.75 $\pm$ 1.79	76.61 $\pm$ 1.34
	ours	**83.25 $\pm$ 2.37**	**92.12 $\pm$ 1.52**	**77.04 $\pm$ 1.73**	**88.12 $\pm$ 1.15**
DBLP	GIB	66.68 $\pm$ 2.55	81.12 $\pm$ 1.90	55.45 $\pm$ 1.79	69.61 $\pm$ 1.49
	ours	**74.07 $\pm$ 2.49**	**86.09 $\pm$ 1.81**	**65.08 $\pm$ 1.94**	**76.49 $\pm$ 1.46**
ogbn-arxiv	GIB	47.19 $\pm$ 1.98	64.19 $\pm$ 1.68	33.62 $\pm$ 1.20	50.29 $\pm$ 1.03
	ours	**55.07 $\pm$ 3.84**	**72.34 $\pm$ 1.62**	**43.03 $\pm$ 1.30**	**61.01 $\pm$ 0.94**

Exp-2: 2-way k-shot node classification. we evaluate the effectiveness of our method on FSNC tasks by comparing it with FSL-based methods. The results on all datasets under 2-way k-shot settings are reported in Table 2. Our method is not optimal, but remains competitive. This may be due to insufficient compression of label information from the base classes in the 2-way k-shot settings.

Table 2. Effectiveness comparison with FSL-based methods on node classification under 2-way k-shot settings. Accuracy (↑) and confidence interval (↓) are in %. The best and second best results are bold and underline, respectively.

Dataset	CoraFull		Amazon-Clothing		DBLP		ogbn-arxiv	
Way	2-way		2-way		2-way		2-way	
Shot	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
MAML	52.13 ± 2.16	56.70 ± 2.32	63.40 ± 2.89	76.38 ± 2.54	54.92 ± 2.46	58.96 ± 2.45	59.77 ± 2.53	67.05 ± 2.50
ProtoNet	62.93 ± 2.66	75.75 ± 2.27	66.27 ± 1.22	80.40 ± 2.56	61.31 ± 2.19	76.00 ± 1.61	63.68 ± 1.75	74.83 ± 3.26
G-meta	67.24 ± 2.21	76.35 ± 2.90	76.60 ± 2.19	82.00 ± 2.38	65.25 ± 0.25	70.48 ±1.77	61.15 ± 1.77	65.55 ± 1.70
TENT	81.28 ± 2.39	90.75 ± 1.10	89.95 ± 1.66	96.20 ± 0.56	86.14 ± 2.03	92.58 ± 1.02	70.81 ± 2.37	79.89 ± 2.58
TEG	83.60 ± 4.68	92.60 ± 2.42	88.41 ± 3.84	94.24 ± 2.03	88.34 ± 4.36	93.41 ± 2.45	73.02 ± 6.07	85.14 ± 3.74
GPPT	84.77 ± 2.99	91.98 ± 1.79	87.96 ± 3.03	95.67 ± 1.50	**89.90 ± 2.76**	**94.53 ± 1.90**	**78.27 ± 3.40**	**87.98 ± 2.00**
COSMIC	79.00 ± 2.02	92.08 ± 1.13	78.40 ± 2.55	96.34 ± 1.00	83.32 ± 2.16	92.15 ± 1.56	73.50 ± 1.74	81.98 ± 0.83
GLITTER	80.58 ± 1.09	89.98 ± 0.54	81.05 ± 1.19	92.65 ± 0.99	84.78 ± 0.97	92.59 ± 1.02	71.50 ± 7.55	80.00 ± 0.55
ours	**86.99 ± 2.73**	**94.36 ± 1.35**	**90.07 ± 1.57**	**96.82 ± 0.78**	87.13 ± 1.26	94.18 ± 1.10	75.58 ± 2.53	87.88 ± 1.96

Exp-3: Comparison with COSMIC. We compared the parameters provided in the COSMIC paper(Wang et al., 2023) with the parameters we fine-tuned under the 10-way 1-shot setting for all methods, the experimental results are shown in Table 3.

Table 3. Effectiveness comparison with COSMIC on node classification under different under different hyper parameters. Accuracy (↑) and confidence interval (↓) are in %. The best results are bold.

Dataset	Method	5-way 1-shot	5-way 5-shot	10-way 1-shot	10-way 5-shot
CoraFull	COSMIC	68.30 ± 2.24	**87.14 ± 1.15**	55.09 ± 1.26	70.21 ± 0.89
	COSMIC-fine turning	**70.09 ± 1.53**	85.13 ± 1.55	**57.98 ± 1.57**	**72.32 ± 1.09**
Amazon-Clothing	COSMIC	**80.21 ± 1.63**	86.43 ± 1.14	**68.20 ± 1.25**	78.16 ± 1.51
	COSMIC-fine turning	73.16 ± 2.35	**89.37 ± 1.53**	67.04 ± 1.75	**80.80 ± 1.52**
DBLP	COSMIC	**70.72 ± 2.05**	77.19 ± 1.45	56.85 ± 1.40	**74.70 ± 0.98**
	COSMIC-fine turning	66.82 ± 2.37	**83.98 ± 1.85**	**57.28 ± 1.72**	70.77 ± 1.45
ogbn-arxiv	COSMIC	47.84 ± 1.65	**64.22 ± 2.16**	**35.39 ± 1.45**	51.39 ± 0.89
	COSMIC-fine turning	**50.30 ± 1.93**	62.00 ± 1.68	32.50 ± 1.18	**56.98 ± 0.99**

C.2. Efficiency analysis

Exp-1: Encoding tree construction time. we record the running time for constructing encoding trees. The results on all datasets are reported in Table 4. It takes only a few seconds on CoraFull and Amazon-Clothing datasets. However, our implementation utilizes only one thread for computation, resulting in inefficient construction on large graphs.

Table 4. The overall Encoding tree construction time (sec.) results of different depths.

Tree height	CoraFull	Amazon-Clothing	DBLP	ogbn-arxiv
2	13.52	12.23	371.08	10297.74
3	15.45	13.80	380.17	10622.56
4	15.48	13.96	381.28	10647.50
5	16.45	14.07	381.47	10703.32

Exp-2: Training time. We conduct experiments to evaluate the computational cost of our method compared to other few-shot learning methods. We exclude the data preprocessing steps (e.g., subgraph construction time per node in COSMIC, encoding tree construction time in our method) and reported the average of 10 random runs. The results on all datasets under 10-way 1-shot settings are reported in Table 5. These results tell us that our method requires fewer episodes, because our contrastive loss is more effective to utilize the hierarchical information.

Table 5. The efficiency results of our method and FSL-based methods under 10-way 1-shot setting.

Dataset	CoraFull		Amazon-Clothing		DBLP		ogbn-arxiv	
Result	Time(s)	#Episodes	Time(s)	#Episodes	Time(s)	#Episodes	Time(s)	#Episodes
MAML	6.69	18.70	15.78	54.30	6.39	11.00	6.14	7.70
ProtoNet	38.53	183.30	40.77	180.00	38.23	150.00	59.87	273.30
G-meta	3072.60	180.50	1372.35	320.00	2291.20	170.50	2432.70	353.40
TENT	40.50	150.00	45.83	183.40	141.83	616.70	455.67	233.30
TEG	53.97	6.00	56.43	6.67	211.69	24.05	445.59	46.03
GPPT	59.04	6.56	41.41	4.56	139.21	11.25	403.91	8.06
COSMIC	174.07	411.00	112.01	197.00	118.32	244.00	88.30	217.00
CLITTER	520.88	667.80	487.37	3749.00	5197.17	6250.00	1531.82	2250.00
ours	150.67	28.70	52.81	11.40	184.89	39.70	703.47	100.00

References

Li, A. and Pan, Y. Structural information and dynamical complexity of networks. *IEEE Transactions on Information Theory*, 62:3290–3339, 2016.

Oord, A. v. d., Li, Y., and Vinyals, O. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Wang, S., Tan, Z., Liu, H., and Li, J. Contrastive meta-learning for few-shot node classification. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2386–2397, 2023.

Wu, T., Ren, H., Li, P., and Leskovec, J. Graph information bottleneck. In *Proceedings of Advances in Neural Information Processing Systems*, volume 33, pp. 20437–20448, 2020.

Zhu, Y., Xu, W., Zhang, J., Liu, Q., Wu, S., and Wang, L. Deep graph structure learning for robust representations: A survey. *arXiv preprint arXiv:2103.03036*, 14, 2021.