

Learning to Model Graph Structural Information on MLPs via Graph Structure Self-Contrasting

Lirong Wu, Haitao Lin, Guojiang Zhao, Cheng Tan, and Stan Z. Li[†], *Fellow, IEEE*

Abstract—Recent years have witnessed great success in handling graph-related tasks with Graph Neural Networks (GNNs). However, most existing GNNs are based on message passing to perform feature aggregation and transformation, where the structural information is explicitly involved in the forward propagation by coupling with node features through graph convolution at each layer. As a result, subtle feature noise or structure perturbation may cause severe error propagation, resulting in extremely poor robustness. In this paper, we rethink the roles played by graph structural information in graph data training and identify that message passing is not the only path to modeling structural information. Inspired by this, we propose a simple but effective *Graph Structure Self-Contrasting* (GSSC) framework that learns graph structural information without message passing. The proposed framework is based purely on Multi-Layer Perceptrons (MLPs), where the structural information is only implicitly incorporated as prior knowledge to guide the computation of supervision signals, substituting the explicit message propagation as in GNNs. Specifically, it first applies structural sparsification to remove potentially uninformative or noisy edges in the neighborhood, and then performs structural self-contrasting in the sparsified neighborhood to learn robust node representations. Finally, structural sparsification and self-contrasting are formulated as a bi-level optimization problem and solved in a unified framework. Extensive experiments have qualitatively and quantitatively demonstrated that the GSSC framework can produce truly encouraging performance with better generalization and robustness than other leading competitors. Codes are publicly available at: <https://github.com/LirongWu/GSSC>.

Index Terms—Graph Neural Networks, Contrastive Learning, Graph Sparsification, Graph Structure Learning.

I. INTRODUCTION

Recently, the emerging Graph Neural Networks (GNNs) have demonstrated their powerful capability in handling graph-related tasks [1–6]. Despite their great success, most existing GNNs are based on message passing, which consists of two key computations: (1) AGGREGATE: aggregating messages from its neighborhood, and (2) UPDATE: updating node representation from its representation in the previous layer and the aggregated messages. Due to the *explicit coupling of node features and graph structural information* in the AGGREGATE operation, subtle feature noise or structure perturbation may cause severe error propagation during message passing, resulting in extremely poor robustness [7–10]. There has been some pioneering work [11, 12] delving into the necessity of message passing for modeling graph structural information. These methods adopt a pure MLP architecture

that is free from feature-structure coupling, where structural information is only implicitly used to guide the computation of downstream supervision. For example, Graph-MLP [11] designs a neighborhood contrastive loss to bridge the gap between GNNs and MLPs by implicitly utilizing the adjacency information. Besides, LinkDist [12] directly distills self-knowledge from connected node pairs into MLPs without the need for aggregating messages. Despite their great progress, these MLP-based models still cannot match the state-of-the-art GNNs in terms of classification performance due to the underutilization of graph structural information. Therefore, “*how to make better use of graph structural information without message passing*” is still a challenging problem.

The quality of graph structural information plays a very key crucial in various graph learning algorithms, making graph sparsification techniques, such as DropEdge [13], STR-Sparse [8] and GAUG [14], etc., emerge as a common means to improve performance. The goal of graph sparsification is to filter out noisy structural information, i.e., finding an informative subgraph from the input graph by removing noisy edges. However, most existing graph sparsification methods are tailored for general GNNs, and they jointly optimize graph sparsification and GNN training by back-propagation of downstream supervision in an end-to-end manner. However, such an optimization may work for GNNs but is hard to be extended directly to MLP-based models, where structural information has been used to guide the computation of downstream supervision, and it is not feasible to use downstream supervision in turn to optimize the structural sparsification.

The differences between the two cases of using graph sparsification techniques for GNNs and MLP-based models are illustrated in Fig. 1, where the key point is *whether graph structural information is used explicitly or implicitly*. For GNN models, the structural information G is explicitly involved in the forward propagation, coupled with node features through graph convolution at each layer, and thus we can directly obtain the gradients, i.e., the derivative of supervision loss w.r.t the sparsification parameters. As a result, the parameters of GNN and graph sparsification can be jointly optimized by downstream supervision. In contrast, MLP-based models only implicitly utilize structural information G to guide the computation of downstream supervision signals (denoted as a dashed line), and thus the derivative of supervision loss w.r.t the sparsification parameters is not available, which prevents the sparsification network from being directly optimized through downstream supervision (denoted as a red cross). As an alternative, this paper proposes a homophily-oriented objective to guide the optimization of the sparsification network.

Lirong Wu, Haitao Lin, Guojiang Zhao, Cheng Tan, and Stan Z. Li are with the AI Lab, Research Center for Industries of the Future, Westlake University, Hangzhou 310024, China. E-mail: {wulirong, linhaitao, zhaoguojiang, tancheng, stan.zq.li}@westlake.edu.cn. [†] Corresponding Author.

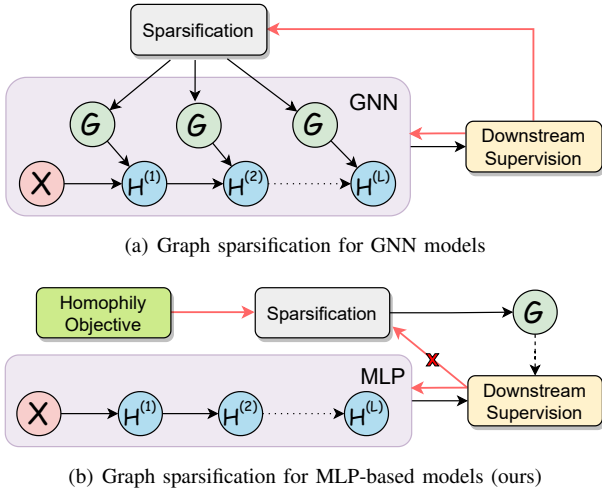


Fig. 1: A comparison between two cases of using graph sparsification for GNNs and MLP-based models. \mathbf{X} denotes the input features, \mathbf{H} denotes the hidden features, and \mathbf{G} is the sparsified subgraph. The forward and backward propagation are marked as black and red lines, respectively.

In this paper, we propose a simple yet effective *Graph Structural Self-Contrasting* (GSSC) framework, which consists of two main networks: (i) *Structural Sparsification (STR-Sparse)* and (ii) *Structural Self-Contrasting (STR-Contrast)*. In the proposed GSSC framework, we first apply the STR-Sparse network on the input graph to remove potentially uninformative or noisy edges in the neighborhood and then perform STR-Contrast in the sparsified neighborhood by imposing structural smoothness constraints between connected nodes. Specifically, the STR-Contrast network is based on MLPs, where structural information is only implicitly used to guide the computation of supervision signals but does not involve the forward propagation. Finally, we formulate structural sparsification and self-contrasting as a bi-level optimization problem and optimize the two networks using a tailor-made homophily-oriented objective and downstream supervision, respectively. Extensive experiments have shown that GSSC can produce truly encouraging performance with better generalization and robustness than other state-of-the-art competitors. To our best knowledge, we are the first work to explore the applicability of graph sparsification to (non-GNN) MLP-based models.

II. RELATED WORK

A. Graph Representation Learning

Recent years have witnessed the great success of GNNs in graph learning [15–19]. There are two categories of GNNs: spectral GNNs and spatial GNNs. The spectral-based GNNs define convolution kernels in the spectral domain based on the graph signal processing theory. For example, ChebyNet [20] uses the polynomial of the Laplacian matrix as the convolution kernel to perform message passing, and GCN is its first-order approximation. The spatial-based GNNs focus on the design of aggregation functions directly. For example, GraphSAGE [21] employs a generalized induction framework to efficiently generate node embeddings for previously unseen

data by aggregating known node features. GAT [22], on the other hand, adopts the attention mechanism to calculate the coefficients of neighbors for better information aggregation. We refer interested readers to the recent survey [2] for more GNN variants, such as SGC [23], APPNP [24] and DAGNN [1]. However, the above GNNs all share the de facto design that structural information is explicitly utilized for message passing to aggregate node features from the neighborhood.

Recently, there are some recent attempts to train graph data by combining contrastive learning and knowledge distillation [25, 26] with MLPs. For example, Graph-MLP [11] designs a neighborhood contrastive loss to bridge the gap between GNNs and MLPs by implicitly utilizing the adjacency information. Instead, LinkDist [12] directly distills self-knowledge from connected node pairs into MLPs without the need to aggregate messages. Despite their great progress, they still cannot match the state-of-the-art GNN models in terms of classification performance, more importantly, they ignore the potential noise in the graph structure. Another MLP-based model is Graph MLP-Mixer [27], which generalizes ViT/MLP-Mixer to graph data and have achieved promising results.

Graph transformers (GTs) is another research area closely related to graph representation learning. For example, [28] is the first work that generalizes Transformer to graphs, which uses Laplacian position encoding to preserve local structural information. GraphiT [29] utilizes relative position encoding to enhance attention mechanism. Recently, GRIT [30] proposes a novel structural encoding called relative random walk probabilities (RRWP) to enhance local structure expressive power. Besides, Graph Transformers (SGFormer) [31] simplifies and empowers Transformers, which requires none of positional encodings, feature/graph pre-processing, or augmented loss. For more GT architectures, please refer to recent survey [32].

B. Graph Sparsification

The robust learning on graphs includes adversarial training [9, 33, 34], label denoising [35], structure learning [36–38], etc., among which the closest one to ours is graph sparsification, which aims to find a small subgraph from the input graph that best preserve some desired properties. Existing graph sparsification techniques can be divided into two categories: unsupervised and supervised. The unsupervised sparsification techniques, such as Spectral Sparsifier (SS) [39] and Rank Degree (RD) [40], mainly deal with simple graphs by some pre-defined graph metrics, e.g., node degree distribution, clustering coefficient, etc. Besides, DropEdge [13] randomly removes a fraction of edges according to the pre-defined probability before each training epoch to get sparsified graphs. In contrast, supervised sparsification directly parameterizes the sparsification process and optimizes it end-to-end along with GNN parameters under downstream supervision. For example, Learning Discrete Structure (LDS) [41] works under a transductive setting and learns Bernoulli variables associated with individual edges. Besides, GAUG [14] first optimizes the graph structure learning and GNN parameters in an end-to-end manner and then directly removes some low-importance edges during training. Moreover, NeuralSparse [8] proposes a general

framework that simultaneously learns to remove task-irrelevant edges and node representations by downstream supervision. Furthermore, L2A [42] unifies graph sparsification (augmentation) and GNN training in a variational inference framework, which is applicable to both homophily and heterophily graphs. However, these supervised sparsification techniques may be hard to be directly extended to existing MLP-based models.

C. Graph Structure Learning

Recent advances in Graph Structure Learning (GSL) provide new insights into reducing the dependency on the given graph structure. The primary goal of graph structure learning is to infer an underlying graph structure from node features and then apply a GNN classifier to the inferred graph [43]. For example, [43] proposes *Homophily-Enhanced Self-supervision for Graph Structure Learning* (HES-GSL), a method that provides additional self-supervision for learning an underlying graph structure. Similarly, [44] proposes a novel method called Homophily-enhanced structure Learning for graph clustering (HoLe), based on the observation that subtly enhancing the degree of homophily within the graph structure can significantly improve GNNs and clustering outcomes. To address the under-supervision problem, *Simultaneous Learning of Adjacency and GNN Parameters with Self-supervision* (SLAPS) [45] proposes a feature reconstruction-based pretext task to provide more self-supervision for graph structure learning. Besides, RDGSL [46] proposes dynamic graph structure learning, where dynamic graph filters are designed to address the noise dynamics issue. Moreover, CGI [47] proposes a contrastive graph structure learning via information bottleneck for recommendation, which adaptively learns whether to drop an edge or node to obtain optimized structures in an end-to-end manner. We refer interested readers to a recent survey [48] for more methods.

III. METHODOLOGY

Used Notations. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where \mathcal{V} is the set of N nodes with features $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N] \in \mathbb{R}^{N \times d}$ and \mathcal{E} denotes the edge set. Each node $v_i \in \mathcal{V}$ is associated with a d -dimensional features vector \mathbf{x}_i , and each edge $e_{i,j} \in \mathcal{E}$ denotes a connection between node v_i and v_j . The graph structure can also be denoted by an adjacency matrix $\mathbf{A} \in [0, 1]^{N \times N}$ with $\mathbf{A}_{i,j} = 1$ if $e_{i,j} \in \mathcal{E}$ and $\mathbf{A}_{i,j} = 0$ if $e_{i,j} \notin \mathcal{E}$. Node classification is a typical node-level task where only a subset of node \mathcal{V}_L with corresponding labels \mathcal{Y}_L are known, and we denote the labeled set as $\mathcal{D}_L = (\mathcal{V}_L, \mathcal{Y}_L)$ and unlabeled set as $\mathcal{D}_U = (\mathcal{V}_U, \mathcal{Y}_U)$, where $\mathcal{V}_U = \mathcal{V} \setminus \mathcal{V}_L$. The task of node classification aims to learn a mapping $p(Y | \mathbf{X}, \mathbf{A})$ on labeled data \mathcal{D}_L , so that it can be used to infer the label $Y \in \mathcal{Y}_U$.

A. Theoretical Justification

From the perspective of statistical learning, the key of node classification is to learn a mapping $p(Y | \mathbf{X}, \mathbf{A})$ based on node features \mathbf{X} and adjacency matrix \mathbf{A} . However, instead of directly working with the original graph, we would like to leverage sparsified subgraphs to remove task-irrelevant

information and learn more robust representations. In other words, we are interested in the variant as follows

$$p(Y | \mathbf{X}, \mathbf{A}) = \sum_{g \in \mathbb{S}_{\mathcal{G}}} p(Y | \mathbf{X}, g) p(g | \mathbf{X}, \mathbf{A}), \quad (1)$$

where $g \in \mathbb{S}_{\mathcal{G}}$ is a sparsified subgraph of original graph \mathcal{G} . In practice, the distribution space size of $\mathbb{S}_{\mathcal{G}}$ is $2^{|\mathcal{E}|}$, and it is intractable to enumerate all possible g as well as estimate the exact values of $p(Y | \mathbf{X}, g)$ and $p(g | \mathbf{X}, \mathbf{A})$. Therefore, we turn to approximate the distributions by two tractable parameterized functions $q_{\theta}(\cdot)$ and $q_{\phi}(\cdot)$, as follows

$$p(Y | \mathbf{X}, \mathbf{A}) = \sum_{g \in \mathbb{S}_{\mathcal{G}}} q_{\theta}(Y | \mathbf{X}, g) q_{\phi}(g | \mathbf{X}, \mathbf{A}), \quad (2)$$

where $q_{\theta}(\cdot)$ and $q_{\phi}(\cdot)$ are approximation functions for $p(Y | \mathbf{X}, g)$ and $p(g | \mathbf{X}, \mathbf{A})$. Moreover, to make the above graph sparsification process differentiable, we employ the commonly used reparameterization tricks [49] to transform the discrete combinatorial optimization problem to a continuous probabilistic generative model, as follows

$$\sum_{g \in \mathbb{S}_{\mathcal{G}}} q_{\theta}(Y | \mathbf{X}, g) q_{\phi}(g | \mathbf{X}, \mathbf{A}) \propto \sum_{g' \sim q_{\phi}(g | \mathbf{X}, \mathbf{A})} q_{\theta}(Y | \mathbf{X}, g'), \quad (3)$$

where g' is a subgraph sampled from $q_{\phi}(g | \mathbf{X}, \mathbf{A})$.

To optimize Eq. (3), a bi-level optimization framework is adopted to alternate between learning $q_{\phi}(g | \mathbf{X}, \mathbf{A})$ and $q_{\theta}(Y | \mathbf{X}, g')$. In addition to the downstream supervision for learning $q_{\theta}(Y | \mathbf{X}, g')$, another objective function $\mathcal{H}(\cdot)$ is required to optimize $q_{\phi}(g | \mathbf{X}, \mathbf{A})$. Intuitively, $\mathcal{H}(\cdot)$ should be an unsupervised metric to evaluate the quality of the sparsified graph g' . Graph homophily, as an important graph property, may be a desirable option for $\mathcal{H}(\cdot)$.

Introduction on Graph Homophily: The graph homophily is defined as the fraction of inter-class edges in a graph,

$$r = \frac{|\{(i, j) : e_{i,j} \in \mathcal{E} \wedge y_i \neq y_j\}|}{|\mathcal{E}|}. \quad (4)$$

The homophily ratio of the sparsified subgraph g' may be a desirable option for $\mathcal{H}(\cdot)$ for the following three reasons: (1) Most common graphs adhere to the principle of homophily, i.e., “birds of a feather flock together”, which suggests that connected nodes often belong to the same class, e.g., friends in social networks often share the same interests or hobbies. (2) It has been shown in previous work [50] that common GNN classifiers, such as GCN and GAT, usually perform better on datasets with higher homophily ratios. (3) When downstream supervision is not accessible, the prior knowledge of graph homophily can serve as strong guidance for searching the suitable sparsified subgraph.

Problem Statement: In this paper, the three important issues on framework design can be summarized as:

- Implementing sparsification network $q_{\phi}(g | \mathbf{X}, \mathbf{A})$ that takes node features \mathbf{X} and adjacency matrix \mathbf{A} as inputs to generate a sparsified subgraph g' . (Sec. III-B)
- Implementing self-contrasting network $q_{\theta}(Y | \mathbf{X}, g')$ that takes node features \mathbf{X} and the sparsified subgraph g' as inputs to make predictions Y . (Sec. III-C)

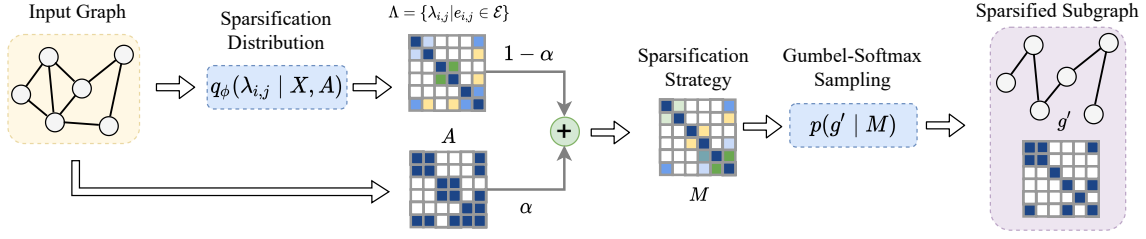


Fig. 2: Illustration of the proposed structural sparsification network, which consists of three main components: (1) Estimate $\Lambda = \{\lambda_{i,j} | i \in \mathcal{V}, j \in \mathcal{N}_i\}$ by sparsification distribution $q_\phi(\lambda_{i,j} | \mathbf{X}, \mathbf{A})$; (2) Obtain the sparsification strategy \mathbf{M} by the weighted fusion; (3) Sample a sparsified subgraph g' from sparsification strategy \mathbf{M} through Gumbel-Softmax sampling.

- Formulating the sparsification and self-contrasting in a bi-level optimization framework and proposing a homophily-oriented objective function (Sec. III-D)

B. Structural Sparsification Network

The sparsification network aims to generate a discrete sparsified subgraph g' for graph \mathcal{G} as shown in Fig. 2, serving as the approximation function $q_\phi(g | \mathbf{X}, \mathbf{A})$. Therefore, we need to answer three questions about the sparsification network: (1) How to model the sparsification distribution? (2) How to **differentiably** sample discrete sparsified subgraphs from the learned sparsification distribution? (3) How to optimize the sparsification distribution and subgraph sampling process? Next, we first answer *Question (1)(2)* and defer the discussion of optimization strategy until Sec. III-D.

1) **Sparsification Distribution:** To model the sparsification distribution, we introduce a set of latent variables $\Lambda = \{\lambda_{i,j} | e_{i,j} \in \mathcal{E}\}$, where $\lambda_{i,j} \in [0, 1]$ denotes the sparsification probability between node v_i and v_j . We can estimate $\lambda_{i,j}$ directly with distribution $q_\phi(\lambda_{i,j} | \mu_{i,j})$ parameterized by $\mu_{i,j} \in \mathbb{R}^F$. However, directly fitting each $q_\phi(\lambda_{i,j} | \mu_{i,j})$ locally involves solving $|\mathcal{E}|F$ parameters, which increases the over-fitting risk given the limited labels in the graph. Thus, we consider the amortization inference [51], which avoids the optimization of parameter $\mu_{i,j}$ for each local probability distribution $q_\phi(\lambda_{i,j} | \mu_{i,j})$ and instead fits a shared neural network to model parameterized posterior. Specifically, we first transform the input to a low-dimensional hidden space, done by multiplying the features of input nodes with a shared parameter matrix $\mathbf{W} \in \mathbb{R}^{F \times d}$, that is, $\mathbf{z}_i = \mathbf{W}x_i$. Then, we parameterize the sparsification distribution $\lambda_{i,j}$ as follows:

$$q_\phi(\lambda_{i,j} | \mathbf{X}, \mathbf{A}) = \sigma(\mathbf{z}_i \mathbf{z}_j^T), \quad (5)$$

where $\sigma(\cdot)$ is an element-wise sigmoid function.

2) **Subgraph Sampling:** To sample discrete sparsified subgraphs from the learned sparsification distribution and make the sampling process differentiable, we adopt Gumbel-Softmax sampling [49], which can be formulated as follows

$$g'_{i,j} = \left[\frac{1}{1 + \exp^{-(\log M_{i,j} + G)/\tau}} + \frac{1}{2} \right], \quad \text{where } i \in \mathcal{V}, j \in \mathcal{N}_i \quad (6)$$

where $M_{i,j} = (1 - \alpha)q_\phi(\lambda_{i,j} | \mathbf{X}, \mathbf{A}) + \alpha A_{i,j}$ is defined as the learned *structural sparsification strategy*. In addition,

$\alpha \in [0, 1]$ is the fusion factor, which aims to prevent the sampled sparsified subgraph g' from deviating too much from the original graph. Besides, τ is the distribution temperature, and $G \sim \text{Gumbel}(0, 1)$ is a gumbel random variate.

Next, we will discuss in detail how to model the STR-Contrast network $q_\theta(Y | \mathbf{X}, g')$ based on the node features \mathbf{X} and the sampled sparsified subgraph g' . Without loss of generality, we can denote the edge set of the sparsified graph g' as \mathcal{E}'_g to distinguish \mathcal{E} of the original graph \mathcal{G} .

C. Structural Self-Contrasting Network

Not involving any explicit message passing, the structural self-contrasting network treats the structural information implicitly as prior to guide the computation of supervision signals. In this section, we introduce the structural self-contrasting network from the following three aspects: (1) backbone architecture design, including an MLP and two prediction heads; (2) objective function design, how to self-contrast between the target node v_i and its neighboring node $v_j \in \mathcal{V}_i$; (3) optimization difficulty and strategy, including how to properly sample negative samples and support batch-style training. A high-level overview of the proposed GSSC framework is shown in Fig. 3.

1) **Architecture:** The structural self-contrasting network is based on a pure MLP architecture, with each layer composed of a linear transformation, an activation function, a batch normalization, and a dropout function, formulated as:

$$\mathbf{H}^{(l)} = \text{Dropout}(\text{BN}(\sigma(\mathbf{H}^{(l-1)} \mathbf{W}^{(l-1)}))), \quad \mathbf{H}^{(0)} = \mathbf{X} \quad (7)$$

where $1 \leq l \leq L$, $\sigma = \text{ReLU}(\cdot)$ denotes an activation function, $\text{BN}(\cdot)$ is the batch normalization, and $\text{Dropout}(\cdot)$ is the dropout function. $\mathbf{W}^{(0)} \in \mathbb{R}^{d \times F}$ and $\mathbf{W}^{(l)} \in \mathbb{R}^{F \times F}$ ($1 \leq l \leq L - 1$) are layer-specific weight matrices with the hidden dimension F . Furthermore, we define two additional prediction heads: $\mathbf{y}_i = f_\omega(\mathbf{h}_i^{(L)}) \in \mathbb{R}^C$ and $\mathbf{z}_j = g_\gamma(\mathbf{h}_j^{(L)}) \in \mathbb{R}^C$, where C is the number of categories.

2) **Structural Smoothness Constraint:** The structural smoothness assumption indicates that connected nodes should be similar, while disconnected nodes should be far away. With such motivation, we propose a structural smoothness constraint that enables the model to learn the graph connectivity and disconnectivity without explicit message passing. Given a target node v_i , we first generate an augmented node $v_{i \rightarrow j}$ between

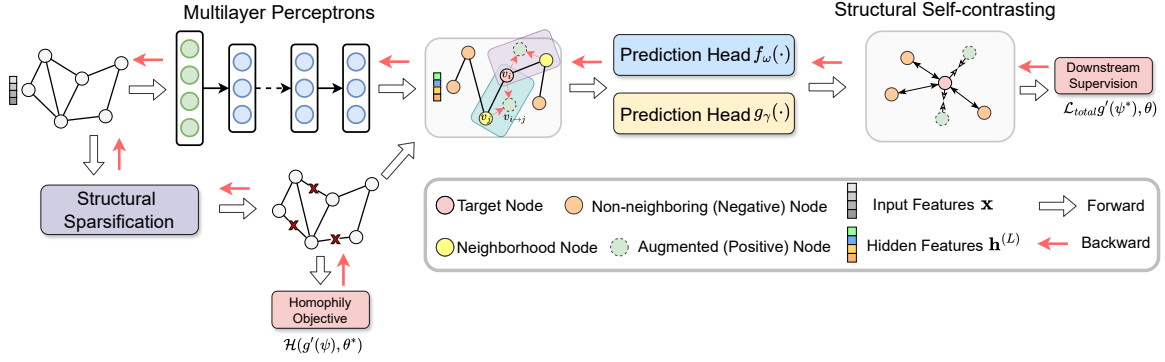


Fig. 3: Illustration of the proposed GSSC framework, consisting of a structural sparsification network, a multilayer perceptron, two label prediction heads ($f_\omega(\cdot)$ and $g_\gamma(\cdot)$), a structural self-contrasting network, as well as two optimization losses.

node v_i and its neighboring node $v_j \in \mathcal{N}_i$ by *learnable interpolation*, with its node representation $\mathbf{g}_{i \rightarrow j}$ defined as

$$\mathbf{g}_{i \rightarrow j} = g_\gamma(\beta_{i,j} \mathbf{h}_j^{(L)} + (1 - \beta_{i,j}) \mathbf{h}_i^{(L)}), \quad (8)$$

where $\beta_{i,j} = \text{sigmoid}(\mathbf{a}^T [\mathbf{h}_i^{(L)} \parallel \mathbf{h}_j^{(L)}])$

where $\beta_{i,j}$ is defined as *learnable interpolation coefficients* with the shared weight \mathbf{a} . Then, we take the generated augmented node $v_{i \rightarrow j}$ as a positive sample and other non-neighboring nodes as negative samples and define the constraint between nodes v_i, v_j as follows

$$l_{i,j} = \log \frac{e^{\mathcal{D}(\mathbf{y}_i, \mathbf{g}_{i \rightarrow j})}}{\sum_{e_{i,k} \notin \mathcal{E}'_g} e^{\mathcal{D}(\mathbf{y}_i, \mathbf{z}_k)}} \quad (9)$$

$$= \mathcal{D}(\mathbf{y}_i, \mathbf{g}_{i \rightarrow j}) - \log \sum_{e_{i,k} \notin \mathcal{E}'_g} e^{\mathcal{D}(\mathbf{y}_i, \mathbf{z}_k)},$$

where $\mathbf{y}_i = f_\omega(\mathbf{h}_i^{(L)})$, $\mathbf{z}_k = g_\gamma(\mathbf{h}_k^{(L)})$, and $\mathcal{D} : \mathbb{R}^C \times \mathbb{R}^C \rightarrow \mathbb{R}$ is a discriminator that maps two representations to an agreement score and taken as Mean Square Error (MSE) in our implementation by default. The motivations why we adopt learnable interpolation to augment nodes is based on the following judgment: compared with those non-neighboring nodes, the number of neighboring nodes is much smaller, which makes the model overemphasize the differences between the target and non-neighboring nodes, resulting in imprecise class boundaries. We have demonstrated the benefits of node augmentation and negative samples in Table. I.

The total structural smoothness constraints over the edge set \mathcal{E}'_g of the sparsified subgraph g' can be defined as

$$\mathcal{L}_{smooth} = \frac{1}{N} \sum_{i=1}^N \sum_{e_{i,j} \in \mathcal{E}'_g} \left(\mathcal{D}(\mathbf{y}_i, \mathbf{g}_{i \rightarrow j}) - \log \sum_{e_{i,k} \notin \mathcal{E}'_g} e^{\mathcal{D}(\mathbf{y}_i, \mathbf{z}_k)} \right). \quad (10)$$

3) Optimization Difficulty and Strategy: Directly optimizing Eq. (10) is computationally expensive for two tricky optimization difficulties: (1) it treats all non-neighboring nodes as negative samples, which suffers from both data redundancy and huge computational burden; and (2) it performs the summation over the entire set of nodes, i.e., requiring a large memory space for keeping the entire graph. To address these problems, we adopt the edge sampling strategy [52] for batch-style training. More specifically, we first sample a mini-batch of edges from the entire edge set \mathcal{E}'_g to construct a mini-batch $\mathcal{E}_b \in \mathcal{E}'_g$. Then we randomly sample negative nodes from a

pre-defined negative sample distribution $P_k(v)$ for each edge $e_{i,j} \in \mathcal{E}_b$ instead of enumerating all non-neighboring nodes as negative samples. Finally, we can rewrite Eq. (10) as follows

$$\mathcal{L}_{smooth} = \frac{1}{B} \sum_{b=1}^B \sum_{e_{i,j} \in \mathcal{E}_b} \left(\mathcal{D}(\mathbf{y}_i, \mathbf{g}_{i \rightarrow j}) + \mathcal{D}(\mathbf{y}_j, \mathbf{g}_{j \rightarrow i}) - \mathbb{E}_{v_k \sim P_k(v)} \left(\log e^{\mathcal{D}(\mathbf{y}_i, \mathbf{z}_k)} + \log e^{\mathcal{D}(\mathbf{y}_j, \mathbf{z}_k)} \right) \right), \quad (11)$$

where B is the batch size, and v_k is a random sample drawn from the pre-defined negative sample distribution $P_k(v_i) = \frac{d_i}{|\mathcal{E}'_g|}$ for each node v_i , where d_i is the degree of node v_i .

Similarly, we can formulate the cross-entropy loss on the labeled node set \mathcal{V}_L as a classification loss, as follows

$$\mathcal{L}_{cla} = \frac{1}{B} \sum_{b=1}^B \sum_{i \in \mathcal{V}_L \cap \mathcal{V}_b} \left(CE(y_i, \hat{\mathbf{y}}_i) + \sum_{e_{i,j} \in \mathcal{E}_b} CE(y_i, \hat{\mathbf{z}}_j) \right), \quad (12)$$

where $\mathcal{V}_b = \{v_i, v_j | e_{i,j} \in \mathcal{E}_b\}$ is all the sampled nodes in \mathcal{E}_b , $\hat{\mathbf{y}}_i = \text{softmax}(\mathbf{y}_i) \in \mathbb{R}^C$, and $\hat{\mathbf{z}}_j = \text{softmax}(\mathbf{z}_j) \in \mathbb{R}^C$. Besides, $CE(\cdot)$ is the cross-entropy loss, and y_i is the ground-truth label of node v_i . The total training loss is defined as:

$$\mathcal{L}_{total}(g'(\psi), \theta) = \mathcal{L}_{smooth} + \mathcal{L}_{cla}. \quad (13)$$

Note that Eq. (13) is defined for node classification and needs some minor modifications to be extended to graph-level tasks. Since there are no node labels in graph-level tasks, it requires replacing \mathcal{L}_{cla} in Eq. (13) with another loss \mathcal{L}_{graph} . Therefore, we aggregate the node embeddings in a graph into a graph embedding by average pooling, and then compute the loss between it and the graph label, e.g., MSE for a regression task, or cross-entropy for a classification task. The total training loss for graph-level tasks is $\mathcal{L}_{total}(g'(\psi), \theta) = \mathcal{L}_{smooth} + \mathcal{L}_{graph}$.

D. Optimization Strategy

1) Problem Statement: Since the structural information g' is not explicitly involved in the forward propagation in the STR-Contrast network in Eq. (11), we have $\frac{\partial \mathcal{L}_{smooth}}{\partial g'(\psi)} = 0$, i.e., the parameter ψ of the sparsification network cannot be directly optimized end-to-end through downstream supervision as many existing sparsification methods have done. A very straightforward idea is to directly modify Eq. (11) as follows

$$\mathcal{L}'_{smooth} = \frac{1}{B} \sum_{b=1}^B \sum_{e_{i,j} \in \mathcal{E}_b} g'_{i,j} \left(\mathcal{D}(\mathbf{y}_i, \mathbf{g}_{i \rightarrow j}) + \mathcal{D}(\mathbf{y}_j, \mathbf{g}_{j \rightarrow i}) - \mathbb{E}_{v_k \sim P_k(v)} \left(\log e^{\mathcal{D}(\mathbf{y}_i, \mathbf{z}_k)} + \log e^{\mathcal{D}(\mathbf{y}_j, \mathbf{z}_k)} \right) \right) \quad (14)$$

where Eq. (14) allows structural information g' to be explicitly involved in the computation of the supervision signal, so the sparsification network can now be directly optimized by loss $\mathcal{L}'_{smooth}(\cdot)$. However, optimizing Eq. (14) may result in trivial solutions, i.e., $\mathcal{L}'_{smooth}(\cdot)$ reaches a minimum by forcing $g'_{i,j}$ close to 0 rather than minimizing $l_{i,j}$, and thus an empty edge set \mathcal{E}'_g is learned. The experimental results in subsection IV-D confirm the potential trend towards trivial solutions. Therefore, we turn to another more sophisticated design by formulating structural sparsification and self-contrasting as a bi-level optimization problem, as follows

$$\max_{\psi} \mathcal{H}(g'(\psi), \theta^*), \text{ s.t. } \theta^* = \arg \min_{\theta} \mathcal{L}_{total}(g'(\psi), \theta) \quad (15)$$

where lower-level objective $\mathcal{L}_{total}(g'(\psi), \theta)$ is defined in Eq. (13), and upper-level objective $\mathcal{H}(g'(\psi), \theta^*)$ denotes the quality measure for the sparsified subgraph g' . However, as discussed earlier, we cannot directly employ the downstream performance to measure the sparsified subgraph, so we need an unsupervised quality measure $\mathcal{H}(\cdot)$ to evaluate the quality of the sparsified subgraph g' . In a nutshell, one of the most critical optimization difficulties is how to construct a proper optimization objective for structural sparsification without the direct access to downstream supervision.

2) **Homophily-oriented Objective:** Inspired by the discussions on homophily in Sec. III-A, we design a homophily-oriented objective and used it as a measure for the quality of the sparsified graph. The homophily-oriented objective is defined as follows

$$\max_{\psi} \mathcal{H}(g'(\psi), \theta^*) = \frac{\sum_i \sum_{j \in \mathcal{E}'_g} g'_{i,j} \cdot \mathbb{I}(s_i = s_j)}{|\mathcal{E}'_g|} \quad (16)$$

where $\mathbb{I}(\cdot)$ is an indicator function, $s_i = \operatorname{argmax}(\mathbf{y}_i)$ and $s_j = \operatorname{argmax}(\mathbf{y}_j)$ are the pseudo-labels obtained from self-contrasting network $q_{\theta^*}(Y | \mathbf{X}, g')$. Extensive experiments have been provided in Sec. IV-D to demonstrate the effectiveness of the homophily-oriented objective in Eq. (16).

3) **Bi-level optimization:** We adopt the alternating gradient descent (AGD) algorithm [53] to optimize Eq. (15), alternating between upper-level maximization and lower-level minimization, with the pseudo-code outlined in Algorithm 1.

Updating lower-level θ . The lower-level minimization follows the conventional gradient descent procedure given the sampled sparsified subgraph $g'(\psi^{(n-1)})$, represented as:

$$\theta^{(n)} = \theta^{(n-1)} - \alpha_{\theta} \nabla_{\theta} \mathcal{L}_{total}(g'(\psi^{(n-1)}), \theta^{(n-1)}) \quad (17)$$

Updating upper-level ψ . After updating parameter $\theta^{(n)}$, we perform the upper-level maximization and update $\psi^{(n-1)}$ as:

$$\psi^{(n)} = \psi^{(n-1)} + \alpha_{\psi} \nabla_{\psi} \mathcal{H}(g'(\psi^{(n-1)}), \theta^{(n)}) \quad (18)$$

where $\alpha_{\theta} \in \mathcal{R}_{>0}$ and $\alpha_{\psi} \in \mathcal{R}_{>0}$ are the learning rates.

E. Discussion and Comparison

1) **Discuss on Structural Self-contrasting Network:** Different from existing GNNs, such as GCN and GAT, that guide the feature aggregation among neighbors through powerful

Algorithm 1 AGD for bi-level optimization

- 1: Initialize sparsification network parameter $\phi^{(0)}$ and neighborhood self-contrasting network parameter $\theta^{(0)}$.
 - 2: **for** $n \in \{1, \dots, N\}$ **do**
 - 3: Generate sparsified subgraph $g'(\psi^{(n-1)})$ by Eq. (5-6).
 - 4: Compute $\mathcal{L}_{total}(g'(\psi^{(n-1)}), \theta^{(n-1)})$ by Eq. (13);
 - 5: **Lower-level maximization:** Fix parameter $\psi^{(n-1)}$, and update parameter $\theta^{(n)}$ by Eq. (17).
 - 6: Compute $\mathcal{H}(g'(\psi^{(n-1)}), \theta^{(n)})$ by Eq. (16);
 - 7: **Upper-level minimization:** Fix parameter $\theta^{(n)}$, and update parameter $\psi^{(n)}$ by Eq. (18).
 - 8: **end for**
 - 9: **return** Predicted labels \mathcal{Y}_U and optimized parameter $\theta^{(N)}$
-

message passing, the structural self-contrasting network is based purely on a multilayer perceptron where structural information is only implicitly used as prior in the computation of supervision signals, but does not explicitly involve in the forward propagation. Another research topic that is close to us is graph contrastive learning [54, 55], but we differ from them in the three aspects: (1) learning objective, graph contrastive learning aims to learn transferable knowledge from abundant unlabeled data in an *unsupervised* setting and then generalize the learned knowledge to downstream tasks. Instead, the structural self-contrasting network works in a *semi-supervised* setting, i.e., the partial label information is available during training. (2) augmentation, graph contrastive learning usually requires multiple types of sophisticated augmentation to obtain different views for contrasting [56–58]. However, we augment nodes only by simple linear interpolation. (3) We remove those noisy edges by the sparsification network, which can be considered as an “adaptive” edge-wise self-contrasting.

2) **Discuss on the structural sparsification network:** Different from existing graph sparsification methods that are either attention-based or gradient-based, our STR-Sparse formulates graph sparsification learning as a two-stage process based on a probabilistic generative model, where (1) *in the first stage*, learning the sparsification distribution and modeling the sparsification strategy; (2) *in the second stage*, sampling sparsified subgraphs from the learned sparsification strategy through Gumbel-Softmax sampling, which transforms the problem from a discrete combinatorial optimization to a continuous subgraph generation problem.

IV. EXPERIMENTS

A. Experimental setups

The experiments aim to answer the following six questions:

- (Q1) How effective is GSSC compared to those GNNs, graph sparsification, and MLP-based models?
- (Q2) Is GSSC robust to label noise and structure noise?
- (Q3) How effective is the homophily-oriented objective?
- (Q4) Is STR-Sparse applicable to other GNNs and MLP-based models? How does STR-Contrast perform with other existing graph sparsification methods?
- (Q5) How efficient is the model in terms of inference time?
- (Q6) How do hyperparameters affect performance?

TABLE I: Classification accuracy \pm std (%) of general GNNs, graph sparsification, graph transformer, and MLP-based models, where the metrics marked in **bold**, *underline*, and **gray** are the top results for rank 1, 2, and 3 (same for Table. IV and Table. I).

Type	Method	Cora	Citeseer	Coauthor-CS	Coauthor-Phy	ogbn-arxiv	ogbn-proteins
GNNs	GCN [22]	81.28 \pm 0.42	71.06 \pm 0.44	88.66 \pm 0.48	92.14 \pm 0.34	71.74 \pm 0.29	72.51 \pm 0.35
	GAT [64]	83.02 \pm 0.45	72.56 \pm 0.51	89.28 \pm 0.63	92.40 \pm 0.52	73.65\pm0.11	73.44 \pm 0.41
	GraphSAGE [21]	82.22 \pm 0.80	71.22 \pm 0.58	89.18 \pm 0.45	91.54 \pm 0.54	69.83 \pm 0.25	72.81 \pm 0.46
	SGC [23]	80.88 \pm 0.47	71.84 \pm 0.72	88.56 \pm 0.60	90.92 \pm 0.62	67.79 \pm 0.27	70.31 \pm 0.23
	APPNP [24]	83.28 \pm 0.33	71.74 \pm 0.27	89.72 \pm 0.59	92.54 \pm 0.59	71.14 \pm 0.28	73.17 \pm 0.40
	DAGNN [1]	84.30 \pm 0.51	73.14 \pm 0.62	90.20 \pm 0.61	93.02 \pm 0.72	71.91 \pm 0.23	73.75 \pm 0.36
Graph Sparsification	SS/RD [39, 40]	79.42 \pm 0.64	70.43 \pm 0.52	87.89 \pm 0.64	91.75 \pm 0.47	67.27 \pm 0.23	72.13 \pm 0.45
	DropEdge [13]	82.23 \pm 0.51	72.14 \pm 0.63	88.95 \pm 0.57	92.43 \pm 0.63	71.84 \pm 0.24	72.78 \pm 0.54
	LDS [41]	83.14 \pm 0.56	72.34 \pm 0.70	89.24 \pm 0.48	92.72 \pm 0.38	-	-
	NeuralSparse [8]	83.78 \pm 0.54	<u>73.19\pm0.56</u>	89.63 \pm 0.50	92.88 \pm 0.43	71.45 \pm 0.26	73.54 \pm 0.39
	GAUG [14]	83.53 \pm 0.38	<u>72.86\pm0.33</u>	89.90 \pm 0.45	93.23 \pm 0.36	72.12 \pm 0.25	74.13 \pm 0.51
Graph Transformer	NAGformer [65]	84.24 \pm 0.68	72.45 \pm 0.46	89.37 \pm 0.46	92.71 \pm 0.79	70.10 \pm 0.28	73.64 \pm 0.71
	SGFormer [63]	<u>84.50\pm0.90</u>	72.60 \pm 0.20	<u>90.42\pm0.51</u>	<u>93.46\pm0.74</u>	72.63 \pm 0.13	79.53\pm0.38
MLP-based Model	MLP	61.86 \pm 0.43	59.76 \pm 0.51	83.12 \pm 0.53	86.24 \pm 0.66	55.50 \pm 0.23	72.04 \pm 0.48
	Graph-MLP [11]	81.45 \pm 0.52	72.87 \pm 0.70	89.80 \pm 0.68	91.85 \pm 0.49	OOM	OOM
	LinkDist [12]	76.70 \pm 0.47	65.19 \pm 0.55	89.56 \pm 0.58	92.36 \pm 0.70	69.24 \pm 0.26	72.45 \pm 0.42
	GSSC (ours)	85.28\pm0.42	73.74\pm0.48	91.00\pm0.40	94.40\pm0.86	72.90 \pm 0.25	79.25 \pm 0.31
	w/o Augmentation	84.12 \pm 0.48	73.10 \pm 0.53	90.27 \pm 0.51	93.39 \pm 0.90	<u>72.70\pm0.24</u>	<u>78.90\pm0.35</u>
w/o Negative Samples	83.42 \pm 0.63	72.64 \pm 0.55	89.80 \pm 0.62	92.76 \pm 0.78	72.26 \pm 0.22	78.34 \pm 0.42	

1) *Datasets*: The experiments are conducted on ten real-world datasets. For the four small-scale datasets, namely Cora [59], Citeseer [60], Coauthor-CS, and Coauthor-Phy [61], we follow [1, 22] to select 20 nodes per class to construct a training set, 500 nodes for validation, and 1000 nodes for testing. For the ogbn-arxiv and ogbn-proteins datasets, we use the public data splits provided by the authors [62]. For the four heterophily graphs, namely Actor, Squirrel, Chameleon, and Deezer, we follow the data splitting strategy by [63].

2) *Baseline*: We consider the following six general GNN models: GCN [22], GAT [64], GraphSAGE [21], SGC [23], APPNP [24], and DAGNN [1]. Besides, we compare the proposed framework with Graph-MLP [11] and LinkDist [1], both of which are based on a pure MLP architecture. Furthermore, two current state-of-the-art graph transformers, SGFormer [63] and NAGformer [65], are also included in the comparison. Besides, six graph sparsification methods are included as baselines: Spectral Sparsifier (SS) [39], Rank Degree (RD) [40], DropEdge [13], LDS [41], STR-Sparse [8], and GAUG [14]. When not mentioned specifically, we default to taking the experimental settings of the original paper for implementing these six methods and adopting GCN as the backbone. Each set of experiments is run five times with different random seeds, and the averages and standard deviations are reported.

3) *Hyperparameter*: The following hyperparameters are set the same for all datasets: Adam optimizer with learning rate $\alpha_\theta = \alpha_\psi = 0.01$ and weight decay $decay = 5e-4$; Epoch $E = 200$; Layer number $L = 2$. The other dataset-specific hyperparameters are determined by an AutoML toolkit NNI with the search spaces as: hidden dimension $F = \{256, 512, 1024\}$; batch size $B = \{512, 1024, 4096\}$, fusion factor $\alpha = \{0.1, 0.3, 0.5\}$; temperature $\tau = \{0.1, 0.5, 0.8, 1.0\}$. Moreover, we follow [14] to first warm-up the structural self-contrasting network with the loss \mathcal{L}_{total} for 100 epochs in the original graph \mathcal{G} .

TABLE II: Classification accuracy \pm std (%) on 4 heterophily graphs, where the results of various baselines come from [63].

Method	Actor	Squirrel	Chameleon	Deezer
GCN	30.1 \pm 0.2	38.6 \pm 1.8	41.3 \pm 3.0	62.7 \pm 0.7
GAT	29.8 \pm 0.6	35.6 \pm 2.1	39.2 \pm 3.1	61.7 \pm 0.8
SGC	27.0 \pm 0.2	39.3 \pm 2.3	39.0 \pm 3.3	62.3 \pm 0.4
APPNP	31.1 \pm 1.5	35.3 \pm 1.9	38.4 \pm 3.5	66.1 \pm 0.6
H2GCN [50]	34.4 \pm 1.7	35.1 \pm 1.2	38.1 \pm 4.0	66.2 \pm 0.8
SIGN [66]	36.5 \pm 1.0	40.7 \pm 2.5	41.7 \pm 2.2	66.3 \pm 0.3
GloGNN [67]	36.4 \pm 1.6	35.7 \pm 1.3	40.2 \pm 3.9	65.8 \pm 0.8
SGFormer [63]	37.9\pm1.1	41.8 \pm 2.2	44.9 \pm 3.9	<u>67.1\pm1.1</u>
GSSC (ours)	<u>37.5\pm0.5</u>	43.1\pm1.5	45.7\pm3.6	67.6\pm0.9

B. Performance for Node&Graph Classification (Q1)

To answer Q1, we conduct experiments on six real-world datasets in comparison to other baselines, where SS/RD is the combinational method of Spectral Sparsifier (SS) and Rank Degree (RD). It can be seen from Table. I that (1) While Graph-MLP and LinkDist can achieve comparable performance with GCN on a few datasets, they still lag far behind the state-of-the-art GNN models, such as APPNP and DAGNN, and cannot even match the performance of GraphSAGE and GAT on some datasets. (2) More importantly, the performance of Graph-MLP and LinkDist is hardly comparable to that of graph sparsification methods except for SS/RD, which indicates that these MLP-based models have no advantage in terms of classification performance on clean data. (3) Instead, our model consistently achieves the best overall performance on 5 of 6 datasets, significantly outperforming other GNN models and graph sparsification methods, and is even comparable to two state-of-the-art graph transformers. For example, our model obtains the best performance on two large-scale ogbn-arxiv and ogbn-proteins datasets, and more notably, it outperforms DAGNN by 0.99% and 5.50%, respectively.

Furthermore, to evaluate the effectiveness of node aug-

TABLE III: Classification accuracy \pm std (%) on 4 graphs for molecular property prediction, where all baselines use GCN as the backbone by default, and their results come from [68].

Method	BBBP	Tox21	MUV	HIV
GPT-GNN [69]	65.3 \pm 1.5	74.3 \pm 0.7	75.6 \pm 1.8	74.8 \pm 1.4
GraphCL [70]	69.9 \pm 1.6	75.1 \pm 0.8	75.1 \pm 1.5	74.5 \pm 0.6
GraphLoG [71]	66.4 \pm 2.8	73.9 \pm 1.4	73.5 \pm 1.0	75.5 \pm 0.5
G-Motif [72]	70.3 \pm 1.5	73.1 \pm 0.4	74.5 \pm 0.6	76.0 \pm 1.3
GraphMAE [73]	66.3 \pm 0.7	68.8 \pm 0.5	71.8 \pm 0.3	73.5 \pm 0.8
SimGRACE [74]	66.7 \pm 0.6	74.3 \pm 0.2	74.3 \pm 0.8	74.4 \pm 1.0
GraphMVP [75]	67.9 \pm 1.0	74.6 \pm 0.3	75.5 \pm 1.6	76.2 \pm 0.8
GSSC (ours)	70.8\pm0.7	75.8\pm0.6	76.3\pm1.0	77.9\pm1.1

mentation and negative samples described in Sec. III-C, we perform the **ablation study** without negative samples and augmentation (fixing $\beta_{i,j} = 1$), respectively. It can be observed from Table. I that both negative samples and augmentation contribute to improving classification performance. In addition, the performance improvements brought by negative samples outweigh the node augmentation across various datasets.

To evaluate how well GSSC handles heterophily graphs, we compare GSSC with four models specializing in heterophily graphs, i.e., H2GCN [50], SIGN [66], GloGNN [67], and SGFormer [63]. The experimental results in Table. II show that GSSC achieves the best performance on 3 of the 4 datasets, and is second only to SGFormer on the Actor dataset.

We further evaluate how well GSSC handles the graph classification task by molecular property prediction. There are four datasets, BBBP, Tox21, MUV, and HIV [62] are considered. Besides, seven classical graph self-supervised models are included as baselines for comparison. Table. III shows that GSSC outperforms the other baselines on all four datasets, demonstrating the potential of GSSC for graph-level tasks.

C. Evaluation on Robustness (Q2)

To demonstrate the model’s robustness, we evaluate the model under different ratios of (1) label noise and (2) structure perturbations on the Cora and Citeseer datasets.

1) **Performance with Noisy Labels:** The performance with various label noise ratios $r \in \{20\%, 40\%, 60\%\}$ is reported in Table. IV for two types of label noise: symmetric and asymmetric [76, 77]. The symmetric noise indicates that label i ($0 \leq i \leq C - 1$) of each training sample changes independently with probability $\frac{r}{|C|-1}$ to another class j ($j \neq i$), but with probability $1 - r$ preserved as label i ; the asymmetric noise indicates that label i flips independently with probability r to fixed class j ($j = (i + 1)\%C$), but with probability $1 - r$ preserved as label i . It can be seen from Table. IV that (1) The Graph-MLP, as a classical MLP-based model, significantly outperforms various GNN models under extremely high label noise, but it slightly falls behind DAGNN and GAUG at low-noise settings. (2) GSSC performs most robustly under various label noise types and ratios, especially with asymmetric noise and high noise ratios. For example, with $r = 60\%$ asymmetric noise, our model outperforms DAGNN by 4.55% and 7.19% on the Cora and Citeseer datasets, respectively.

2) **Performance with Corrupted Structures:** The classification performance under different structural perturbation ratios $r \in \{10\%, 20\%, 30\%\}$ is reported in Table. V, where the corrupted structures are obtained by randomly removing and adding $r \cdot |\mathcal{E}|$ edges from the original graph for training. It can be seen from Table. V that (1) MLP-based models, such as Graph-MLP, are more advantageous at extremely high ratios of structural perturbations. (2) GSSC is more robust than other baselines under various structural perturbation ratios, especially under severe perturbations. For example, when $r = 30\%$, GSSC outperforms NeuralSparse by 1.55% and 3.06% on the Cora and Citeseer datasets, respectively.

To further demonstrate the advantages of GSSC in terms of robustness, we select a few baselines that perform well on two large-scale datasets, ogbn-arxiv and ogbn-proteins, as shown in Table. I. We compare the performance of GSSC with these baselines under structural perturbations in Table. VI. It can be seen that GSSC is only comparable to those traditional GNNs and GTs on clean data without structural noise. However, models that work well on clean data, e.g., GAT and SGFormer, may fail on noisy data; for example, GAT performs poorer than GCN on the ogbn-arxiv dataset. By contrast, our GSSC performs best under noisy data, showing superior robustness.

D. Evaluation on Homophily Objective (Q3)

To evaluate the effectiveness of homophily-oriented objective $\mathcal{H}(\cdot)$, we conduct a more in-depth analysis through the following two sets of experiments: (1) correlation analysis between the homophily ratio of the sparsified graph and the downstream performance and (2) evolution curves of the edge number, homophily ratio (in the learned sparsified graph), and downstream performance during the training process.

1) **Correlation Analysis:** We randomly remove a fraction of edges from the original graph to obtain a set of candidate subgraphs with different homophily ratios and then train a structural self-contrasting network **from scratch** on each candidate subgraph. The homophily ratio of the candidate sparsified subgraphs and their corresponding downstream performance are visualized in Fig. 4, from which we can observe that the downstream performance tends to be better on the sparsified subgraph with a higher homophily ratio. The tight correlation between homophily ratio and classification accuracy inspires us that homophily ratio may be a desirable option for evaluating the quality of sparsified subgraphs.

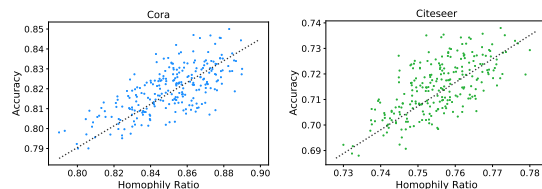


Fig. 4: Correlation between accuracy and homophily ratio.

2) **Training Evolution Visualization:** We visualize in Fig. 5(a) the evolution of edge number, homophily ratio (in the learned sparsified graph), and downstream performance during

TABLE IV: Classification accuracy \pm std (%) with different label noise types and ratios.

Dataset	Flipping-Rate	GCN	GAT	GraphSAGE	APPNP	DAGNN	DropEdge	LDS	NeuralSparse	GAUG	Graph-MLP	LinkDist	GSSC (ours)
Cora	symmetric 20%	77.77 \pm 0.62	79.75 \pm 0.92	78.38 \pm 0.68	79.53 \pm 0.56	80.76 \pm 0.59	78.16 \pm 0.65	78.90 \pm 0.70	79.64 \pm 0.62	79.94 \pm 0.66	79.21 \pm 0.64	77.21 \pm 0.59	81.36\pm0.59
	symmetric 40%	69.39 \pm 0.70	72.69 \pm 0.78	71.33 \pm 0.81	74.38 \pm 0.76	75.23 \pm 0.86	70.48 \pm 0.74	72.25 \pm 0.81	73.10 \pm 0.93	73.38 \pm 0.78	75.57 \pm 0.76	72.66 \pm 0.81	76.98\pm0.40
	symmetric 60%	52.17 \pm 0.93	55.16 \pm 0.81	53.99 \pm 0.89	58.87 \pm 1.04	63.10 \pm 0.88	53.72 \pm 0.75	56.45 \pm 0.94	57.71 \pm 1.03	56.78 \pm 0.77	64.35 \pm 0.90	57.21 \pm 1.06	67.38\pm0.72
	asymmetric 20%	71.97 \pm 0.97	73.63 \pm 0.83	73.58 \pm 1.05	74.53 \pm 0.69	76.50 \pm 0.92	72.62 \pm 0.97	73.80 \pm 1.10	74.55 \pm 0.85	74.78 \pm 0.85	74.59 \pm 0.70	72.48 \pm 0.84	79.72\pm0.69
	asymmetric 40%	64.07 \pm 0.58	64.24 \pm 0.78	63.86 \pm 0.68	65.99 \pm 0.69	67.18 \pm 0.66	64.59 \pm 0.75	66.20 \pm 0.72	67.36 \pm 0.68	66.84 \pm 0.81	67.85 \pm 0.94	66.31 \pm 0.80	69.40\pm1.16
	asymmetric 60%	38.47 \pm 0.95	39.38 \pm 0.99	39.49 \pm 0.78	40.39 \pm 1.01	42.61 \pm 0.81	38.74 \pm 1.14	40.18 \pm 0.92	40.87 \pm 1.03	41.52 \pm 0.88	43.54 \pm 1.00	41.52 \pm 1.16	47.16\pm1.18
Citeseer	symmetric 20%	66.91 \pm 0.58	67.61 \pm 0.59	67.34 \pm 0.43	68.20 \pm 0.48	71.25 \pm 0.61	67.30 \pm 0.54	68.84 \pm 0.68	69.22 \pm 0.64	69.10 \pm 0.47	70.53 \pm 0.50	62.51 \pm 0.58	72.66\pm0.48
	symmetric 40%	61.65 \pm 0.59	63.88 \pm 0.46	62.21 \pm 0.58	65.61 \pm 0.58	69.32 \pm 0.77	62.85 \pm 0.59	64.58 \pm 0.65	65.84 \pm 0.73	65.64 \pm 0.70	69.52 \pm 0.69	61.23 \pm 0.80	71.68\pm0.75
	symmetric 60%	54.83 \pm 0.63	55.26 \pm 0.90	54.20 \pm 0.58	55.84 \pm 0.56	59.36 \pm 0.77	55.05 \pm 0.74	57.25 \pm 0.64	58.29 \pm 0.69	58.46 \pm 0.81	61.79 \pm 0.78	57.35 \pm 0.73	65.06\pm0.82
	asymmetric 20%	65.38 \pm 0.89	66.62 \pm 0.85	66.52 \pm 0.70	68.17 \pm 0.96	68.61 \pm 0.89	65.72 \pm 0.76	66.80 \pm 0.81	67.42 \pm 0.85	67.74 \pm 0.79	67.93 \pm 0.95	60.62 \pm 0.87	72.74\pm0.74
	asymmetric 40%	55.70 \pm 1.07	56.42 \pm 0.81	56.60 \pm 0.99	57.63 \pm 0.79	60.39 \pm 0.86	56.10 \pm 0.77	57.86 \pm 0.90	58.65 \pm 0.83	58.93 \pm 0.70	61.76 \pm 0.83	54.30 \pm 1.02	67.32\pm0.78
	asymmetric 60%	41.90 \pm 0.98	43.70 \pm 1.15	42.65 \pm 0.58	45.15 \pm 0.91	46.05 \pm 0.87	42.21 \pm 0.68	43.85 \pm 0.72	44.75 \pm 0.74	44.48 \pm 0.83	48.25 \pm 1.12	44.09 \pm 1.20	53.24\pm1.09

TABLE V: Classification accuracy \pm std (%) under structure perturbations on two small-scale datasets, Cora and Citeseer.

Method	Cora			Citeseer		
	10%	20%	30%	10%	20%	30%
GCN	74.19 \pm 0.96	68.69 \pm 0.67	63.82 \pm 0.54	64.84 \pm 0.69	62.87 \pm 0.93	60.51 \pm 0.73
GAT	75.01 \pm 0.97	69.62 \pm 0.51	64.76 \pm 0.74	63.35 \pm 0.89	61.81 \pm 0.96	58.57 \pm 1.23
GraphSAGE	74.72 \pm 0.69	69.02 \pm 0.50	64.14 \pm 0.94	63.38 \pm 0.67	62.80 \pm 0.66	59.54 \pm 0.77
APPNP	74.36 \pm 0.71	70.02 \pm 0.93	64.90 \pm 1.05	63.88 \pm 0.93	61.56 \pm 1.08	58.32 \pm 0.69
DAGNN	75.32 \pm 0.85	70.41 \pm 0.84	65.60 \pm 0.82	64.74 \pm 0.65	61.92 \pm 0.91	58.96 \pm 1.19
DropEdge	74.42 \pm 0.88	69.58 \pm 0.75	64.12 \pm 0.76	65.04 \pm 0.85	62.99 \pm 0.95	60.89 \pm 0.92
LDS	74.75 \pm 0.93	69.89 \pm 0.71	64.47 \pm 0.81	65.32 \pm 0.73	63.43 \pm 1.10	61.13 \pm 1.05
NeuralSparse	75.23 \pm 0.79	70.32 \pm 0.68	65.45 \pm 0.72	66.10 \pm 0.78	63.82 \pm 0.87	61.54 \pm 1.30
GAUG	75.46 \pm 0.90	70.73 \pm 0.80	64.83 \pm 0.85	65.70 \pm 0.80	63.65 \pm 0.84	61.90 \pm 0.88
Graph-MLP	75.23 \pm 0.74	70.90 \pm 0.78	65.74 \pm 0.83	65.78 \pm 0.80	64.20 \pm 1.21	62.65 \pm 1.10
LinkDist	73.92 \pm 0.80	69.30 \pm 0.88	64.36 \pm 0.98	64.55 \pm 0.75	61.37 \pm 1.05	58.47 \pm 0.94
GSSC (ours)	77.42\pm0.95	71.98\pm0.81	67.00\pm0.59	69.70\pm0.89	67.06\pm0.84	64.60\pm0.68

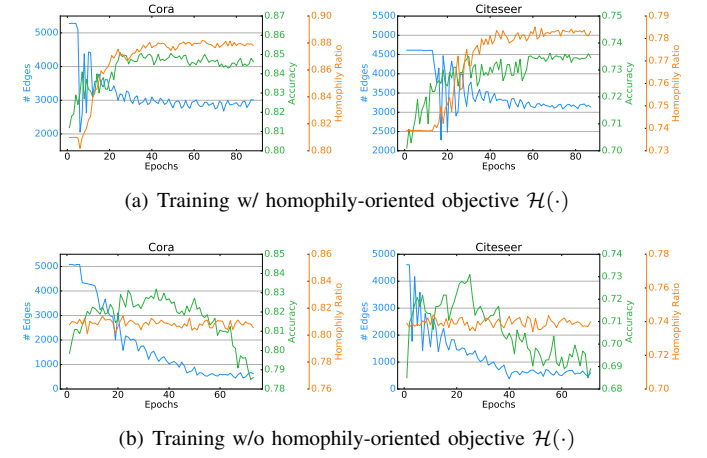
TABLE VI: Classification accuracy \pm std (%) under structure perturbations on two large-scale datasets, arxiv and proteins.

Method	ogbn-arxiv		ogbn-proteins	
	0%	20%	0%	20%
GCN	71.74 \pm 0.29	63.53 \pm 0.34	72.51 \pm 0.35	64.30 \pm 0.44
GAT	73.65\pm0.11	62.70 \pm 0.23	73.44 \pm 0.41	65.74 \pm 0.50
GraphSAGE	69.83 \pm 0.25	62.49 \pm 0.41	72.81 \pm 0.46	64.93 \pm 0.38
DAGNN	71.91 \pm 0.23	63.74 \pm 0.42	73.75 \pm 0.36	66.45 \pm 0.56
NeuralSparse	71.45 \pm 0.26	64.51 \pm 0.36	73.54 \pm 0.39	67.16 \pm 0.46
GAUG	72.12 \pm 0.25	65.04 \pm 0.21	74.13 \pm 0.51	67.47 \pm 0.64
NAGformer	70.10 \pm 0.28	63.40 \pm 0.35	73.64\pm0.71	66.21 \pm 0.51
SGFormer	72.63 \pm 0.13	64.28 \pm 0.38	79.53 \pm 0.38	69.55 \pm 0.43
GSSC (ours)	<u>72.90\pm0.25</u>	66.25\pm0.28	<u>79.25\pm0.31</u>	70.20\pm0.42

training. It is clear that the homophily-oriented objective $\mathcal{H}(\cdot)$ can effectively increase the homophily ratio while reducing the number of edges, i.e., making the graph sparse. More importantly, the downstream performance also improves steadily as the homophily ratio increases until it finally converges. To further analyze the role of homophily-oriented objective $\mathcal{H}(\cdot)$, we also train the model with the objective defined in Eq. (14) instead of $\mathcal{H}(\cdot)$ as a comparison. It can be seen that Eq. (14) may result in *trivial solutions*, i.e., the number of edges decreases sharply to a small value with the homophily ratio unchanged as shown in Fig. 5(b). More importantly, we observe that their downstream performance first improves as the edge number decreases and then drops sharply on the obtained over-sparsified graph, which suggests that over-sparsification is harmful to model performance.

E. Analysis on STR-Sparse & STR-Contrast (Q4)

1) *Evaluation on STR-Sparse*: Table. VII evaluates whether STR-Sparse is applicable to other models, where

Fig. 5: Visualization of the training process *with* (Top) and *without* (Bottom) homophily-oriented objective $\mathcal{H}(\cdot)$.

six GNN models are jointly optimized with STR-Sparse by downstream supervision and three MLP-based models are optimized by the proposed homophily-oriented objective $\mathcal{H}(\cdot)$ in Eq. 16. We make the following observations: (1) STR-Sparse consistently improves performance across various models compared to the results reported in Table. I; (2) MLP-based models benefit more from STR-Sparse than GNNs.

TABLE VII: Accuracy \pm std (%) of GNNs and MLP-based models with Structural Sparsification (STR-Sparse).

Model	Cora	Citeseer	Actor	Coauthor-CS	Coauthor-Phy
GCN	81.28 \pm 0.42	71.06 \pm 0.44	24.84 \pm 0.56	88.66 \pm 0.48	92.14 \pm 0.34
+ STR-Sparse	83.45 \pm 0.60	72.96 \pm 0.52	25.89 \pm 0.70	89.52 \pm 0.64	92.84 \pm 0.49
GAT	83.02 \pm 0.45	72.56 \pm 0.51	26.28 \pm 0.45	89.28 \pm 0.63	92.40 \pm 0.52
+ STR-Sparse	83.92 \pm 0.61	73.30 \pm 0.68	27.88 \pm 0.56	89.82 \pm 0.79	93.13 \pm 0.65
GraphSAGE	82.22 \pm 0.80	71.22 \pm 0.58	26.54 \pm 0.70	89.18 \pm 0.45	91.54 \pm 0.54
+ STR-Sparse	83.70 \pm 0.79	73.05 \pm 0.63	28.05 \pm 0.86	90.08 \pm 0.61	92.63 \pm 0.69
SGC	80.88 \pm 0.47	71.84 \pm 0.72	25.24 \pm 0.55	88.56 \pm 0.60	90.92 \pm 0.62
+ STR-Sparse	82.61 \pm 0.54	72.62 \pm 0.56	26.42 \pm 0.71	89.37 \pm 0.66	91.90 \pm 0.73
APPNP	83.28 \pm 0.33	71.74 \pm 0.27	27.82 \pm 1.02	89.72 \pm 0.59	92.54 \pm 0.59
+ STR-Sparse	84.08 \pm 0.51	73.18 \pm 0.57	28.91 \pm 0.88	90.18 \pm 0.64	93.40 \pm 0.57
DAGNN	84.30 \pm 0.51	73.14 \pm 0.62	28.98 \pm 0.86	90.20 \pm 0.61	93.02 \pm 0.72
+ STR-Sparse	84.52 \pm 0.48	73.55 \pm 0.74	30.19 \pm 0.79	90.49 \pm 0.72	93.88 \pm 0.76
Graph-MLP	81.45 \pm 0.52	72.87 \pm 0.70	25.40 \pm 0.49	89.80 \pm 0.68	91.85 \pm 0.49
+ STR-Sparse	83.50 \pm 0.46	73.38 \pm 0.69	29.25 \pm 0.75	90.22 \pm 0.73	93.17 \pm 0.69
LinkDist	76.70 \pm 0.47	65.19 \pm 0.55	23.96 \pm 0.65	89.56 \pm 0.58	92.36 \pm 0.70
+ STR-Sparse	80.05 \pm 0.55	71.20 \pm 0.66	27.59 \pm 0.80	90.48 \pm 0.64	93.46 \pm 0.77
GSSC (w/o STR-Sparse)	83.16 \pm 0.50	71.89 \pm 0.78	27.96 \pm 0.76	89.41 \pm 0.48	92.65 \pm 0.49
+ STR-Sparse	85.28\pm0.42	73.74\pm0.48	31.24\pm1.09	91.00\pm0.40	94.40\pm0.86

2) *Evaluation on STR-Contrast*: Table. VIII evaluates the applicability of different graph sparsification methods to STR-

TABLE VIII: Accuracy \pm std (%) of the STR-Contrast network with other graph sparsification methods, where "N/A" denotes the results without any sparsification as a baseline.

Sparsifier	Cora	Citeseer	Actor	Coauthor-CS	Coauthor-Phy
N/A (STR-Contrast)	83.16 \pm 0.50	71.89 \pm 0.78	27.96 \pm 0.76	89.41 \pm 0.48	92.65 \pm 0.49
+ SS/RD	80.50 \pm 0.76	69.74 \pm 0.81	24.26 \pm 0.69	86.32 \pm 0.62	89.58 \pm 0.76
+ DropEdge	83.42 \pm 0.62	72.81 \pm 0.52	28.69 \pm 0.54	89.73 \pm 0.47	93.96 \pm 0.58
+ LDS	83.56 \pm 0.54	72.75 \pm 0.44	28.90 \pm 0.62	89.84 \pm 0.49	93.17 \pm 0.61
+ NeuralSparse	84.08 \pm 0.42	73.22 \pm 0.47	29.82 \pm 0.68	90.18 \pm 0.55	93.44 \pm 0.51
+ GAUG	84.15 \pm 0.45	73.16 \pm 0.39	30.37 \pm 0.56	89.95 \pm 0.40	93.63 \pm 0.43
+ STR-Sparse	85.28\pm0.42	73.74\pm0.48	31.24\pm1.09	91.00\pm0.40	94.40\pm0.86

Contrast, where SS/RD is the combinational method of Spectral Sparsifier (SS) and Rank Degree (RD). The three supervised sparsification methods, LDS, STR-Sparse, and GAUG, are optimized by the homophily-oriented objective $\mathcal{H}(\cdot)$ for a fair comparison. We make the following observations: (1) Compared with the vanilla STR-Contrast model (**without any sparsification, denoted as "N/A"**), all graph sparsification methods except SS/RD can improve generalization; (2) Compared with unsupervised sparsification methods, such as DropEdge, STR-Sparse achieves up to 1.86% and 2.55% improvements on the Cora and Actor datasets. Even when compared to supervised sparsification methods, such as STR-Sparse and GAUG, STR-Sparse still shows a huge advantage.

F. Evaluation on Complexity Analysis (Q5)

The time complexity of the GSSC framework mainly comes from two main parts: (1) STR-Sparse $\mathcal{O}(|\mathcal{V}|dF + |\mathcal{E}|F)$ and (2) STR-Contrast $\mathcal{O}(|\mathcal{V}|dF + |\mathcal{E}'_g|F)$, with a total complexity of $\mathcal{O}(|\mathcal{V}|dF + (|\mathcal{E}| + |\mathcal{E}'_g|)F)$, where $|\mathcal{E}|$ and $|\mathcal{E}'_g|$ are the number of edges in the original and sparsified graph. Due to graph sparsification, we have $|\mathcal{E}'_g| \ll |\mathcal{E}|$, so the total complexity is linear w.r.t the number of nodes $|\mathcal{V}|$ and edges $|\mathcal{E}|$, which is in the same order of magnitude as GCN. However, with the removal of neighborhood-fetching latency, the inference time complexity can be reduced from $\mathcal{O}(|\mathcal{V}|dF + |\mathcal{E}|F)$ of GCN to $\mathcal{O}(|\mathcal{V}|dF)$. The inference time (ms) averaged over 30 runs is reported in Fig. 6, where all methods use $L = 2$ layers and hidden dimension $F = 16$. Besides, all baselines are implemented based on the standard implementation in the DGL library [78] using the PyTorch 1.6.0 library on NVIDIA v100 GPU. In a fair comparison, we observe that GSSC achieves the fastest inference speed on all datasets.

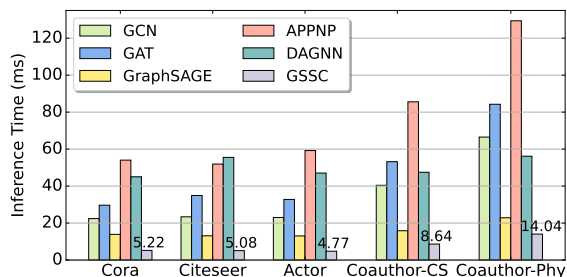


Fig. 6: Inference time (ms) for various methods.

G. Hyperparameter Sensitivity Analysis (Q6)

We evaluate the hyperparameter sensitivity with respect to two key hyperparameters: fusion factor α and batch size B , and the results are reported in Fig. 7 and Fig. 8, respectively. In practice, we can determine α and B by selecting the model with the highest accuracy on the validation set.

1) *Fusion Factor*: As can be observed in Fig. 7, fusion factor α is crucial for the proposed framework. If we set $\alpha = 1$, i.e., removing the structural sparsification, the performance is usually the poorest compared with other settings. In practice, we find that setting α to a small value, e.g., $\alpha = 0.1$ usually produces pretty good performance. However, a too-small α may cause the sparsified subgraph to deviate too much from the original graph on some datasets, resulting in poor performance. For example, on the Coauthor-CS dataset, the model achieves much better performance when setting α as 0.3 than 0.1. In our experiments, we have only tested with the smallest $\alpha = 0.1$, and further performance improvements are expected by finer-grained hyperparameter search.

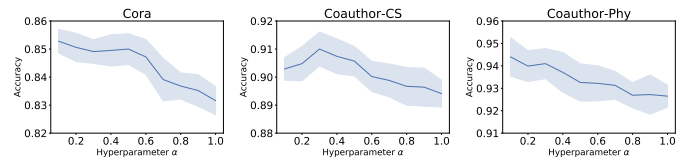


Fig. 7: Parameter sensitivity analysis on fusion factor α .

2) *Batch Size*: Fig. 8 shows the performance of GSSC trained with batch size $B \in \{256, 512, 1024, 2048, 4096\}$, from which we observe that B is a dataset-specific hyperparameter. For simple graphs with few nodes and edges, such as Cora and Citeseer, a small batch size, $B = 256$ or 512, can yield fairly good performance. However, for large-scale graphs, such as Coauthor-CS and Coauthor-Phy, the performance improves with the increase of batch size B .

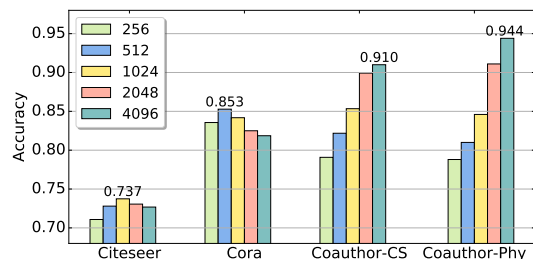


Fig. 8: Parameter sensitivity analysis on batch size B .

V. CONCLUSION

In this paper, we propose a novel *Graph Structural Self-Contrasting* (GSSC) framework to train graph data without **explicit** message passing. The GSSC framework is based purely on MLPs, where structural information is only **implicitly** incorporated to guide the computation of supervision signals. We formulate structural sparsification as a probabilistic generative model and then perform self-contrasting in the sparsified

neighborhood to learn robust representations. To address the problem that the sparsification process cannot be directly optimized through back-propagation of downstream supervision, we propose a homophily-oriented objective to optimize the structural sparsification. Finally, structural sparsification and self-contrasting are formulated in a bi-level optimization framework. Despite the great progress, limitations still exist; for example exploring the applicability of the GSSC framework to other graph types, such as temporal and heterogeneous graphs, may be a promising direction for future work.

VI. ACKNOWLEDGEMENT

This work was supported by National Science and Technology Major Project of China (No. 2022ZD0115101), National Natural Science Foundation of China Project (No. U21A20427), and Project (No. WU2022A009) from the Center of Synthetic Biology and Integrated Bioengineering of Westlake University.

REFERENCES

- [1] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 338–348.
- [2] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, 2020.
- [3] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.
- [4] R. A. Rossi, R. Zhou, and N. K. Ahmed, "Deep inductive graph representation learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 3, pp. 438–452, 2018.
- [5] L. Wu, Y. Tian, Y. Huang, S. Li, H. Lin, N. V. Chawla, and S. Z. Li, "Mape-ppi: Towards effective and efficient protein-protein interaction prediction via microenvironment-aware protein embedding," *arXiv preprint arXiv:2402.14391*, 2024.
- [6] L. Wu, Y. Tian, H. Lin, Y. Huang, S. Li, N. V. Chawla, and S. Z. Li, "Learning to predict mutation effects of protein-protein interactions by microenvironment-aware hierarchical prompt learning," *arXiv preprint arXiv:2405.10348*, 2024.
- [7] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2847–2856.
- [8] C. Zheng, B. Zong, W. Cheng, J. Ni, W. Yu, H. Chen, and W. Wang, "Robust graph representation learning via neural sparsification," in *International Conference on Machine Learning*. PMLR, 2020, pp. 11 458–11 468.
- [9] F. Feng, X. He, J. Tang, and T.-S. Chua, "Graph adversarial training: Dynamically regularizing based on graph structure," *IEEE Transactions on Knowledge and Data Engineering*, 2019.
- [10] S. Freitas, D. Yang, S. Kumar, H. Tong, and D. H. Chau, "Graph vulnerability and robustness: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 6, pp. 5915–5934, 2022.
- [11] Y. Hu, H. You, Z. Wang, Z. Wang, E. Zhou, and Y. Gao, "Graph-mlp: Node classification without message passing in graph," *arXiv preprint arXiv:2106.04051*, 2021.
- [12] Y. Luo, A. Chen, K. Yan, and L. Tian, "Distilling self-knowledge from contrastive links to classify graph nodes without passing messages," *arXiv preprint arXiv:2106.08541*, 2021.
- [13] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropege: Towards deep graph convolutional networks on node classification," *arXiv preprint arXiv:1907.10903*, 2019.
- [14] T. Zhao, Y. Liu, L. Neves, O. Woodford, M. Jiang, and N. Shah, "Data augmentation for graph neural networks," *arXiv preprint arXiv:2006.06830*, 2020.
- [15] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2020.
- [16] H. Li, X. Wang, Z. Zhang, and W. Zhu, "Ood-gnn: Out-of-distribution generalized graph neural network," *IEEE Transactions on Knowledge and Data Engineering*, 2022.
- [17] L. Wu, H. Lin, B. Hu, C. Tan, Z. Gao, Z. Liu, and S. Z. Li, "Beyond homophily and homogeneity assumption: Relation-based frequency adaptive graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.
- [18] L. Wu, H. Lin, Y. Huang, and S. Z. Li, "Knowledge distillation improves graph structure augmentation for graph neural networks," *Advances in Neural Information Processing Systems*, vol. 35, pp. 11 815–11 827, 2022.
- [19] L. Wu, H. Lin, Z. Gao, C. Tan, S. Li *et al.*, "Graph-mixup: Improving class-imbalanced node classification on graphs by self-supervised context prediction," *arXiv preprint arXiv:2106.11133*, 2021.
- [20] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *arXiv preprint arXiv:1606.09375*, 2016.
- [21] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Neural information processing systems*, 2017, pp. 1024–1034.
- [22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
- [23] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International conference on machine learning*. PMLR, 2019, pp. 6861–6871.
- [24] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," *arXiv preprint arXiv:1810.05997*, 2018.
- [25] L. Wu, H. Lin, Y. Huang, and S. Z. Li, "Quantifying the knowledge in gnns for reliable distillation into mlps," in *International Conference on Machine Learning*. PMLR,

- 2023, pp. 37 571–37 581.
- [26] L. Wu, H. Lin, Y. Huang, T. Fan, and S. Z. Li, “Extracting low-/high-frequency knowledge from graph neural networks and injecting it into mlps: An effective gnn-to-mlp distillation framework,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 9, 2023, pp. 10 351–10 360.
- [27] X. He, B. Hooi, T. Laurent, A. Perold, Y. LeCun, and X. Bresson, “A generalization of vit/mlp-mixer to graphs,” in *International conference on machine learning*. PMLR, 2023, pp. 12 724–12 745.
- [28] V. P. Dwivedi and X. Bresson, “A generalization of transformer networks to graphs,” *arXiv preprint arXiv:2012.09699*, 2020.
- [29] G. Mialon, D. Chen, M. Selosse, and J. Mairal, “Graphit: Encoding graph structure in transformers,” *arXiv preprint arXiv:2106.05667*, 2021.
- [30] L. Ma, C. Lin, D. Lim, A. Romero-Soriano, P. K. Dokania, M. Coates, P. Torr, and S.-N. Lim, “Graph inductive biases in transformers without message passing,” in *International Conference on Machine Learning*. PMLR, 2023, pp. 23 321–23 337.
- [31] C. Lv, M. Qi, X. Li, Z. Yang, and H. Ma, “Sgformer: Semantic graph transformer for point cloud-based 3d scene graph generation,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 5, 2024, pp. 4035–4043.
- [32] V. T. Hoang, O. Lee *et al.*, “A survey on structure-preserving graph transformers,” *arXiv preprint arXiv:2401.16176*, 2024.
- [33] S. Pan, R. Hu, S.-f. Fung, G. Long, J. Jiang, and C. Zhang, “Learning graph embedding with adversarial training methods,” *IEEE transactions on cybernetics*, vol. 50, no. 6, pp. 2475–2487, 2019.
- [34] H. Jin and X. Zhang, “Latent adversarial training of graph convolution networks,” in *ICML workshop on learning and reasoning with graph-structured representations*, vol. 2, 2019.
- [35] J. Xia, H. Lin, L. Wu, Z. Gao, S. Li, and S. Z. Li, “Towards robust graph neural networks against label noise,” 2020.
- [36] B. Jiang, Z. Zhang, D. Lin, J. Tang, and B. Luo, “Semi-supervised learning with graph learning-convolutional networks,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11 313–11 320.
- [37] J. Klicpera, S. Weißenberger, and S. Günnemann, “Diffusion improves graph learning,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [38] D. Yu, R. Zhang, Z. Jiang, Y. Wu, and Y. Yang, “Graph-revised convolutional network,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2020, pp. 378–393.
- [39] V. Sadhanala, Y.-X. Wang, and R. Tibshirani, “Graph sparsification approaches for laplacian smoothing,” in *Artificial Intelligence and Statistics*. PMLR, 2016, pp. 1250–1259.
- [40] E. Voudigari, N. Salamanos, T. Papageorgiou, and E. J. Yannakoudakis, “Rank degree: An efficient algorithm for graph sampling,” in *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2016, pp. 120–129.
- [41] L. Franceschi, M. Niepert, M. Pontil, and X. He, “Learning discrete structures for graph neural networks,” in *International conference on machine learning*. PMLR, 2019, pp. 1972–1982.
- [42] L. Wu, C. Tan, Z. Liu, Z. Gao, H. Lin, and S. Z. Li, “Learning to augment graph structure for both homophily and heterophily graphs,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2023, pp. 3–18.
- [43] L. Wu, H. Lin, Z. Liu, Z. Liu, Y. Huang, and S. Z. Li, “Homophily-enhanced self-supervision for graph structure learning: Insights and directions,” *IEEE Transactions on Neural Networks and Learning Systems*, 2023.
- [44] M. Gu, G. Yang, S. Zhou, N. Ma, J. Chen, Q. Tan, M. Liu, and J. Bu, “Homophily-enhanced structure learning for graph clustering,” in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 2023, pp. 577–586.
- [45] B. Fatemi, L. El Asri, and S. M. Kazemi, “Slaps: Self-supervision improves structure learning for graph neural networks,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 22 667–22 681, 2021.
- [46] S. Zhang, Y. Xiong, Y. Zhang, Y. Sun, X. Chen, Y. Jiao, and Y. Zhu, “Rdgs: Dynamic graph representation learning with structure learning,” in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 2023, pp. 3174–3183.
- [47] C. Wei, J. Liang, D. Liu, and F. Wang, “Contrastive graph structure learning via information bottleneck for recommendation,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 20 407–20 420, 2022.
- [48] Y. Zhu, W. Xu, J. Zhang, Q. Liu, S. Wu, and L. Wang, “Deep graph structure learning for robust representations: A survey,” *arXiv preprint arXiv:2103.03036*, 2021.
- [49] E. Jang, S. Gu, and B. Poole, “Categorical reparameterization with gumbel-softmax,” *arXiv preprint arXiv:1611.01144*, 2016.
- [50] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra, “Beyond homophily in graph neural networks: Current limitations and effective designs,” *Advances in neural information processing systems*, vol. 33, pp. 7793–7804, 2020.
- [51] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.
- [52] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Advances in neural information processing systems*, 2013, pp. 3111–3119.
- [53] J. Wang, T. Zhang, S. Liu, P.-Y. Chen, J. Xu, M. Fardad, and B. Li, “Towards a unified min-max framework for adversarial exploration and robustness,” *arXiv preprint arXiv:1906.03563*, 2019.
- [54] L. Wu, H. Lin, C. Tan, Z. Gao, and S. Z. Li, “Self-

- supervised learning on graphs: Contrastive, generative, or predictive,” *IEEE Transactions on Knowledge and Data Engineering*, 2021.
- [55] Y. Liu, M. Jin, S. Pan, C. Zhou, Y. Zheng, F. Xia, and S. Y. Philip, “Graph self-supervised learning: A survey,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 6, pp. 5879–5900, 2022.
- [56] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, “Graph contrastive learning with augmentations,” *Advances in Neural Information Processing Systems*, vol. 33, 2020.
- [57] Y. Jiao, Y. Xiong, J. Zhang, Y. Zhang, T. Zhang, and Y. Zhu, “Sub-graph contrast for scalable self-supervised graph representation learning,” *arXiv preprint arXiv:2009.10273*, 2020.
- [58] K. Hassani and A. H. Khasahmadi, “Contrastive multi-view representation learning on graphs,” in *International Conference on Machine Learning*. PMLR, 2020, pp. 4116–4126.
- [59] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, “Collective classification in network data,” *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.
- [60] C. L. Giles, K. D. Bollacker, and S. Lawrence, “Citeseer: An automatic citation indexing system,” in *Proceedings of the third ACM conference on Digital libraries*, 1998, pp. 89–98.
- [61] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, “Pitfalls of graph neural network evaluation,” *arXiv preprint arXiv:1811.05868*, 2018.
- [62] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, “Open graph benchmark: Datasets for machine learning on graphs,” *arXiv preprint arXiv:2005.00687*, 2020.
- [63] Q. Wu, W. Zhao, C. Yang, H. Zhang, F. Nie, H. Jiang, Y. Bian, and J. Yan, “Simplifying and empowering transformers for large-graph representations,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [64] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *arXiv preprint arXiv:1710.10903*, 2017.
- [65] J. Chen, K. Gao, G. Li, and K. He, “Nagphormer: A tokenized graph transformer for node classification in large graphs,” *arXiv preprint arXiv:2206.04910*, 2022.
- [66] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti, “Sign: Scalable inception graph neural networks,” *arXiv preprint arXiv:2004.11198*, 2020.
- [67] X. Li, R. Zhu, Y. Cheng, C. Shan, S. Luo, D. Li, and W. Qian, “Finding global homophily in graph neural networks when meeting heterophily,” in *International Conference on Machine Learning*. PMLR, 2022, pp. 13 242–13 256.
- [68] T. Fan, L. Wu, Y. Huang, H. Lin, C. Tan, Z. Gao, and S. Z. Li, “Decoupling weighing and selecting for integrating multiple graph pre-training tasks,” *arXiv preprint arXiv:2403.01400*, 2024.
- [69] Z. Hu, Y. Dong, K. Wang, K.-W. Chang, and Y. Sun, “Gpt-gnn: Generative pre-training of graph neural networks,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1857–1867.
- [70] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, “Graph contrastive learning with augmentations,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 5812–5823. [Online]. Available: <https://proceedings.neurips.cc/paper/2020/file/3fe230348e9a12c13120749e3f9fa4cd-Paper.pdf>
- [71] M. Xu, H. Wang, B. Ni, H. Guo, and J. Tang, “Self-supervised graph-level representation learning with local and global structure,” in *International Conference on Machine Learning*. PMLR, 2021, pp. 11 548–11 558.
- [72] Y. Rong, Y. Bian, T. Xu, W. Xie, Y. Wei, W. Huang, and J. Huang, “Self-supervised graph transformer on large-scale molecular data,” *Advances in Neural Information Processing Systems*, vol. 33, 2020.
- [73] Z. Hou, X. Liu, Y. Cen, Y. Dong, H. Yang, C. Wang, and J. Tang, “Graphmae: Self-supervised masked graph autoencoders,” in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 594–604.
- [74] J. Xia, L. Wu, J. Chen, B. Hu, and S. Z. Li, “Simgrace: A simple framework for graph contrastive learning without data augmentation,” in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1070–1079.
- [75] S. Liu, H. Wang, W. Liu, J. Lasenby, H. Guo, and J. Tang, “Pre-training molecular graph representation with 3d geometry,” in *International Conference on Learning Representations*, 2022. [Online]. Available: <https://openreview.net/forum?id=xQUe1pOKPam>
- [76] C. Tan, J. Xia, L. Wu, and S. Z. Li, “Co-learning: Learning from noisy labels with self-supervision,” in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, pp. 1405–1413.
- [77] J. Xia, H. Lin, Y. Xu, C. Tan, L. Wu, S. Li, and S. Z. Li, “Gnn cleaner: Label cleaner for graph structured data,” *IEEE Transactions on Knowledge and Data Engineering*, 2023.
- [78] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. He, G. Karypis, J. Li, and Z. Zhang, “Deep graph library: A graph-centric, highly-performant package for graph neural networks,” *arXiv preprint arXiv:1909.01315*, 2019.