EMPOWERING GRAPH NEURAL NETWORKS FROM A DATA-CENTRIC VIEW

By

Wei Jin

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science —Doctor of Philosophy

2023

ABSTRACT

Many learning tasks in Artificial Intelligence (AI) require dealing with graph data, ranging from biology and chemistry to finance and education. As powerful learning tools for graph inputs, graph neural networks (GNNs) have demonstrated remarkable performance in various applications such as recommender systems and drug discovery. Recent research has primarily focused on model-centric approaches to enhance GNN performance by modifying model architectures while keeping the dataset fixed. However, these approaches have limitations, particularly in terms of robustness and scalability. For example, these approaches often yield suboptimal performance when confronted with limited high-quality data. Moreover, training GNNs is often computationally expensive on large-scale data; and such cost becomes even prohibitive when we need to train numerous models on the same dataset, such as hyper-parameter and architecture search. Given the challenges arising from data, a crucial question arises: *Can we address these problems directly from a data perspective?*

This dissertation presents a data-centric view that directly optimizes the given dataset to improve the performance of imperfect GNN models. Instead of modifying the model architectures, the data-centric view advocates for a set of techniques in graph dataset optimization to enhance the effectiveness and efficiency of GNNs. First, we demonstrate the potential to improve the quality of a graph dataset, enabling GNNs to exhibit robustness against severe noise and attacks. Furthermore, we showcase the possibility of substantially reducing the size of a graph dataset while preserving its information, thereby significantly decreasing the training cost. Unlike model-centric approaches that are typically specific to a single model, data-centric approaches yield improved datasets that benefit various existing models simultaneously. By embracing this data-centric perspective, this dissertation not only addresses crucial challenges associated with data quality and efficiency but also unlocks new opportunities for next-generation AI systems.

To my parents and family for their love and support.

ACKNOWLEDGEMENTS

First and foremost, I am deeply grateful to my advisor, Dr. Jiliang Tang, for his invaluable advice, inspiration, encouragement, and support throughout my Ph.D. journey. Dr. Tang is not only an advisor in academia but also a sincere friend who guided me with his knowledge and experience in many aspects of life. Joining Dr. Tang's research lab has been an invaluable experience, and I consider myself fortunate for the decision I made four years ago. Throughout this period, I have gained a wealth of knowledge from him, ranging from effective research practices to charismatic leadership skills. Dr. Tang is one of the greatest advisors in the world, and I always appreciate his supportiveness and responsiveness in fostering the growth of his students. I could not have achieved everything I have without his continuous support over the years.

I would like to express my deepest appreciation to my dissertation committee members, Dr. Anil K. Jain, Dr. Charu C. Aggarwal, Dr. Neil Shah, Dr. Sijia Liu, and Dr. Yuying Xie for their insightful comments and helpful suggestions. Their expertise and feedback have played a pivotal role in shaping both this dissertation and my job talk.

I was fortunate to work as an intern at Snap Research and Amazon with amazing colleagues and mentors: Dr. Neil Shah, Dr. Tong Zhao, Lingxiao Zhao, Shichang Zhang, Zhichun Guo, William Shiao, Xiaotian Han, Yozen Liu, and Leonardo Neves from Snap Research; Dr. Xianfeng Tang, Dr. Haoming Jiang, Dr. Zheng Li, Dr. Chen Luo, Hanqing Lu, Dr. Zhengyang Wang, Dr. Ruirui Li, Dr. Zhen Li, Monica Xiao Cheng, Dr. Danqing Zhang, Rahul Goutam, Haiyang Zhang, Dr. Karthik Subbian, and Bing Yin. I enjoyed the wonderful and productive years with you.

I am grateful to have had the pleasure and fortune of having supportive and encouraging colleagues during my Ph.D. I am thankful to all my colleagues from the Data Science and Engineering Lab: Jamell Dacon, Dr. Wenqi Fan, Dr. Jiangtao Huang, Dr. Hamid Karimi, Dr. Tyler Derr, Juanhui Li, Dr. Yao Ma, Norah Alfadhli, Xochitl Weiss, Yaxin Li, Dr. Haochen Liu, Hua Liu, Dr. Xiaorui Liu, Namratha Shah, Wentao Wang, Xiaoyang Wang, Dr. Xin Wang, Yiqi Wang, Dr. Zhiwei Wang, Han Xu, Dr. Xiangyu Zhao, Harry Shomer, Hongzhi Wen, Haitao Mao, Jiayuan Ding, Haoyu Han, Yingqian Cui, Jie Ren, Kaiqi Yang, Hang Li, Yuxuan Wan, Zhikai Chen, Kai Guo, Wenzhuo Tang,

and Remy Liu. In particular, thanks to my DSE collaborators: Dr. Wenqi Fan, Dr. Jiangtao Huang, Dr. Tyler Derr, Dr. Yao Ma, Juanhui Li, Haochen Liu, Hua Liu, Dr. Xiaorui Liu, Wentao Wang, Xiaoyang Wang, Dr. Xin Wang, Dr. Yiqi Wang, Dr. Han Xu, Yaxin Li, Haitao Mao, Dr. Haochen Liu, Haoyu Han, Hongzhi Wen, Jiayuan Ding, Dr. Hamid Karimi, Wenzhuo Tang, Remy Liu, Zhikai Chen, and Dr. Xiangyu Zhao. I would like to extend my sincere thanks to other collaborators: Dr. Suhang Wang, Dr. Yizhou Sun, Dr. Leman Akoglu, Dr. Enyan Dai, Yu Wang, Julian Venegas, Runze Su, Dr. Dylan Molho, Zhaoheng Li, Yixin Wang, Dr. Zitao Liu, Dr. Hui Liu, Dr. Qing Li, Dr. Qing Li, Dr. Xianfeng Tang, Dr. Shuiwang Ji, Dr. Xin Wang. I look forward to continued collaboration with you.

I would also like to extend my heartfelt appreciation to my dear friends at MSU who provided unwavering support and encouragement during the moments when I felt frustrated during my Ph.D. study. Jiaxin Yang, Kaicheng Chen, Saera Oh, Yunqi Tan, Bocheng Chen, Lanpeng Li, Maolin Gan, Junwen Chen, Gen Li, Yidong Ren, Yuguang Yao, Chenning Li, Junyuan Hong, Li Liu, Juexing Wang, and Zhiyu Xue, your friendship and belief in me have been truly invaluable.

Finally, I would like to thank my family, for their unconditional love and support.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Motivation

Graphs are ubiquitous data structures that describe pairwise relations between objects, ranging from biology and chemistry to finance and education. Due to their prevalence, graphs play a key role in various real-world applications for Artificial Intelligence (AI). For example, by exploiting graph structural information, we can predict the chemical property of a given molecular graph [170], detect fraud activities in a financial transaction graph [141], or recommend new friends to users in a social network [39]. To capture the rich information in graph data, graph neural networks (GNNs) [78, 139, 4, 156] have been developed and they have demonstrated remarkable performance in various graph-related tasks including node classification [78, 97], graph classification [160], and link prediction [193, 38].

Despite the promise of GNNs, they also face significant limitations, particularly in scalability and robustness. For example, GNNs tend to yield less satisfying performance in the absence of a sufficiently large amount of high-quality data, and their vulnerability to adversarial attacks can further diminish their effectiveness, as observed in various studies [67, 208, 210]. Moreover, training GNNs is often computationally expensive on large-scale data [57, 183, 54, 29]; and such cost becomes even prohibitive when we need to train numerous models on the same dataset, e.g., searching for the best hyper-parameters and architectures [198]. In response to these challenges, significant efforts have been made on developing new techniques from the modeling perspective, e.g., designing new architectures and modifying the training losses, while keeping the data unchanged. However, this model-centric paradigm overlooks the potential of directly enhancing data quality and efficiency, as both the data and model play critical roles in graph machine learning systems (as shown in Figure 1.1). The presence of these issues raises a question at the heart of this dissertation: *Can we empower GNNs from a data perspective, thereby complementing the existing efforts in model development?*

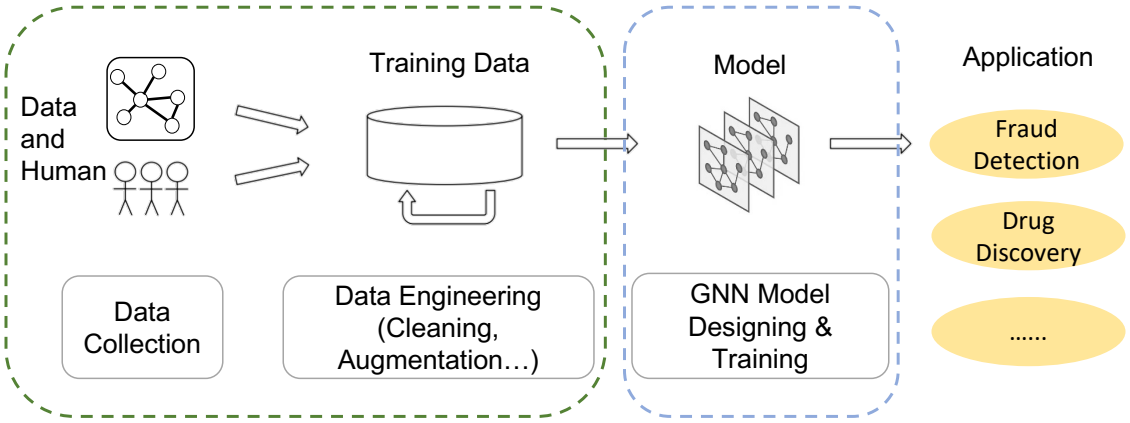


Figure 1.1: The pipeline of graph machine learning systems. Both model and data are important aspects of the systems.

Addressing this question necessitates an exploration of data-centric methods for graph data, which resides within the rising research field of Data-Centric AI [88, 180, 150, 179, 178, 116, 90]. Rather than devising new model architectures, we concentrate on refining the provided dataset to serve the existing models more effectively and efficiently. This innovative learning approach aims to deliver higher quality data, potentially leading to substantial improvements in AI system performance. Furthermore, since data-centric methods produce an optimized dataset as their output, they offer the advantage of enhancing a variety of existing models that utilize the improved dataset, unlike model-centric approaches that typically cater to a specific model. Given these advantages, the development of data-centric methodologies to augment GNNs holds promising potential.

1.2 Dissertation Contributions

To expand the horizon of GNN research beyond models and emphasize the significance of datasets, we introduce a comprehensive set of techniques for optimizing graph datasets during model training or inference. On the one hand, we present two methods for reducing the size of a graph dataset while retaining its essential information, leading to substantial reductions in training costs. On the other hand, we present two methods on enhancing the quality of a graph dataset to enable GNNs to exhibit robustness against severe noise and attacks. The major contributions of this dissertation can be summarized as follows:

- With the prevalence of large-scale graphs in real-world applications, concerns have been raised

regarding the storage and training time required for graph neural networks. To address these concerns, we introduce the concept of graph condensation [72] for graph neural networks for the first time. Graph condensation distills the original large graph into a smaller, synthetic graph that retains high informativeness while achieving comparable performance to GNNs trained on the original graph. To tackle the condensation problem, we propose GCond to learn synthetic graphs that can lead to a similar model training trajectory to the one trained on the original graph. This is accomplished by simultaneously condensing node features and structural information through the optimization of a gradient matching loss.

- Despite the promise of the proposed GCond, it has two inherent limitations. First, the condensation process is computationally expensive due to the involved nested optimization. Second, it does not generate discrete graph structures that could provide additional benefits in terms of storage efficiency. To address these limitations, we explore efficient dataset condensation and introduce a method called DosCond [71]. In DosCond, we model the discrete graph structure as a probabilistic model. Additionally, we propose a one-step gradient matching scheme that performs gradient matching for a single step without training the network weights. Our theoretical analysis demonstrates that this strategy can generate synthetic graphs that result in low classification loss on real graphs.
- Recent research has revealed the vulnerability of graph neural networks (GNNs) to adversarial attacks, which are carefully-crafted perturbations. This susceptibility to adversarial attacks raises concerns regarding the application of GNNs in safety-critical domains. Consequently, it is crucial to mitigate the negative impact of such attacks by purifying the data. We have identified that adversarial attacks during the training phase undermine important properties of graphs. For instance, real-world graphs often exhibit low-rank and sparse structures, with similar features observed among adjacent nodes. Remarkably, adversarial attacks tend to violate these intrinsic graph properties. Motivated by this observation, we propose a novel framework called Pro-GNN [70] to learn a clean graph structure from the perturbed graph by recovering these underlying graph properties. Extensive experiments demonstrate that the Pro-GNN achieves

significantly better performance compared with the state-of-the-art defense methods, even when the graph is heavily perturbed.

- In contrast to Pro-GNN, which defends against training-time attacks, we introduce a new data-centric approach called GTrans [73], which focuses on refining suboptimal test graphs to enhance the performance of pre-trained models. GTrans involves a test-time graph transformation framework that minimizes a parameter-free surrogate loss to improve the quality of the test graph data. This framework can be combined with any pre-trained graph neural networks (GNNs), and the refined graph data can be utilized with any model, thanks to its favorable transferability. One notable aspect of GTrans is its ability to provide interpretability. By visualizing the data, GTrans can reveal the types of graph modifications that lead to performance improvements. This interpretability offers valuable insights into the factors affecting the model's performance.

1.3 Dissertation Structure

The remainder of this dissertation is organized as follows. In Chapter 2, we formally define the problem of graph condensation and introduce a framework to tackle this challenging problem, which condenses a large-real graph into a small-synthetic one while preserving most of the original information. In Chapter 3, we provide theoretical analysis on the effectiveness of graph condensation framework and propose a highly efficient condensation approach by only performing one-step gradient matching. In Chapter 4, we identified that training-time adversarial attack essentially violates important graph properties. Based on this observation, we develop a framework to effectively restore the clean graph structure from the perturbed graph data while learning the model parameters at the same time. This framework is shown to be able to defend against various graph adversarial attacks and learn cleaner graph structures. In Chapter 5, we investigate the test-time robustness of graph neural networks under out-of-distribution data, abnormal features, and adversarial attacks. we develop a test-time graph transformation framework that transforms the test graph data by minimizing a parameter-free surrogate loss. Furthermore, we provide a rigorous theoretical understanding of the framework and verify its effectiveness empirically. We conclude the dissertation and discuss the broader impact and promising research directions in Chapter 6.

CHAPTER 2

GRAPH CONDENSATION FOR GRAPH NEURAL NETWORKS

Given the prevalence of large-scale graphs in real-world applications, the storage and time for training neural models have raised increasing concerns. To alleviate the concerns, we propose and study the problem of *graph condensation* for graph neural networks (GNNs). Specifically, we aim to condense the large, original graph into a small, synthetic and highly-informative graph, such that GNNs trained on the small graph and large graph have comparable performance. We approach the condensation problem by imitating the GNN training trajectory on the original graph through the optimization of a gradient matching loss and design a strategy to condense node features and structural information simultaneously. Extensive experiments have demonstrated the effectiveness of the proposed framework in condensing different graph datasets into informative smaller graphs. In particular, we are able to approximate the original test accuracy by 95.3% on Reddit, 99.8% on Flickr and 99.0% on Citeseer, while reducing their graph size by more than 99.9%, and the condensed graphs can be used to train various GNN architectures.

2.1 Introduction

Many real-world data can be naturally represented as graphs such as social networks, chemical molecules, transportation networks, and recommender systems [4, 156, 196]. As a generalization of deep neural networks for graph-structured data, graph neural networks (GNNs) have achieved great success in capturing the abundant information residing in graphs and tackle various graph-related applications [156, 196].

However, the prevalence of large-scale graphs in real-world scenarios, often on the scale of millions of nodes and edges, poses significant computational challenges for training GNNs. More dramatically, the computational cost continues to increase when we need to retrain the models multiple times, e.g., under incremental learning settings, hyperparameter and neural architecture search. To address this challenge, a natural idea is to properly simplify, or reduce the graph so that we can not only speed up graph algorithms (including GNNs) but also facilitate storage, visualization

and retrieval for associated graph data analysis tasks.

There are two main strategies to simplify graphs: graph sparsification [115, 129] and graph coarsening [102, 101] . Graph sparsification approximates a graph with a sparse graph by reducing the number of edges, while graph coarsening directly reduces the number of nodes by replacing the original node set with its subset. However, these methods have some shortcomings: (1) sparsification becomes much less promising in simplifying graphs when nodes are also associated with attributes as sparsification does not reduce the node attributes; (2) the goal of sparsification and coarsening is to preserve some graph properties such as principle eigenvalues [102] that could be not optimal for the downstream performance of GNNs. In this work, we ask if it is possible to significantly reduce the graph size while providing sufficient information to well train GNN models.

Motivated by dataset distillation [144] and dataset condensation [189] which generate a small set of images to train deep neural networks on the downstream task, we aim to condense a given graph through learning a synthetic graph structure and node attributes. Correspondingly, we propose the task of *graph condensation*[1]. It aims to minimize the performance gap between GNN models trained on a synthetic, simplified graph and the original training graph. In this work, we focus on attributed graphs and the node classification task. We show that we are able to reduce the number of graph nodes to as low as 0.1% while training various GNN architectures to reach surprisingly good performance on the synthetic graph. For example, in Figure 4.2, we condense the graph of the Reddit dataset with 153,932 training nodes into only 154 synthetic nodes together with their connections. In essence, we face two challenges for graph condensation: (1) how to formulate the objective for graph condensation tractable for learning; and (2) how to parameterize the to-be-learned node features and graph structure. To address the above challenges, we adapt the gradient matching scheme in [189] and match the gradients of GNN parameters w.r.t. the condensed graph and original graph. In this way, the GNN trained on condensed graph can mimic the training trajectory of that on real data. Further, we carefully design the strategy for parametrizations for the condensed graph. In particular, we introduce the strategy of parameterizing the condensed features as free parameters and

[1]We aim to condense both graph structure and node attributes. A formal definition is given in Section 3.

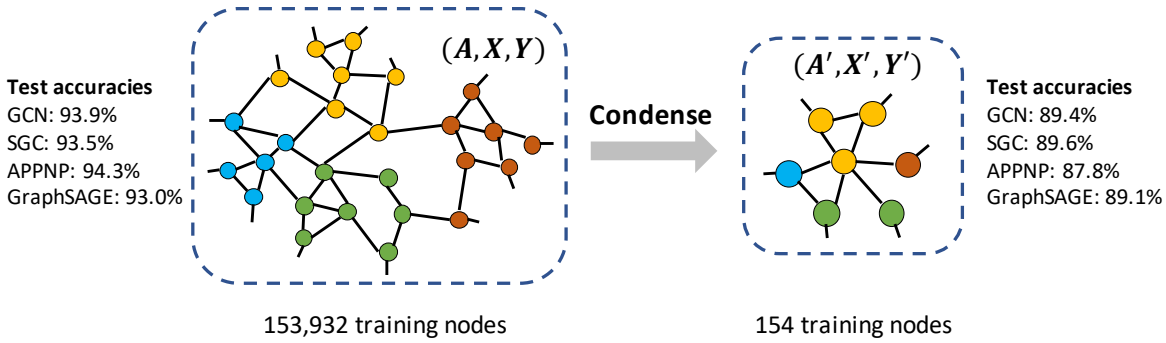


Figure 2.1: We study the graph condensation problem, which seeks to learn a small, synthetic graph, features and labels $\{\mathbf{A}', \mathbf{X}', \mathbf{Y}'\}$ from a large, original dataset $\{\mathbf{A}, \mathbf{X}, \mathbf{Y}\}$, which can be used to train GNN models that generalize comparably to the original. **Shown:** An illustration of our proposed GCond graph condensation approach's empirical performance, which exhibits *95.3%* of original graph test performance with *99.9%* data reduction.

model the synthetic graph structure as a function of features, which takes advantage of the implicit relationship between structure and node features, consumes less number of parameters and offers better performance. Our contributions can be summarized as follows:

1. We make the first attempt to condense a large-real graph into a small-synthetic graph, such that the GNN models trained on the large graph and small graph have comparable performance. We introduce a proposed framework for graph condensation (GCond) which parameterizes the condensed graph structure as a function of condensed node features, and leverages a gradient matching loss as the condensation objective.
2. Through extensive experimentation, we show that GCond is able to condense different graph datasets and achieve comparable performance to their larger counterparts. For instance, GCond approximates the original test accuracy by 95.3% on Reddit, 99.8% on Flickr and 99.0% on Citeseer, while reducing their graph size by more than 99.9%. Our approach consistently outperforms coarsening, coreset and dataset condensation baselines.
3. We show that the condensed graphs can generalize well to different GNN test models. Additionally, we observed reliable correlation of performances between condensed dataset training and whole-dataset training in the neural architecture search (NAS) experiments.

2.2 Related Work

Dataset Distillation & Condensation. Dataset distillation (DD) [144, 7, 110, 143, 15] aims to distill knowledge of a large training dataset into a small synthetic dataset, such that a model trained on the synthetic set is able to obtain the comparable performance to that of a model trained on the original dataset. To improve the efficiency of DD, dataset condensation (DC) [189, 187] is proposed to learn the small synthetic dataset by matching the gradients of the network parameters w.r.t. large-real and small-synthetic training data. However, these methods are designed exclusively for image data and are not applicable to non-Euclidean graph-structured data where samples (nodes) are interdependent. In this work, we generalize the problem of dataset condensation to graph domain and we seek to jointly learn the synthetic node features as well as graph structure. Additionally, our work relates to coreset methods [148, 125, 118], which seek to find informative samples from the original datasets. However, they rely on the presence of representative samples, and tend to give suboptimal performance.

Graph Sparsification & Coarsening. Graph sparsification and coarsening are two means of reducing the size of a graph. Sparsification reduces the number of edges while approximating pairwise distances [115], cuts [74] or eigenvalues [129] while coarsening reduces the number of nodes with similar constraints [102, 101, 26], typically by grouping original nodes into super-nodes, and defining their connections. [12] proposes a GNN-based framework to learn these connections to improve coarsening quality. [62] adopts coarsening as a preprocessing method to help scale up GNNs. Graph condensation also aims to reduce the number of nodes, but aims to learn synthetic nodes and connections in a supervised way, rather than unsupervised grouping as in these prior works. Graph pooling is also related to our work, but it targets at improving graph-level representation learning.

Graph Neural Networks. Graph neural networks (GNNs) are a modern way to capture the intuition that inferences for individual samples (nodes) can be enhanced by utilizing graph-based information from neighboring nodes [78, 57, 79, 139, 156, 151, 93, 97, 171, 197, 190]. Due to their prevalence, various real-world applications have been tremendously facilitated including

recommender systems [168, 39], computer vision [82] and drug discovery [33].

Graph Structure Learning. Our work is also related to graph structure learning, which explores methods to learn graphs from data. One line of work [31, 35] learns graphs under certain structural constraints (e.g. sparsity) based on graph signal processing. Recent efforts aim to learn graphs by leveraging GNNs [45, 70, 18]. However, these methods are incapable of learning graphs with smaller size, and are thus not applicable for graph condensation. For more related works on graph structure learning and graph generation, we kindly refer the reader to recent surveys [28, 206].

2.3 Methodology

In this section, we present our proposed *graph condensation* framework, GCond. Consider that we have a graph dataset $\mathcal{T} = \{\mathbf{A}, \mathbf{X}, \mathbf{Y}\}$, where $\mathbf{A} \in \mathbb{R}^{N\times N}$ is the adjacency matrix, N is the number of nodes, $\mathbf{X} \in \mathbb{R}^{N\times d}$ is the d-dimensional node feature matrix and $\mathbf{Y} \in \{0, \ldots, C-1\}^N$ denotes the node labels over C classes. Graph condensation aims to learn a small, synthetic graph dataset $\mathcal{S} = \{\mathbf{A}', \mathbf{X}', \mathbf{Y}'\}$ with $\mathbf{A}' \in \mathbb{R}^{N'\times N'}$, $\mathbf{X}' \in \mathbb{R}^{N'\times D}$, $\mathbf{Y}' \in \{0, \ldots, C-1\}^{N'}$ and $N' \ll N$, such that a GNN trained on $\mathcal{S}$ can achieve comparable performance to one trained on the much larger $\mathcal{T}$. Thus, the objective can be formulated as the following bi-level problem,

$$\min_{\mathcal{S}} \mathcal{L}\left(\mathrm{GNN}_{\boldsymbol{\theta}_\mathcal{S}}(\mathbf{A}, \mathbf{X}), \mathbf{Y}\right) \quad \text{s.t} \quad \boldsymbol{\theta}_\mathcal{S} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathrm{GNN}_{\boldsymbol{\theta}}(\mathbf{A}', \mathbf{X}'), \mathbf{Y}'), \tag{2.1}$$

where $\mathrm{GNN}_{\boldsymbol{\theta}}$ denotes the GNN model parameterized with $\boldsymbol{\theta}$, $\boldsymbol{\theta}_\mathcal{S}$ denotes the parameters of the model trained on $\mathcal{S}$, and $\mathcal{L}$ denotes the loss function used to measure the difference between model predictions and ground truth, i.e. cross-entropy loss. However, optimizing the above objective can lead to overfitting on a specific model initialization. To generate condensed data that generalizes to a distribution of random initializations $P_{\boldsymbol{\theta}_0}$, we rewrite the objective as follows:

$$\min_{\mathcal{S}} \mathrm{E}_{\boldsymbol{\theta}_0\sim P_{\boldsymbol{\theta}_0}}\left[\mathcal{L}\left(\mathrm{GNN}_{\boldsymbol{\theta}_\mathcal{S}}(\mathbf{A}, \mathbf{X}), \mathbf{Y}\right)\right] \quad \text{s.t.} \quad \boldsymbol{\theta}_\mathcal{S} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathrm{GNN}_{\boldsymbol{\theta}(\boldsymbol{\theta}_0)}(\mathbf{A}', \mathbf{X}'), \mathbf{Y}'). \tag{2.2}$$

where $\boldsymbol{\theta}(\boldsymbol{\theta}_0)$ indicates that $\boldsymbol{\theta}$ is a function acting on $\boldsymbol{\theta}_0$. Note that the setting discussed above is for inductive learning where all the nodes are labeled and test nodes are unseen during training. We can easily generalize graph condensation to transductive setting by assuming $\mathbf{Y}$ is partially labeled.

2.3.1 Graph Condensation via Gradient Matching

To tackle the optimization problem in Eq. (2.2), one strategy is to compute the gradient of $\mathcal{L}$ w.r.t $\mathcal{S}$ and optimize $\mathcal{S}$ via gradient descent, as in dataset distillation [144]. However, this requires solving a nested loop optimization and unrolling the whole training trajectory of the inner problem, which can be prohibitively expensive. To bypass the bi-level optimization, we follow the gradient matching method proposed in [189] which aims to match the network parameters w.r.t. large-real and small-synthetic training data by matching their gradients at each training step. In this way, the training trajectory on small-synthetic data $\mathcal{S}$ can mimic that on the large-real data $\mathcal{T}$, i.e., the models trained on these two datasets converge to similar solutions (parameters). Concretely, the parameter matching process for GNNs can be modeled as follows:

$$\begin{gathered} \min_{\mathcal{S}} \mathrm{E}_{\boldsymbol{\theta}_0 \sim P_{\theta_0}} \left[\sum_{t=0}^{T-1} D\left(\boldsymbol{\theta}_t^{\mathcal{S}}, \boldsymbol{\theta}_t^{\mathcal{T}}\right) \right] \quad \text{with} \\ \boldsymbol{\theta}_{t+1}^{\mathcal{S}} = \mathrm{opt}_{\boldsymbol{\theta}}\left(\mathcal{L}\left(\mathrm{GNN}_{\boldsymbol{\theta}_t^{\mathcal{S}}}(\mathbf{A}', \mathbf{X}'), \mathbf{Y}'\right)\right) \text{ and } \boldsymbol{\theta}_{t+1}^{\mathcal{T}} = \mathrm{opt}_{\boldsymbol{\theta}}\left(\mathcal{L}\left(\mathrm{GNN}_{\boldsymbol{\theta}_t^{\mathcal{T}}}(\mathbf{A}, \mathbf{X}), \mathbf{Y}\right)\right) \end{gathered} \tag{2.3}$$

where $D(\cdot, \cdot)$ is a distance function, T is the number of steps of the whole training trajectory, $\mathrm{opt}_{\boldsymbol{\theta}}$ is the update rule for model parameters, and $\boldsymbol{\theta}_t^{\mathcal{S}}$, $\boldsymbol{\theta}_t^{\mathcal{T}}$ denote the model parameters trained on $\mathcal{S}$ and $\mathcal{T}$ at time step t, respectively. Since our goal is to match the parameters step by step, we then consider one-step gradient descent for the update rule $\mathrm{opt}_{\boldsymbol{\theta}}$:

$$\boldsymbol{\theta}_{t+1}^{\mathcal{S}} \leftarrow \boldsymbol{\theta}_t^{\mathcal{S}} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\mathrm{GNN}_{\boldsymbol{\theta}_t^{\mathcal{S}}}(\mathbf{A}', \mathbf{X}'), \mathbf{Y}'\right) \quad \text{and} \quad \boldsymbol{\theta}_{t+1}^{\mathcal{T}} \leftarrow \boldsymbol{\theta}_t^{\mathcal{T}} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\mathrm{GNN}_{\boldsymbol{\theta}_t^{\mathcal{T}}}(\mathbf{A}, \mathbf{X}), \mathbf{Y}\right) \tag{2.4}$$

where η is the learning rate for the gradient descent. Based on the observation made in [189] that the distance between $\boldsymbol{\theta}_t^{\mathcal{S}}$ and $\boldsymbol{\theta}_t^{\mathcal{T}}$ is typically small, we can simplify the objective as a gradient matching process as follows,

$$\min_{\mathcal{S}} \mathrm{E}_{\boldsymbol{\theta}_0 \sim P_{\theta_0}} \left[\sum_{t=0}^{T-1} D\left(\nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\mathrm{GNN}_{\boldsymbol{\theta}_t}(\mathbf{A}', \mathbf{X}'), \mathbf{Y}'\right), \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\mathrm{GNN}_{\boldsymbol{\theta}_t}(\mathbf{A}, \mathbf{X}), \mathbf{Y}\right)\right) \right] \tag{2.5}$$

where $\boldsymbol{\theta}_t^{\mathcal{S}}$ and $\boldsymbol{\theta}_t^{\mathcal{T}}$ are replaced by $\boldsymbol{\theta}_t$, which is trained on the small-synthetic graph. The distance D is further defined as the sum of the distance dis at each layer. Given two gradients $\mathbf{G}^{\mathcal{S}} \in \mathbb{R}^{d_1 \times d_2}$ and $\mathbf{G}^{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2}$ at a specific layer, the distance $dis(\cdot, \cdot)$ used for condensation is defined as follows,

$$dis(\mathbf{G}^{\mathcal{S}}, \mathbf{G}^{\mathcal{T}}) = \sum_{i=1}^{d_2} \left(1 - \frac{\mathbf{G}_{\mathbf{i}}^{\mathcal{S}} \cdot \mathbf{G}_{\mathbf{i}}^{\mathcal{T}}}{\left\|\mathbf{G}_{\mathbf{i}}^{\mathcal{S}}\right\| \left\|\mathbf{G}_{\mathbf{i}}^{\mathcal{T}}\right\|}\right) \tag{2.6}$$

where $\mathbf{G}_{\mathbf{i}}^{\mathcal{S}}, \mathbf{G}_{\mathbf{i}}^{\mathcal{T}}$ are the i-th column vectors of the gradient matrices. With the above formulations, we are able to achieve parameter matching through an efficient strategy of gradient matching.

We note that jointly learning the three variables $\mathbf{A}', \mathbf{X}'$ and $\mathbf{Y}'$ is highly challenging, as they are interdependent. Hence, to simplify the problem, we fix the node labels $\mathbf{Y}'$ while keeping the class distribution the same as the original labels $\mathbf{Y}$.

Graph Sampling. GNNs are often trained in a full-batch manner [78, 156]. However, as suggested by previous works that reconstruct data from gradients [203], large batch size tends to make reconstruction more difficult because more variables are involved during optimization. To make things worse, the computation cost of GNNs gets expensive on large graphs as the forward pass of GNNs involves the aggregation of enormous neighboring nodes. To address the above issues, we sample a fixed-size set of neighbors on the original graph in each aggregation layer of GNNs and adopt a mini-batch training strategy. To further reduce memory usage and ease optimization, we calculate the gradient matching loss for nodes from different classes separately, as matching the gradients w.r.t. the data from a single class is easier than that from all classes. Specifically, for a given class c, we sample a batch of nodes of class c together with a portion of their neighbors from large-real data $\mathcal{T}$. We denote the process as $(\mathbf{A}_c, \mathbf{X}_c, \mathbf{Y}_c) \sim \mathcal{T}$. For the condensed graph $\mathbf{A}'$, we sample a batch of synthetic nodes of class c but do not sample their neighbors. In other words, we use all of their neighbors, i.e., all other nodes, during the aggregation process, since we need to learn the connections with other nodes. We denote the process as $(\mathbf{A}'_c, \mathbf{X}'_c, \mathbf{Y}'_c) \sim \mathcal{S}$.

2.3.2 Modeling Condensed Graph Data

One essential challenge in the graph condensation problem is how to model the condensed graph data and resolve dependency among nodes. The most straightforward way is to treat both $\mathbf{A}'$ and $\mathbf{X}'$ as free parameters. However, the number of parameters in $\mathbf{A}'$ grows quadratically as N' increases. The increased model complexity can pose challenges in optimizing the framework and increase the risk of overfitting. Therefore, it is desired to parametrize the condensed adjacency matrix in a way where the number of parameters does not grow too fast. On the other hand, treating $\mathbf{A}'$ and $\mathbf{X}'$ as independent parameters overlooks the implicit correlations between graph structure and features,

which have been widely acknowledged in the literature [63, 126]; e.g., in social networks, users interact with others based on their interests, while in e-commerce, users purchase products due to certain product attributes. Hence, we propose to model the condensed graph structure as a function of the condensed node features:

$$\mathbf{A}' = g_\Phi(\mathbf{X}'), \qquad \text{with } \mathbf{A}'_{ij} = \text{Sigmoid}\left(\frac{\text{MLP}_\Phi([\mathbf{x}'_i; \mathbf{x}'_j]) + \text{MLP}_\Phi([\mathbf{x}'_j; \mathbf{x}'_i])}{2}\right) \tag{2.7}$$

where MLP_Φ is a multi-layer neural network parameterized with Φ and $[\cdot;\cdot]$ denotes concatenation. In Eq. (2.7), we intentionally control $\mathbf{A}'_{ij} = \mathbf{A}'_{ji}$ to make the condensed graph structure symmetric since we are mostly dealing with symmetric graphs. It can also adjust to asymmetric graphs by setting $\mathbf{A}'_{ij} = \text{Sigmoid}(\text{MLP}_\Phi([\mathbf{x}_i; \mathbf{x}'_j])$. Then we rewrite our objective as

$$\min_{\mathbf{X}',\Phi} \text{E}_{\theta_0 \sim P_{\theta_0}} \left[\sum_{t=0}^{T-1} D\left(\nabla_\theta \mathcal{L}\left(\text{GNN}_{\theta_t}(g_\Phi(\mathbf{X}'), \mathbf{X}'), \mathbf{Y}'\right), \nabla_\theta \mathcal{L}\left(\text{GNN}_{\theta_t}(\mathbf{A}, \mathbf{X}), \mathbf{Y}\right)\right) \right] \tag{2.8}$$

Note that there are two clear benefits of the above formulation over the naïve one (free parameters). Firstly, the number of parameters for modeling graph structure no longer depends on the number of nodes, hence avoiding jointly learning $O(N'^2)$ parameters; as a result, when N' gets larger, GCond suffers less risk of overfitting. Secondly, if we want to grow the synthetic graph by adding more synthetic nodes condensed from real graph, the trained MLP_Φ can be employed to infer the connections of new synthetic nodes, and hence we only need to learn their features.

Alternating Optimization Schema. Jointly optimizing $\mathbf{X}'$ and Φ is often challenging as they are directly affecting each other. Instead, we propose to alternatively optimize $\mathbf{X}'$ and Φ: we update Φ for the first τ_1 epochs and then update $\mathbf{X}'$ for τ_2 epochs; the process is repeated until the stopping condition is met – we find empirically that this does better.

Sparsification. In the learned condensed adjacency matrix $\mathbf{A}'$, there can exist some small values which have little effect on the aggregation process in GNNs but still take up a certain amount of storage (e.g. 4 bytes per float). Thus, we remove the entries whose values are smaller than a given threshold δ to promote sparsity of the learned $\mathbf{A}'$.

The detailed algorithm can be found in Algorithm 1 in Section C.2. In detail, we first set the condensed label set $\mathbf{Y}'$ to fixed values and initialize $\mathbf{X}'$ as node features randomly selected from each

class. In each outer loop, we sample a GNN model initialization $\boldsymbol{\theta}$ from a distribution $P_{\boldsymbol{\theta}}$. Then, for each class we sample the corresponding node batches from $\mathcal{T}$ and $\mathcal{S}$, and calculate the gradient matching loss within each class. The sum of losses from different classes are used to update $\mathbf{X}'$ or Φ. After that we update the GNN parameters for $\tau_{\boldsymbol{\theta}}$ epochs. When finishing the updating of condensed graph parameters, we filter edge weights smaller than δ to obtain the final sparsified graph structure.

A "Graphless" Model Variant. We now explore another parameterization for the condensed graph data. We provide a model variant named GCond-X that only learns the condensed node features $\mathbf{X}'$ without learning the condensed structure $\mathbf{A}'$. In other words, we use a fixed identity matrix $\mathbf{I}$ as the condensed graph structure. Specifically, this model variant aims to match the gradients of GNN parameters on the large-real data $(\mathbf{A}, \mathbf{X})$ and small-synthetic data $(\mathbf{I}, \mathbf{X}')$. Although GCond-X is unable to learn the condensed graph structure which can be highly useful for downstream data analysis, it still shows competitive performance in Table 3.1 in the experiments because the features are learned to incorporate relevant information from the graph via the matching loss.

2.4 Experiment

In this section, we design experiments to validate the effectiveness of the proposed framework GCond. We first introduce experimental settings, then compare GCond against representative baselines with discussions and finally show some advantages of GCond.

2.4.1 Experimental setup

Datasets. We evaluate the condensation performance of the proposed framework on three transductive datasets, i.e., Cora, Citeseer [78] and Ogbn-arxiv [60], and two inductive datasets, i.e., Flickr [177] and Reddit [57]. We use the public splits for all the datasets. For the inductive setting, we follow the setup in [57] where the test graph is not available during training.

Baselines. We compare our proposed methods to five baselines: (i) one *graph coarsening* method [101, 62], (ii-iv) three coreset methods (*Random*, *Herding* [148] and *K-Center* [40, 125]), and (v) *dataset condensation* (DC). For the graph coarsening method, we adopt the variation neighborhoods method implemented by [62]. For coreset methods, we first use them to select nodes

from the original dataset and induce a subgraph from the selected nodes to serve as the reduced graph. In Random, the nodes are randomly selected. The Herding method, which is often used in continual learning [118, 14], picks samples that are closest to the cluster center. K-Center selects the center samples to minimize the largest distance between a sample and its nearest center. We use the implementations provided by [189] for Herding, K-Center and DC. As vanilla DC cannot leverage any structure information, we develop a variant named DC-Graph, which additionally leverages graph structure during test stage, to replace DC for the following experiments. A comparison between DC, DC-Graph, GCond and GCond-X is shown in Table 2.1.

Evaluation. We first use the aforementioned baselines to obtain condensed graphs and then evaluate them on GNNs for both transductive and inductive node classification tasks. For transductive datasets, we condense the full graph with N nodes into a synthetic graph with rN $(0 < r < 1)$ nodes, where r is the ratio of synthetic nodes to original nodes. For inductive datasets, we only condense the training graph since the rest of the full graph is not available during training. The choices of r are listed in Table 3.1. For each r, we generate 5 condensed graphs with different seeds. To evaluate the effectiveness of condensed graphs, we have two stages: (1) a training stage, where we train a GNN model on the condensed graph, and (2) a test stage, where the trained GNN uses the test graph (or full graph in transductive setting) to infer the labels for test nodes. The resulting test performance is compared with that obtained when training on original datasets. All experiments are repeated 10 times, and we report average performance and variance.

Hyperparameter Settings. As our goal is to generate highly informative synthetic graphs which can benefit GNNs, we choose one representative model, GCN [78], for performance evaluation. For the GNN used in condensation, i.e., the $\text{GNN}_\theta(\cdot)$ in Eq. (2.8), we adopt SGC [151] which decouples the propagation and transformation process but still shares similar graph filtering behavior as GCN. Unless otherwise stated, we use 2-layer models with 256 hidden units. The weight decay and dropout for the models are set to 0 in condensation process.

Table 2.1: Information comparison used during condensation, training and test for reduction methods. $\mathbf{A}', \mathbf{X}'$ and $\mathbf{A}, \mathbf{X}$ are condensed (original) graph and features, respectively.

	DC	DC-Graph	GCond-X	GCond
Condensation	$\mathbf{X}_{\text{train}}$	$\mathbf{X}_{\text{train}}$	$\mathbf{A}_{\text{train}}, \mathbf{X}_{\text{train}}$	$\mathbf{A}_{\text{train}}, \mathbf{X}_{\text{train}}$
Training	$\mathbf{X}'$	$\mathbf{X}'$	$\mathbf{X}'$	$\mathbf{A}', \mathbf{X}'$
Test	$\mathbf{X}_{\text{test}}$	$\mathbf{A}_{\text{test}}, \mathbf{X}_{\text{test}}$	$\mathbf{A}_{\text{test}}, \mathbf{X}_{\text{test}}$	$\mathbf{A}_{\text{test}}, \mathbf{X}_{\text{test}}$

Table 2.2: GCond and GCond-X achieve promising performance in comparison to baselines even with extremely large reduction rates. We report transductive performance on Citeseer, Cora, Ogbn-arxiv; inductive performance on Flickr, Reddit. Performance is reported as test accuracy (%).

		Baselines					Proposed		
Dataset	Ratio (r)	Random ($\mathbf{A}', \mathbf{X}'$)	Herding $\mathbf{A}', \mathbf{X}'$	K-Center $\mathbf{A}', \mathbf{X}'$	Coarsening $\mathbf{A}', \mathbf{X}'$	DC-Graph ($\mathbf{X}'$)	GCond-X ($\mathbf{X}'$)	GCond $\mathbf{A}', \mathbf{X}'$	Whole Dataset
Citeseer	0.9%	54.4±4.4	57.1±1.5	52.4±2.8	52.2±0.4	66.8±1.5	**71.4**±0.8	70.5±1.2	71.7±0.1
	1.8%	64.2±1.7	66.7±1.0	64.3±1.0	59.0±0.5	66.9±0.9	69.8±1.1	**70.6**±0.9	
	3.6%	69.1±0.1	69.0±0.1	69.1±0.1	65.3±0.5	66.3±1.5	69.4±1.4	**69.8**±1.4	
Cora	1.3%	63.6±3.7	67.0±1.3	64.0±2.3	31.2±0.2	67.3±1.9	75.9±1.2	**79.8**±1.3	81.2±0.2
	2.6%	72.8±1.1	73.4±1.0	73.2±1.2	65.2±0.6	67.6±3.5	75.7±0.9	**80.1**±0.6	
	5.2%	76.8±0.1	76.8±0.1	76.7±0.1	70.6±0.1	67.7±2.2	76.0±0.9	**79.3**±0.3	
Ogbn-arxiv	0.05%	47.1±3.9	52.4±1.8	47.2±3.0	35.4±0.3	58.6±0.4	**61.3**±0.5	59.2±1.1	71.4±0.1
	0.25%	57.3±1.1	58.6±1.2	56.8±0.8	43.5±0.2	59.9±0.3	**64.2**±0.4	63.2±0.3	
	0.5%	60.0±0.9	60.4±0.8	60.3±0.4	50.4±0.1	59.5±0.3	63.1±0.5	**64.0**±0.4	
Flickr	0.1%	41.8±2.0	42.5±1.8	42.0±0.7	41.9±0.2	46.3±0.2	45.9±0.1	**46.5**±0.4	47.2±0.1
	0.5%	44.0±0.4	43.9±0.9	43.2±0.1	44.5±0.1	45.9±0.1	45.0±0.2	**47.1**±0.1	
	1%	44.6±0.2	44.4±0.6	44.1±0.4	44.6±0.1	45.8±0.1	45.0±0.1	**47.1**±0.1	
Reddit	0.05%	46.1±4.4	53.1±2.5	46.6±2.3	40.9±0.5	88.2±0.2	**8**8.4±0.4	88.0±1.8	93.9±0.0
	0.1%	58.0±2.2	62.7±1.0	53.0±3.3	42.8±0.8	89.5±0.1	89.3±0.1	**89.6**±0.7	
	0.2%	66.3±1.9	71.0±1.6	58.5±2.1	47.4±0.9	**9**0.5±1.2	88.8±0.4	90.1±0.5	

2.4.2 Comparison with Baselines

In this subsection, we test the performance of a 2-layer GCN on the condensed graphs, and compare the proposed GCond and GCond-X with baselines. Notably, all methods produce both structure and node features, i.e. $\mathbf{A}'$ and $\mathbf{X}'$, except DC-Graph and GCond-X. Since DC-Graph and GCond-X do not produce any structure, we simply use an identity matrix as the adjacency matrix when training GNNs solely on condensed features. However, during inference, we use the full graph (transductive setting) or test graph (inductive setting) to propagate information based on the trained GNNs. This training paradigm is similar to the C&S model [61] which trains an MLP without the graph information and performs label propagation based on MLP predictions. Table 3.1 reports node classification performance; we make the following observations:

Obs 1. Condensation methods achieve promising performance even with extremely large reduction rates. Condensation methods, i.e., GCond, GCond-X and DC-Graph, outperform coreset methods and graph coarsening significantly at the lowest ratio r for each dataset. This shows the importance of learning synthetic data using the guidance from downstream tasks. Notably, GCond achieves 79.8%, 80.1% and 79.3% at 1.3%, 2.6% and 5.2% condensation ratios at Cora, while the whole dataset performance is 81.2%. The GCond variants also show promising performance on Cora, Flickr and Reddit at all coarsening ratios. Although the gap between whole-dataset Ogbn-arxiv and our methods is larger, they still outperform baselines by a large margin.

Obs 2. Learning $\mathbf{X}'$ instead of $(\mathbf{A}', \mathbf{X}')$ as the condensed graph can also lead to good results. GCond-X achieves close performance to GCond on 11 of 15 cases. Since our objective in graph condensation is to achieve parameter matching through gradient matching, training a GNN on the learned features $\mathbf{X}'$ with identity adjacency matrix is also able to mimic the training trajectory of GNN parameters. One reason could be that $\mathbf{X}'$ has already encoded node features and structural information of the original graph during the condensation process. However, there are many scenarios where the graph structure is essential such as the generalization to other GNN architectures (e.g., GAT) and visualizing the patterns in the data. More details are given in the following subsections.

Obs 3. Condensing node features and structural information simultaneously can lead to better performance. In most cases, GCond and GCond-X obtain much better performance than DC-Graph. One key reason is that GCond and GCond-X can take advantage of both node features and structural information in the condensation process. We notice that DC-Graph achieves a highly comparable result (90.5%) on Reddit at 0.2% condensation ratio to the whole dataset performance (93.9%). This may indicate that the original training graph structure might not be useful. To verify this assumption, we train a GCN on the original Reddit dataset without using graph structure (i.e., setting $\mathbf{A}_{\text{train}} = \mathbf{I}$), but allow using the test graph structure for inference using the trained model. The obtained performance is 92.5%, which is very close to the original performance 93.9%, indicating that training without graph structure can still achieve comparable performance. We also note that

learning $\mathbf{X}'$, A' simultaneously creates opportunities to absorb information from graph structure directly into learned features, lessening reliance on distilling graph properties reliably while still achieving good generalization performance from features.

Obs 4. Larger condensed graph size does not strictly indicate better performance. Although larger condensed graph sizes allow for more parameters which can potentially encapsulate more information from original graph, it simultaneously becomes harder to optimize due to the increased model complexity. We observe that once the condensation ratio reaches a certain threshold, the performance becomes stable. However, the performance of coreset methods and graph coarsening is much more sensitive to the reduction ratio. Coreset methods only select existing samples while graph coarsening groups existing nodes into super nodes. When the reduction ratio is too low, it becomes extremely difficult to select informative nodes or form representative super nodes by grouping.

2.4.3 Generalizability of Condensed Graphs

Next, we illustrate the generalizability of condensed graphs from the following three perspectives.

Different Architectures. Next, we show the generalizability of the graph condensation procedure. Specifically, we show test performance when using a graph condensed by one GNN model to train different GNN architectures. Specifically, we choose APPNP [79], GCN, SGC [151], GraphSAGE [57], Cheby [23] and GAT [139]. We also include MLP and report the results in Table 2.3. From the table, we find that the condensed graphs generated by GCond show good generalization on different architectures. We may attribute such transferability across different architectures to similar filtering behaviors of those GNN models, which have been studied in [104, 204].

Versatility of GCond. The proposed GCond is highly composable in that we can adopt various GNNs inside the condensation network. We investigate the performances of various GNNs when using different GNN models in the condensation process, i.e., $\text{GNN}_\theta(\cdot)$ in Eq. (2.8). We choose APPNP, Cheby, GCN, GraphSAGE and SGC to serve as the models used in condensation and evaluation. Note that we omit GAT due to its deterioration under large neighborhood sizes [103]. We choose Cora and Ogbn-arxiv to report the performance in Table 5.4 where C and T denote

Table 2.3: Graph condensation can work well with different architectures. Avg. stands for the average test accuracy of APPNP, Cheby, GCN, GraphSAGE and SGC. SAGE stands for GraphSAGE.

	Methods	Data	MLP	GAT	APPNP	Cheby	GCN	SAGE	SGC	Avg.
Citeseer $r = 1.8\%$	DC-Graph	$\mathbf{X}'$	66.2	-	66.4	64.9	66.2	65.9	69.6	66.6
	GCond-X	$\mathbf{X}'$	69.6	-	69.7	70.6	69.7	69.2	71.6	70.2
	GCond	$\mathbf{A}', X'$	63.9	55.4	69.6	68.3	70.5	66.2	70.3	69.0
Cora $r = 2.6\%$	DC-Graph	$\mathbf{X}'$	67.2	-	67.1	67.7	67.9	66.2	72.8	68.3
	GCond-X	$\mathbf{X}'$	76.0	-	77.0	74.1	75.3	76.0	76.1	75.7
	GCond	$\mathbf{A}', X'$	73.1	66.2	78.5	76.0	80.1	78.2	79.3	78.4
Ogbn-arxiv $r = 0.25\%$	DC-Graph	$\mathbf{X}'$	59.9	-	60.0	55.7	59.8	60.0	60.4	59.2
	GCond-X	$\mathbf{X}'$	64.1	-	61.5	59.5	64.2	64.4	64.7	62.9
	GCond	$\mathbf{A}', X'$	62.2	60.0	63.4	54.9	63.2	62.6	63.7	61.6
Flickr $r = 0.5\%$	DC-Graph	$\mathbf{X}'$	43.1	-	45.7	43.8	45.9	45.8	45.6	45.4
	GCond-X	$\mathbf{X}'$	42.1	-	44.6	42.3	45.0	44.7	44.4	44.2
	GCond	$\mathbf{A}', X'$	44.8	40.1	45.9	42.8	47.1	46.2	46.1	45.6
Reddit $r = 0.1\%$	DC-Graph	$\mathbf{X}'$	50.3	-	81.2	77.5	89.5	89.7	90.5	85.7
	GCond-X	$\mathbf{X}'$	40.1	-	78.7	74.0	89.3	89.3	91.0	84.5
	GCond	$\mathbf{A}', X'$	42.5	60.2	87.8	75.5	89.4	89.1	89.6	86.3

Table 2.4: Cross-architecture performance is shown in test accuracy (%). SAGE: GraphSAGE. Graphs condensed by different GNNs all show strong transfer performance on other architectures.

(a) Cora, r=2.6%

C\T	APPNP	Cheby	GCN	SAGE	SGC
APPNP	72.1±2.6	60.8±6.4	**73.5**±2.4	72.3±3.5	73.1±3.1
Cheby	75.3±2.9	71.8±1.1	**76.8**±2.1	76.4±2.0	75.5±3.5
GCN	69.8±4.0	53.2±3.4	**70.6**±3.7	60.2±1.9	68.7±5.4
SAGE	77.1±1.1	69.3±1.7	77.0±0.7	76.1±0.7	**77.7**±1.8
SGC	78.5±1.0	76.0±1.1	**80.1**±0.6	78.2±0.9	79.3±0.7

(b) Ogbn-arxiv, r=0.05%

C\T	APPNP	Cheby	GCN	SAGE	SGC
APPNP	60.3±0.2	51.8±0.5	59.9±0.4	59.0±1.1	**61.2**±0.4
Cheby	57.4±0.4	53.5±0.5	57.4±0.8	57.1±0.8	**58.2**±0.6
GCN	59.3±0.4	51.8±0.7	**60.3**±0.3	60.2±0.4	59.2±0.7
SAGE	57.6±0.8	53.9±0.6	58.1±0.6	57.8±0.7	**59.0**±1.1
SGC	59.7±0.5	49.5±0.8	59.2±1.1	58.9±1.6	**60.5**±0.6

condensation and test models, respectively. The graphs condensed by different GNNs all show strong transfer performance on other architectures.

Neural Architecture Search. We also perform experiments on neural architecture search. We search 480 architectures of APPNP and perform the search process on Cora, Citeseer and Ogbn-arxiv. Specifically, we train each architecture on the reduced graph for epochs on as the model converges faster on the smaller graph. We observe reliable correlation of performances between condensed dataset training and whole-dataset training as shown in Table A.4: 0.76/0.79/0.64 for Cora/Citeseer/Ogbn-arxiv.

2.4.4 Analysis on Condensed Data

Statistics of Condensed Graphs. In Table 2.5, we compare several properties between condensed graphs and original graphs. Note that a widely used homophily measure is defined

Table 2.5: Comparison between condensed graphs and original graphs. The condensed graphs have fewer nodes and are more dense.

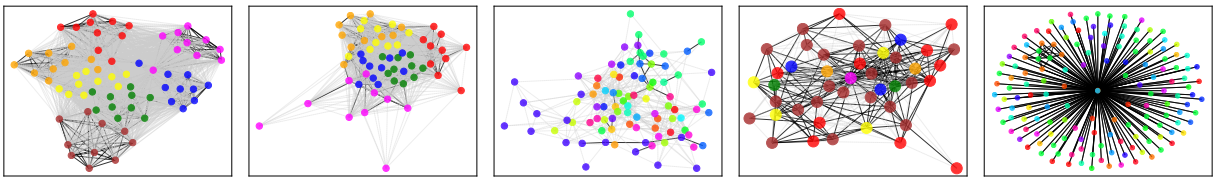
	Citeseer, r=0.9%		Cora, r=1.3%		Ogbn-arxiv, r=0.25%		Flickr, r=0.5%		Reddit, r=0.1%	
	Whole	GCond	Whole	GCond	Whole	GCond	Whole	GCond	Whole	GCond
Accuracy	70.7	70.5	81.5	79.8	71.4	63.2	47.1	47.1	94.1	89.4
#Nodes	3,327	60	2,708	70	169,343	454	44,625	223	153,932	153
#Edges	4,732	1,454	5,429	2,128	1,166,243	3,354	218,140	3,788	10,753,238	301
Sparsity	0.09%	80.78%	0.15%	86.86%	0.01%	3.25%	0.02%	15.23%	0.09%	2.57%
Homophily	0.74	0.65	0.81	0.79	0.65	0.07	0.33	0.28	0.78	0.04
Storage	47.1 MB	0.9 MB	14.9 MB	0.4 MB	100.4 MB	0.3 MB	86.8 MB	0.5 MB	435.5 MB	0.4 MB

in [202] but it does not apply to weighted graphs. Hence, when computing homophily, we binarize the graphs by removing edges whose weights are smaller than 0.5. We make the following observations. First, while achieving similar performance for downstream tasks, the condensed graphs contain fewer nodes and take much less storage. Second, the condensed graphs are less sparse than their larger counterparts. Since the condensed graph is on extremely small scale, there would be almost no connections between nodes if the condensed graph maintains the original sparsity. Third, for Citeseer, Cora and Flickr, the homophily information are well preserved in the condensed graphs.

Visualization. We present the visualization results for all datasets in Figure A.2, where nodes with the same color are from the same class. Notably, as the learned condensed graphs are weighted graphs, we use black lines to denote the edges with weights larger than 0.5 and gray lines to denote the edges with weights smaller than 0.5. From Figure A.2, we can observe some patterns in the condensed graphs, e.g., the homophily patterns on Cora and Citeseer are well preserved. Interestingly, the learned graph for Reddit is very close to a star graph where almost all the nodes only have connections with very few center nodes. Such a structure can be meaningless for GNNs because almost all the nodes receive the information from their neighbors. In this case, the learned features $\mathbf{X}'$ play a major role in training GNN parameters, indicating that the original training graph of Reddit is not very informative, aligning with our observations in Section 4.2.

2.5 Conclusion

The prevalence of large-scale graphs poses great challenges in training graph neural networks. Thus, we study a novel problem of *graph condensation* which targets at condensing a large-real graph



(a) Cora, r=2.5% (b) Citeseer, r=1.8% (c) Arxiv, r=0.05% (d) Flickr, r=0.1% (e) Reddit, r=0.1%

Figure 2.2: Condensed graphs sometimes exhibit structure mimicking the original (a, b, d). Other times (c, e), learned features absorb graph properties and create less explicit graph reliance.

into a small-synthetic one while maintaining the performances of GNNs. Through our proposed framework, we are able to significantly reduce the graph size while approximating the original performance. The condensed graphs take much less space of storage and can be used to efficiently train various GNN architectures. Future work can be done on (1) improving the transferability of condensed graphs for different GNNs, (2) studying graph condensation for other tasks such as graph classification and (3) designing condensation framework for multi-label datasets.

CHAPTER 3

CONDENSING GRAPHS VIA ONE-STEP GRADIENT MATCHING

In this chapter, we propose to solve the limitations in current graph condensation methods: (1) they only produce continuous graph structures which can take more storage than discrete structures ; and (2) the condensation process is computationally expensive due to the involved nested optimization. To bridge the gap, we investigate efficient graph dataset condensation where we model the discrete graph structure as a probabilistic model. We further propose a one-step gradient matching scheme, which performs gradient matching for only one single step without training the network weights. Our theoretical analysis shows this strategy can generate synthetic graphs that lead to lower classification loss on real graphs. Extensive experiments on various graph datasets demonstrate the effectiveness and efficiency of the proposed method. In particular, we are able to reduce the dataset size by 90% while approximating up to 98% of the original performance and our method is significantly faster than multi-step gradient matching (e.g. 15× in CIFAR10 for synthesizing 500 graphs).

3.1 Introduction

Graph-structured data plays a key role in various real-world applications. For example, by exploiting graph structural information, we can predict the chemical property of a given molecular graph [170], detect fraud activities in a financial transaction graph [141], or recommend new friends to users in a social network [39]. Due to its prevalence, graph neural networks (GNNs) [78, 139, 4, 156] have been developed to effectively extract meaningful patterns from graph data and thus tremendously facilitate computational tasks on graphs. Despite their effectiveness, GNNs are notoriously data-hungry like traditional deep neural networks: they usually require massive datasets to learn powerful representations. Thus, training GNNs is often computationally expensive. Such cost even becomes prohibitive when we need to repeatedly train GNNs, e.g., in neural architecture search [91] and continual learning [87].

One potential solution to alleviate the aforementioned issue is *dataset condensation* or *dataset distillation*. It targets at constructing a small-synthetic training set that can provide sufficient

information to train neural networks [144, 189, 187, 110, 111, 15, 143]. In particular, one of the representative methods, DC [189], formulates the condensation goal as matching the gradients of the network parameters between small-synthetic and large-real training data. It has been demonstrated that such a solution can greatly reduce the training set size of image datasets without significantly sacrificing model performance. For example, using 100 images generated by DC can achieve 97.4% test accuracy on MNIST compared with 99.6% on the original dataset ($60,000$ images). These condensed samples can significantly save space for storing datasets and speed up retraining neural networks in many critical applications, e.g., continual learning and neural architecture search. In spite of the recent advances in dataset distillation/condensation for images, limited attention has been paid on domains involving graph structures.

To bridge this gap, we investigate the problem of condensing graphs such that GNNs trained on condensed graphs can achieve comparable performance to those trained on the original dataset. However, directly applying existing solutions for dataset condensation [144, 189, 187, 110] to graph domain faces some challenges. First, existing solutions have been designed for images where the data is continuous and they cannot output binary values to form the discrete graph structure. Thus, we need to develop a strategy that can handle the discrete nature of graphs. Second, they usually involve a complex bi-level problem that is computationally expensive to optimize: they require multiple iterations (inner iterations) of updating neural network parameters before updating the synthetic data for multiple iterations (outer iterations). It can be catastrophically inefficient for learning pairwise relations for nodes, of which the complexity is quadratic to the number of nodes. While one recent work targets at graph condensation for node classification [72], it does not overcome these challenges because it does not produce discrete graph structures and its condensation process is costly.

To address the aforementioned challenges, we propose an efficient condensation method for graphs, where we follow DC [189] to match the gradients of GNNs between synthetic graphs and real graphs. In order to produce discrete values, we model the graph structure as a probabilistic graph model and optimize the discrete structures in a differentiable manner. Based on this formulation, we further propose a *one-step gradient matching* strategy which only performs gradient

matching for one single step. Consequently, the advantages of the proposed strategy are twofold. First, it significantly speeds up the condensation process while providing reasonable guidance for synthesizing condensed graphs. Second, it removes the burden of tuning hyper-parameters such as the number of outer/inner iterations of the bi-level optimization as required by DC. Furthermore, we demonstrate the effectiveness of the proposed one-step gradient matching strategy both theoretically and empirically. Our contributions can be summarized as follows:

1. We study a novel problem of learning discrete synthetic graphs for condensing graph datasets, where the discrete structure is captured via a graph probabilistic model that can be learned in a differentiable manner.
2. We propose a one-step gradient matching scheme that significantly accelerates the vanilla gradient matching process.
3. Theoretical analysis is provided to understand the rationality of the proposed one-step gradient matching. We show that learning with one-step matching produces synthetic graphs that lead to a small classification loss on real graphs.
4. Extensive experiments have demonstrated the effectiveness and efficiency of the proposed method. Particularly, we are able to reduce the dataset size by 90% while approximating up to 98% of the original performance and our method is significantly faster than multi-step gradient matching (e.g. 15× in CIFAR10 for synthesizing 500 graphs).

3.2 Related Work

Graph Neural Networks. As the generalization of deep neural network to graph data, graph neural networks (GNNs) [78, 79, 139, 156, 151, 134, 70, 94, 146] have revolutionized the field of graph representation learning through effectively exploiting graph structural information. GNNs have achieved remarkable performances in basic graph-related tasks such as graph classification [161, 55], link prediction [39] and node classification [78]. Recent years have also witnessed their great success achieved in many real-world applications such as recommender systems [39], computer vision [82], drug discovery [33], computational biology [149, 132], and etc. GNNs take both adjacency matrix and node feature matrix as input and output node-level representations or graph-level representations.

Essentially, they follow a message-passing scheme [49] where each node first aggregates the information from its neighborhood and then transforms the aggregated information to update its representation. Furthermore, there is significant progress in developing deeper GNNs [93, 68], self-supervised GNNs [171, 173, 145, 65] and graph data augmentation [194, 28, 192].

Dataset Distillation & Dataset Condensation. It is widely received that training neural networks on large datasets can be prohibitively costly. To alleviate this issue, dataset distillation (DD) [144] aims to distill knowledge of a large training dataset into a small number of synthetic samples. DD formulates the distillation process as a learning-to-learning problem and solves it through bi-level optimization. To improve the efficiency of DD, dataset condensation (DC) [189, 187] is proposed to learn the small synthetic dataset by matching the gradients of the network parameters w.r.t. large-real and small-synthetic training data. It has been demonstrated that these condensed samples can facilitate critical applications such as continual learning [189, 187, 76, 81, 188], neural architecture search [110, 111, 165] and privacy-preserving scenarios [30] Recently, following the gradient matching scheme in DC, the work [72] proposes a condensation method to condense a large-scale graph to a small graph for node classification. Different from [72] which learns weighted graph structure, we aim to solve the challenge of learning discrete structure and we majorly target at graph classification. Moreover, our method avoids the costly bi-level optimization and is much more efficient than the previous work. A detailed comparison is included in Section 3.4.4.

3.3 The Proposed Framework

Before detailing the framework, we first introduce the main notations used in this paper. We majorly focus on the graph classification task where the goal is to predict the labels of given graphs. Specifically, we denote a graph dataset as $\mathcal{T} = \{G_1, \ldots, G_N\}$ with ground-truth label set $\mathcal{Y}$. Each graph in $\mathcal{T}$ is associated with a discrete adjacency matrix and a node feature matrix. Let $\mathbf{A}_{(i)}, \mathbf{X}_{(i)}$ represent the adjacency matrix and the feature matrix of i-th real graph, respectively. Similarly, we use $\mathcal{S} = \{G'_1, \ldots, G'_{N'}\}$ and $\mathcal{Y}'$ to indicate the synthetic graphs and their labels, respectively. Note that the number of synthetic graphs N' is essentially much smaller than that of real graphs N. We use d and n to denote the number of feature dimensions and number of nodes in each synthetic

graph, respectively[1]. Let C denote the number of classes and ℓ denote the cross entropy loss. The goal of our work is to learn a set of synthetic graphs $\mathcal{S}$ such that a GNN trained on $\mathcal{S}$ can achieve comparable performance to the one trained on the much larger dataset $\mathcal{T}$.

In the following subsections, we first introduce how to apply the vanilla gradient matching to condensing graphs for graph classification (Section 3.3.1). However, it cannot generate discrete graph structure and is highly inefficient. To correspondingly address these two limitations, we discuss the approach to handling the discrete nature of graphs (Section 3.3.2) and propose an efficient solution, one-step gradient matching, which significantly accelerates the condensation process (Section 3.3.3).

3.3.1 Gradient Matching as the Objective

Since we aim at learning synthetic graphs that are highly informative, one solution is to allow GNNs trained on synthetic graphs to imitate the training trajectory on the original large dataset. Dataset condensation [189, 187] introduces a gradient matching scheme to achieve this goal. Concretely, it tries to reduce the difference of model gradients w.r.t. large-real data and small-synthetic data for model parameters at every training epoch. Hence, the model parameters trained on synthetic data will be close to these trained on real data at every training epoch. Let θ_t denote the network parameters at the t-th epoch and f_{θ_t} indicate the neural network parameterized by θ_t. The condensation objective is expressed as:

$$\min_{\mathcal{S}} \sum_{t=0}^{T-1} D(\nabla_\theta \ell(f_{\theta_t}(\mathcal{S}), \mathcal{Y}'), \nabla_\theta \ell(f_{\theta_t}(\mathcal{T}), \mathcal{Y})),$$
$$\text{s.t. } \theta_{t+1} = \text{opt}_\theta(\theta_t, \mathcal{S}), \tag{3.1}$$

where $D(\cdot, \cdot)$ is a distance function, T is the number of steps of the whole training trajectory and $\text{opt}_\theta(\cdot)$ is the optimization operator for updating parameter θ. Note that Eq. (3.1) is a bi-level problem where we need to learn the synthetic graphs $\mathcal{S}$ at the outer optimization and update model parameters θ_t at the inner optimization. To learn synthetic graphs that generalize to a distribution of

[1]We set n to the average number of nodes in original dataset.

model parameters P_{θ_0}, we sample $\theta_0 \sim P_{\theta_0}$ and rewrite Eq. (3.1) as:

$$\min_{\mathcal{S}} \mathop{\mathbb{E}}_{\theta_0 \sim P_{\theta_0}} \left[\sum_{t=0}^{T-1} D\left(\nabla_\theta \ell\left(f_{\theta_t}(\mathcal{S}), \mathcal{Y}'\right), \nabla_\theta \ell\left(f_{\theta_t}(\mathcal{T}), \mathcal{Y}\right)\right) \right],$$
$$\text{s.t. } \theta_{t+1} = \text{opt}_\theta(\theta_t, \mathcal{S}). \tag{3.2}$$

Discussion. The aforementioned strategy has demonstrated promising performance on condensing image datasets [189, 187]. However, it is not clear how to model the discrete graph structure. Moreover, the inherent bi-level optimization inevitably hinders its scalability. To tackle these shortcomings, we propose *DosCond* that models the structure as a probabilistic graph model and is optimized through one-step gradient matching. In the following subsections, we introduce the details of *DosCond*.

3.3.2 Learning Discrete Graph Structure

For graph classification, each graph in the dataset is composed of an adjacency matrix and a feature matrix. For simplicity, we use $\mathbf{X}' \in \mathbb{R}^{N' \times n \times d}$ to denote the node features in all synthetic graphs $\mathcal{S}$ and $\mathbf{A}' \in \{0,1\}^{N' \times n \times n}$ to indicate the graph structure information in $\mathcal{S}$. Note that f_{θ_t} can be instantiated as any graph neural network and it takes both graph structure and node features as input. Then we rewrite the objective in Eq. (4.9) as follows:

$$\min_{\mathbf{A}', \mathbf{X}'} \mathop{\mathbb{E}}_{\theta_0 \sim P_{\theta_0}} \left[\sum_{t=0}^{T-1} D\left(\nabla_\theta \ell\left(f_{\theta_t}(\mathbf{A}', \mathbf{X}'), \mathcal{Y}'\right), \nabla_\theta \ell\left(f_{\theta_t}(\mathcal{T}), \mathcal{Y}\right)\right) \right],$$
$$\text{s.t. } \theta_{t+1} = \text{opt}_\theta(\theta_t, \mathcal{S}), \tag{3.3}$$

where we aim to learn both graph structure $\mathbf{A}'$ and node features $\mathbf{X}'$. However, Eq. (3.3) is challenging to optimize as it requires a function that outputs binary values. To address this issue, we propose to model the graph structure as a probabilistic graph model with Bernoulli distribution. Note that in the following, we reshape $\mathbf{A}'$ from $N' \times n \times n$ to $N' \times n^2$ for the purpose of demonstration only. Specifically, for each entry $\mathbf{A}'_{ij} \in \{0,1\}$ in the adjacency matrix $\mathbf{A}'$, it follows a Bernoulli distribution:

$$P_{\mathbf{\Omega}_{ij}}(\mathbf{A}'_{ij}) = \mathbf{A}'_{ij}\sigma(\mathbf{\Omega}_{ij}) + (1 - \mathbf{A}'_{ij})\sigma(-\mathbf{\Omega}_{ij}), \tag{3.4}$$

where $\sigma(\cdot)$ is the sigmoid function; $\mathbf{\Omega}_{ij} \in \mathbb{R}$ is the success probability of the Bernoulli distribution and also the parameter to be learned. Since $\mathbf{A}'_{ij}$ is independent of all other entries, the distribution of $\mathbf{A}'$ can be modeled as:

$$P_{\mathbf{\Omega}}(\mathbf{A}') = \prod_{i=1}^{N'} \prod_{j=1}^{n^2} P_{\mathbf{\Omega}_{ij}} \left(\mathbf{A}'_{ij} \right). \tag{3.5}$$

Then, the objective in Eq. (4.9) needs to be modified to

$$\min_{\mathbf{A}', \mathbf{X}'} \mathop{\mathbb{E}}_{\theta_0 \sim P_{\theta_0}} \left[\mathop{\mathbb{E}}_{\mathbf{A}' \sim P_{\mathbf{\Omega}}} \left[\ell(\mathbf{A}'(\mathbf{\Omega}), \mathbf{X}', \theta_0) \right] \right]. \tag{3.6}$$

With the new parameterization, we obtain a function that outputs discrete values but it is not differentiable due to the involved sampling process. Thus, we employ the reparameterization method [106], binary concrete distribution, to refactor the discrete random variable into a differentiable function of its parameters and a random variable with fixed distribution. Specifically, we first sample $\alpha \sim \text{Uniform}(0, 1)$, and edge weight $\mathbf{A}'_{ij} \in [0, 1]$ is calculated by:

$$\mathbf{A}'_{ij} = \sigma \left(\left(\log \alpha - \log(1 - \alpha) + \mathbf{\Omega}_{ij} \right) / \tau \right), \tag{3.7}$$

where $\tau \in (0, \infty)$ is the temperature parameter that controls the continuous relaxation. As $\tau \to 0$, the random variable $\mathbf{A}'_{ij}$ smoothly approaches the Bernoulli distribution. In other words, we have $\lim_{\tau \to 0} P\left(\mathbf{A}'_{ij} = 1 \right) = \sigma(\mathbf{\Omega}_{ij})$. While small τ is necessary for obtaining discrete samples, large τ is useful in getting large gradients as suggested by [106]. In practice, we employ an annealing schedule [1] to gradually decrease the value of τ in training. With the reparameterization trick, the objective function becomes differentiable w.r.t. $\mathbf{\Omega}_{ij}$ with well-defined gradients. Then we rewrite our objective as:

$$\begin{aligned} &\min_{\mathbf{\Omega}, \mathbf{X}'} \mathop{\mathbb{E}}_{\theta_0 \sim P_{\theta_0}} \left[\mathop{\mathbb{E}}_{\alpha \sim \text{Uniform}(0,1)} \left[\ell(\mathbf{A}'(\mathbf{\Omega}), \mathbf{X}', \theta_0) \right] \right] = \\ &\mathop{\mathbb{E}}_{\theta_0} \left[\mathop{\mathbb{E}}_{\alpha} \left[\sum_{t=0}^{T-1} D \left(\nabla_\theta \ell \left(f_{\theta_t}(\mathbf{A}'(\mathbf{\Omega}), \mathbf{X}'), \mathcal{Y}' \right), \nabla_\theta \ell \left(f_{\theta_t}(\mathcal{T}), \mathcal{Y} \right) \right) \right] \right], \\ &\text{s.t. } \theta_{t+1} = \text{opt}_\theta(\theta_t, \mathcal{S}). \end{aligned} \tag{3.8}$$

3.3.3 One-Step Gradient Matching

The vanilla gradient matching scheme in Eq. (4.9) presents a bi-level optimization problem. To solve this problem, we need to update the synthetic graphs $\mathcal{S}$ at the outer loop and then optimize the network parameters θ_t at the inner loop. The nested loops heavily impede the scalability of the condensation method, which motivates us to design a new strategy for efficient condensation. In this work, we propose a *one-step gradient matching* scheme where we only match the network gradients for the model initializations θ_0 while discarding the training trajectory of θ_t. Essentially, this strategy approximates the overall gradient matching loss for θ_t with the initial matching loss at the first epoch, which we term as *one-step matching loss*. The intuition is: the one-step matching loss informs us about the direction to update the synthetic data, in which, we have empirically observed a strong decrease in the cross-entropy loss (on real samples) obtained from the model trained on synthetic data. Hence, we can drop the summation symbol $\sum_{t=0}^{T-1}$ in Eq. (3.8) and simplify Eq. (3.8) as follows:

$$\min_{\mathbf{\Omega},\mathbf{X}'} \mathbb{E}_{\theta_0} \left[\mathbb{E}_{\alpha} \left[D \left(\nabla_\theta \ell \left(f_{\theta_0}(\mathbf{A}'(\mathbf{\Omega}), \mathbf{X}'), \mathcal{Y}' \right), \nabla_\theta \ell \left(f_{\theta_0}(\mathcal{T}), \mathcal{Y} \right) \right) \right] \right], \tag{3.9}$$

where we sample $\theta_0 \sim P_{\theta_0}$ and $\alpha \sim \text{Uniform}(0, 1)$. Compared with Eq. (3.8), one-step gradient matching avoids the expensive nested-loop optimization and directly updates the synthetic graph $\mathcal{S}$. It greatly simplifies the condensation process. In practice, as shown in Section 4.5.4, we find this strategy yields comparable performance to its bi-level counterpart while enabling much more efficient condensation. Next, we provide theoretical analysis to understand the rationality of the proposed one-step gradient matching scheme.

Theoretical Understanding. We denote the cross entropy loss on the real graphs as $\ell_{\mathcal{T}}(\theta) = \sum_i \ell_i(\mathbf{A}_{(i)}, \mathbf{X}_{(i)}, \theta)$ and that on synthetic graphs as $\ell_{\mathcal{S}}(\theta) = \ell_{\mathcal{S}}(\mathbf{A}'_{(i)}, \mathbf{X}'_{(i)}, \theta)$. Let θ^* denote the optimal parameter and θ_t be the parameter trained on $\mathcal{S}$ at the t-th epoch by optimizing $\ell_{\mathcal{S}}(\theta)$. For notation simplicity, we assume that $\mathbf{A}$ and $\mathbf{A}'$ are already normalized. The matrix norm $\|\cdot\|$ is the Frobenius norm. We focus on the GNN of Simple Graph Convolutions (SGC) [151] to study our problem since SGC has a simpler architecture but shares a similar filtering pattern as GCN.

Theorem 1. *When we use a K-layer SGC as the GNN used in condensation, i.e.,* $f_\theta(\mathbf{A}_{(i)}, \mathbf{X}_{(i)}) = Pool(\mathbf{A}_{(i)}^K \mathbf{X}_{(i)} \mathbf{W}_1)\mathbf{W}_2$ *with* $\theta = [\mathbf{W}_1; \mathbf{W}_2]$ *and assume that all network parameters satisfy* $\|\theta\|^2 \leq M^2 (M > 0)$*, we have*

$$\min_{t=0,1,\ldots,T-1} \ell_{\mathcal{T}}(\theta_t) - \ell_{\mathcal{T}}(\theta^*) \leq \sum_{t=0}^{T-1} \frac{\sqrt{2}M}{T} \|\nabla_\theta \ell_{\mathcal{T}}(\theta_t) - \nabla_\theta \ell_{\mathcal{S}}(\theta_t)\| + \frac{3M}{2\sqrt{T}} \cdot \frac{C-1}{CN'} \sqrt{\sum_i \gamma_i \|\mathbf{1}^\top \mathbf{A}'^K_{(i)} \mathbf{X}'_{(i)}\|^2} \tag{3.10}$$

where $\gamma_i = 1$ *if we use sum pooling in* f_θ*;* $\gamma_i = \frac{1}{n_i}$ *if we use mean pooling, with* n_i *as the number of nodes in the i-th synthetic graph.*

We provide the proof of Theorem 1 in Appendix B.1.1. Theorem 1 suggests that the smallest gap between the resulted loss (by training on synthetic graphs) and the optimal loss has an upper bound. This upper bound depends on two terms: (1) the difference of gradients w.r.t. real data and synthetic data and (2) the norm of input matrices. Thus, the theorem justifies that reducing the gradient difference w.r.t real and synthetic graphs can help learn desirable synthetic data that preserves sufficient information to train GNNs well. Based on Theorem 1, we have the following proposition.

Proposition 1. *Assume the largest gradient gap happens at* 0*-th epoch, i.e.,* $\|\nabla_\theta \ell_{\mathcal{T}}(\theta_0) - \nabla_\theta \ell_{\mathcal{S}}(\theta_0)\| = \max_t \|\nabla_\theta \ell_{\mathcal{T}}(\theta_t) - \nabla_\theta \ell_{\mathcal{S}}(\theta_t)\|$ *with* $t = 0, 1, \ldots, T-1$*, we have*

$$\min_{t=0,1,\ldots,T-1} \ell_{\mathcal{T}}(\theta_t) - \ell_{\mathcal{T}}(\theta^*) \leq \sqrt{2}M \|\nabla_\theta \ell_{\mathcal{T}}(\theta_0) - \nabla_\theta \ell_{\mathcal{S}}(\theta_0)\| + \frac{3M}{2\sqrt{T}} \cdot \frac{C-1}{CN'} \sqrt{\sum_i \gamma_i \|\mathbf{1}^\top \mathbf{A}'^K_{(i)} \mathbf{X}'_{(i)}\|^2}. \tag{3.11}$$

We omit the proof for the proposition since it is straightforward. The above proposition suggests that the smallest gap between the $\ell_{\mathcal{T}}(\theta_t)$ and $\ell_{\mathcal{T}}(\theta^*)$ is bounded by the one-step matching loss and the norm $\|\mathbf{1}^\top \mathbf{A}'^K_{(i)} \mathbf{X}'_{(i)}\|^2$. As we will show in Section 3.4.3.4, when using mean pooling, the second term tends to have a smaller scale than the first one and can be neglected; the second term matters more when we use sum pooling. Hence, we solely optimize the one-step gradient matching loss for GNNs with mean pooling and additionally include the second term (the norm of input matrices) as a

regularization for GNNs with sum pooling. As such, if we consider the optimal loss $\ell_{\mathcal{T}}(\theta^*)$ as a constant, reducing the one-step matching loss indeed learns synthetic graphs that lead to a small classification loss on real graphs. This demonstrates the rationality of one-step gradient matching from a theoretical perspective.

Remark 1. *Note that the spectral analysis from [151] demonstrated that both GCN and SGC share similar graph filtering behaviors. Thus practically, we extend the one-step gradient matching loss from K-layer SGC to K-layer GCN and observe that the proposed framework works well under the non-linear scenario.*

Remark 2. *While we focus on the graph classification task, it is straightforward to extend our framework to node classification and we obtain similar conclusions for node classification as shown in Theorem 2 in Appendix B.1.2.*

3.3.4 Final Objective and Training Algorithm

In this subsection, we describe the final objective function and the detailed training algorithm. We note that the objective in Eq. (3.8) involves two nested expectations, we adopt Monte Carlo to approximately optimize the objective function. Together with one-step gradient matching, we have

$$\begin{aligned} &\min_{\mathbf{\Omega},\mathbf{X}'} \mathop{\mathbb{E}}_{\theta_0\sim P_{\theta_0}} \mathop{\mathbb{E}}_{\alpha\sim\text{Uniform}(0,1)} \left[\left[\ell(\mathbf{A}'(\mathbf{\Omega}),\mathbf{X}',\theta_0)\right]\right] \\ &\approx \sum_{k_1=1}^{K_1}\sum_{k_2=1}^{K_2} D\left(\nabla_\theta \ell\left(f_{\theta_0}(\mathbf{A}'(\mathbf{\Omega}),\mathbf{X}'),\mathcal{Y}'\right), \nabla_\theta \ell\left(f_{\theta_0}(\mathcal{T}),\mathcal{Y}\right)\right) \end{aligned} \tag{3.12}$$

where K_1 is the number of sampled model initializations and K_2 is the number of sampled graphs. We find that $K_2 = 1$ is able to yield good performance in our experiments.

Regularization. In addition to the one-step gradient matching loss, we note that the proposed *DosCond* can be easily integrated with various priors as regularization terms. In this work, we focus on exerting sparsity regularization on the adjacency matrix, since a denser adjacency matrix will lead to higher cost for training graph neural networks. Specifically, we penalize the difference of the

Table 3.1: The classification performance comparison to baselines. We report the ROC-AUC for the first three datasets and accuracies (%) for others. *Whole Dataset* indicates the performance with original dataset.

	Graphs/Cls.	Ratio	Random	Herding	K-Center	DCG	*DosCond*	Whole Dataset
ogbg-molbace (ROC-AUC)	1	0.2%	0.580±0.067	0.548±0.034	0.548±0.034	0.623±0.046	**0.657±0.034**	0.714±0.005
	10	1.7%	0.598±0.073	0.639±0.039	0.591±0.056	0.655±0.033	**0.674±0.035**	
	50	8.3%	0.632±0.047	0.683±0.022	0.589±0.025	0.652±0.013	**0.688±0.012**	
ogbg-molbbbp (ROC-AUC)	1	0.1%	0.519±0.016	0.546±0.019	0.546±0.019	0.559±0.044	**0.581±0.005**	0.646±0.004
	10	1.2%	0.586±0.040	**0.605±0.019**	0.530±0.039	0.568±0.032	**0.605±0.008**	
	50	6.1%	0.606±0.020	0.617±0.003	0.576±0.019	0.579±0.032	**0.620±0.007**	
ogbg-molhiv (ROC-AUC)	1	0.01%	0.719±0.009	0.721±0.002	0.721±0.002	0.718±0.013	**0.726±0.003**	0.757±0.007
	10	0.06%	0.720±0.011	0.725±0.006	0.713±0.009	0.728±0.002	**0.728±0.005**	
	50	0.3%	0.721±0.014	0.725±0.003	0.725±0.006	0.726±0.010	**0.731±0.004**	
DD (Accuracy)	1	0.2%	57.69±4.92	61.97±1.32	61.97±1.32	58.81±2.90	**70.42±2.21**	78.92±0.64
	10	2.1%	64.69±2.55	69.79±2.30	63.46±2.38	61.84±1.44	**73.53±1.13**	
	50	10.6%	67.29±1.53	73.95±1.70	67.41±0.92	61.27±1.01	**77.04±1.86**	
MUTAG (Accuracy)	1	1.3%	67.47±9.74	70.84±7.71	70.84±7.71	75.00±8.16	**82.21±1.61**	88.63±1.44
	10	13.3%	77.89±7.55	80.42±1.89	81.00±2.51	82.66±0.68	**82.76±2.31**	
	20	26.7%	78.21±5.13	80.00±1.10	82.97±4.91	82.89±1.03	**83.26±2.34**	
NCI1 (Accuracy)	1	0.1%	51.27±1.22	53.98±0.67	53.98±0.67	51.14±1.08	**56.58±0.48**	71.70±0.20
	10	0.6%	54.33±3.14	57.11±0.56	53.21±1.44	51.86±0.81	**58.02±1.05**	
	50	3.0%	58.51±1.73	58.94±0.83	56.58±3.08	52.17±1.90	**60.07±1.58**	
CIFAR10 (Accuracy)	1	0.06%	15.61±0.52	22.38±0.49	22.37±0.50	21.60±0.42	**24.70±0.70**	50.75±0.14
	10	0.2%	23.07±0.76	28.81±0.35	20.93±0.62	29.27±0.77	**30.70±0.23**	
	50	1.1%	30.56±0.81	33.94±0.37	24.17±0.51	34.47±0.52	**35.34±0.14**	
E-commerce (Accuracy)	1	0.2%	51.31±2.89	52.18±0.25	52.36±0.38	57.14±1.72	**60.82±1.23**	69.25±0.50
	10	0.9%	54.99±2.74	56.83±0.87	56.49±0.36	61.03±1.32	**64.73±1.34**	
	20	3.6%	57.80±3.58	62.56±0.71	62.76±0.45	64.92±1.35	**67.71±1.22**	

sparsity between $\sigma(\Omega)$ and a given sparsity ϵ:

$$\ell_{\text{reg}} = \max\left(\frac{1}{|\mathbf{\Omega}|}\sum_{i,j}\sigma(\mathbf{\Omega}_{ij}) - \epsilon, 0\right). \tag{3.13}$$

We initialize $\sigma(\mathbf{\Omega})$ and $\mathbf{X}'$ as randomly sampled training graphs[2] and set ϵ to the average sparsity of initialized $\sigma(\mathbf{\Omega})$ so as to maintain a low sparsity. On top of that, as we discussed earlier in Section 3.3.3, we include the following regularization for GNNs with sum pooling:

$$\ell_{\text{reg2}} = \frac{3}{2\sqrt{2T}} \cdot \frac{C-1}{CN'} \sqrt{\sum_{i} \|\mathbf{1}^\top \mathbf{A}'^{K}_{(i)} \mathbf{X}'_{(i)}\|^2} \tag{3.14}$$

Training Algorithm. We provide the details of our proposed framework in Algorithm 4 in Appendix B.2.1. Specifically, we sample K_1 model initializations θ_0 to perform one-step gradient

[2]If an entry in the real adjacency matrix is 1, the corresponding value in $\mathbf{\Omega}$ is initialized as a large value, e.g.,5.

matching. Following the convention in DC [189], we match gradients and update synthetic graphs for each class separately in order to make matching easier. For class c, we first retrieve the synthetic graphs of that class, denoted as $(\mathbf{A}'_c, \mathbf{X}'_c, \mathcal{Y}'_c) \sim \mathcal{S}$, and sample a batch of real graphs $(\mathbf{A}_c, \mathbf{X}_c, \mathcal{Y}_c)$. We then forward them to the graph neural network and calculate the one-step gradient matching loss together with the regularization term. Afterwards, $\mathbf{\Omega}$ and $\mathbf{X}'$ are updated via gradient descent. It is worth noting that the training process for each class can be run in parallel since the graph updates for one class is independent of another class.

Comparison with DC. Recall that the gradient matching scheme in DC involves a complex bi-level optimization. If we denote the number of inner-iterations as τ_i and that of outer-iterations as τ_o, its computational complexity can be $\tau_i \times \tau_o$ of our method. Thus DC is significantly slower than *DosCond*. In addition to speeding up condensation, *DosCond* removes the burden of tuning some hyper-parameters, i.e., the number of iterations for outer/inner optimization and learning rate for updating f_θ, which can potentially save us enormous training time when learning larger synthetic sets.

Comparison with Coreset Methods. Coreset methods [148, 125] select representative data samples based on some heuristics calculated on the pre-trained embedding. Thus, it requires training the model first. Given the cheap cost on calculating and ranking heuristics, the major computational bottleneck for coreset method is on pre-training the neural network for a certain number of iterations. Likewise, our proposed *DosCond* has comparable complexity because it also needs to forward and backward the neural network for multiple iterations. Thus, their efficiency difference majorly depends on how many epochs we run for learning synthetic graphs in *DosCond* and for pre-training the model embedding in coreset methods. In practice, we find that *DosCond* even requires less training cost than the coreset methods as shown in Section 3.4.2.2.

3.4 Experiment

In this section, we conduct experiments to evaluate *DosCond*. Particularly, we aim to answer the following questions: (a) how well can we condense a graph dataset and (b) how efficient is *DosCond*.

Our code can be found in the supplementary files.

3.4.1 Experimental Settings

Datasets. To evaluate the performance of our method, we use multiple molecular datasets from Open Graph Benchmark (OGB) [60] and TU Datasets (DD, MUTAG and NCI1) [109] for graph-level property classification, and one superpixel dataset CIFAR10 [34]. We also introduce a real-world e-commerce dataset. In particular, we randomly sample 1,109 sub-graphs from a large, anonymized internal knowledge graph. Each sub-graph is created from the ego network of a random selected product on the e-commerce website. We form a binary classification problem aiming at predicting the product category of the central product node in each sub-graph. We use the public splits for OGB datasets and CIFAR10. For TU Datasets and the e-commerce dataset, we randomly split the graphs into 80%/10%/10% for training/validation/test. Detailed dataset statistics are shown in Appendix C.3.

Baselines. We compare our proposed methods with four baselines that produce discrete structures: three coreset methods (*Random*, *Herding* [148] and *K-Center* [40, 125]), and a *dataset condensation* method DCG [189]: (a) Random: it randomly picks graphs from the training dataset. (b) Herding: it selects samples that are closest to the cluster center. Herding is often used in replay-based methods for continual learning [118, 14]. (c) K-Center: it selects the center samples to minimize the largest distance between a sample and its nearest center. (d) DCG: As vanilla DC [189] cannot generate discrete structure, we randomly select graphs from training and apply DC to learn the features for them, which we term as DCG. We use the implementations provided by [189] for Herding, K-Center and DCG. Note that coreset methods only select existing samples from training while DCG learns the node features.

Evaluation Protocol. To evaluate the effectiveness of the proposed method, we test the classification performance of GNNs trained with condensed graphs on the aforementioned graph datasets. Concretely, it involves three stages: (1) learning synthetic graphs, (2) training a GCN on the synthetic graphs and (3) test the performance of GCN. We first generate the condensed graphs following the procedure in Algorithm 1. Then we train a GCN classifier with the condensed graphs.

Finally we evaluate its classification performance on the real graphs from test set. For baseline methods, we first get the selected/condensed graphs and then follow the same procedure. We repeat the generation process of condensed graphs 5 times with different random seeds and train GCN on these graphs with 10 different random seeds. In all experiments, we report the mean and standard deviation of these results.

Parameter Settings. When learning the synthetic graphs, we adopt 3-layer GCN with 128 hidden units as the model for gradient matching. The learning rates for structure and feature parameters are set to 1.0 (0.01 for ogbg-molbace and CIFAR10) and 0.01, respectively. We set K_1 to 1000 and β to 0.1. Additionally, we use mean pooling to obtain graph representation for all datasets except ogbg-molhiv. We use sum pooling for ogbg-molhiv as it achieves better classification performance on the real dataset. During the test stage, we use GCN with the same architecture and we train the model for 500 epochs (100 epochs for ogbg-molhiv) with an initial learning rate of 0.001.

3.4.2 Performance with Condensed Graphs

3.4.2.1 Classification Performance Comparison

To validate the effectiveness of the proposed framework, we measure the classification performance of GCN trained on condensed graphs. Specifically, we vary the number of learned synthetic graphs per class in the range of $\{1, 10, 50\}$ ($\{1, 10, 20\}$ for MUTAG and E-commerce) and train a GCN on these graphs. Then we evaluate the classification performance of the trained GCN on the original test graphs. Following the convention in OGB [60], we report the ROC-AUC metric for ogbg-molbace, ogbg-molbbbp and ogbg-molhiv; for other datasets we report the classification accuracy (%). The results are summarized in Table 3.1. Note that the *Ratio* column presents the ratio of synthetic graphs to original graphs and we name it as *condensation ratio*; the *Whole Dataset* column shows the GCN performance achieved by training on the original dataset. From the table, we make the following observations:

(a) The proposed *DosCond* consistently achieves better performance than the baseline methods under different condensation ratios and different datasets. Notably, when generating only 2 graphs on

ogbg-molbace dataset (0.2%), we achieve an ROC-AUC of 0.657 while the performance on full training set is 0.714, which means we approximate 92% of the original performance with only 0.2% data. Likewise, we are able to approximate 96.5% of the original performance on ogbg-molhiv with 0.3% data. By contrast, baselines underperform our method by a large margin. Similar observations can be made on other datasets, which demonstrates the effectiveness of learned synthetic graphs in preserving the information of the original dataset.

(b) Increasing the number of synthetic graphs can improve the classification performance. For example, we can approximate the original performance by 89%/93%/98% with 0.2%/2.1%/10.6% data on DD. More synthetic samples indicate more learnable parameters that can preserve the information residing in the original dataset and present more diverse patterns that can help train GNNs better. This observation is in line with our experimental results in Section 3.4.3.1.

(c) The performance on CIFAR10 is less promising due to the limit number of synthetic graphs. We posit that the dataset has more complex topology and feature information and thus requires more parameters to preserve sufficient information. However, we note that our method still outperforms the baseline methods especially when producing only 1 sample per class, which suggests that our method is much more data-efficient. Moreover, we are able to promote the performance on CIFAR10 by learning a larger synthetic set as shown in Section 3.4.3.1.

(d) Learning both synthetic graph structure and node features is necessary for preserving the information in original graph datasets. By checking the performance DCG, which only learns node features based on randomly selected graph structure, we see that DCG underperforms *DosCond* by a large margin in most cases. This indicates that learning node features solely is sub-optimal for condensing graphs.

3.4.2.2 Efficiency Comparison

Since one of our goals is to enable scalable dataset condensation, we now evaluate the efficiency of *DosCond*. We compare *DosCond* with the coreset method Herding, as it is less time-consuming than DCG and generally achieves better performance than other baselines. We adopt the same setting as in Table 3.1: 1000 iterations for *DosCond*, i.e., $K_1 = 1000$, and 500 epochs (100 epochs

Table 3.2: Comparison of running time (minutes).

	CIFAR10		ogbg-molhiv		DD	
G./Cls.	Herding	*DosCond*	Herding	*DosCond*	Herding	*DosCond*
1	44.5m	4.7m	4.3m	0.66m	1.6m	1.5m
10	44.5m	4.9m	4.3m	0.67m	1.6m	1.5m
50	44.5m	5.7m	4.3m	0.68m	1.6m	2.0m

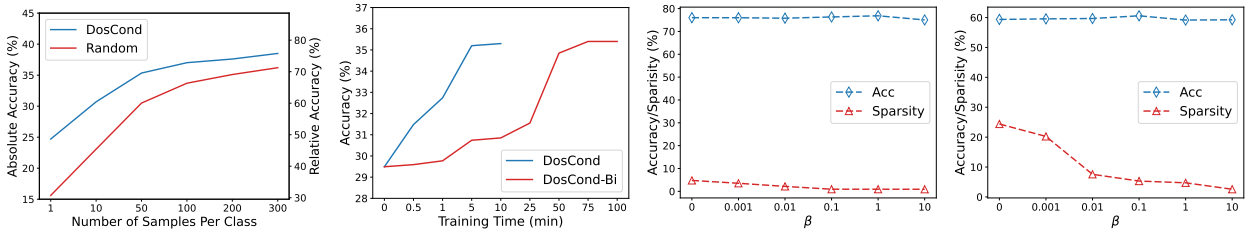
for ogbg-molhiv) for pre-training the graph convolutional network as required by Herding. We also note that pre-training the neural network need to go over the whole dataset at every epoch while *DosCond* only processes a batch of graphs. In Table A.5, we report the running time on an NVIDIA V100 GPU for CIFAR10, ogbg-molhiv and DD. From the table, we make two observations:

(a) *DosCond* can be faster than Herding. In fact, *DosCond* requires less training time in all the cases except in DD with 50 graphs per class. Herding needs to fully train the model on the whole dataset to obtain good-quality embedding, which can be quite time-consuming. On the contrary, *DosCond* only requires matching gradients for K_1 initializations and does not need to fully train the model on the large real dataset.

(b) The running time of *DosCond* increases with the increase of the number of synthetic graphs N'. It is because *DosCond* processes the condensed graphs at each iteration, of which the time complexity is $O(N'L(n^2d + nd^2))$ for an L-layer GCN. Thus, the additional complexity depends on N'. By contrast, the increase of N' has little impact on Herding since the process of selecting samples based on pre-defined heuristic is very fast.

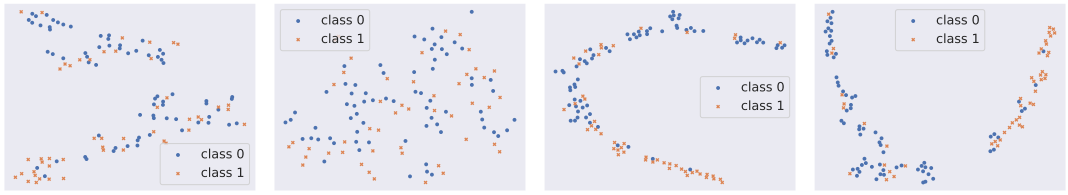
(c) The average nodes in synthetic graph n also impacts the training cost of *DosCond*. For instance, the training cost on ogbg-molhiv (n=26) is much lower than that on DD (n=285), and the gap of cost between the two methods on ogbg-molhiv and DD is very different. As mentioned earlier, it is because the complexity of the forward process in GCN is $O(N'L(n^2d + nd^2))$ for N' condensed graphs with node size of n.

To summarize, the efficiency difference of Herding and *DosCond* depends on the number of condensed/selected samples and the training iterations adopted in practice and we empirically found that *DosCond* consumes less training cost.



(a) Larger synthetic set. (b) One-step v.s. bi-level (c) Varying β on DD (d) Varying β on NCI1

Figure 3.1: Algorithm analysis and parameter analysis w.r.t. the sparsity regularization.



(a) Random (b) DCG (c) *DosCond* (d) Whole Dataset

Figure 3.2: T-SNE visualizations of embedding learned with condensed graphs on DD.

3.4.3 Further Investigation

In this subsection, we perform further investigations to provide a better understanding of our proposed method.

3.4.3.1 Increasing the Number of Synthetic Graphs

We study whether the classification performance can be further boosted when using larger synthetic size. Concretely, we vary the size of the learned graphs from 1 to 300 and report the results of absolute and relative accuracy w.r.t. whole dataset training accuracy for CIFAR10 in Figure 3.1a. It is clear to see that both Random and *DosCond* achieve better performance when we increase the number of samples used for training. Moreover, our method outperforms the random baseline under different condensed dataset sizes. It is worth noting that the performance gap between the two methods diminishes with the increase of the number of samples. This is because the random baseline will finally approach the whole dataset training if we continue to enlarge the size of the condensed set, in which the performance can be considered as the upper bound of *DosCond*.

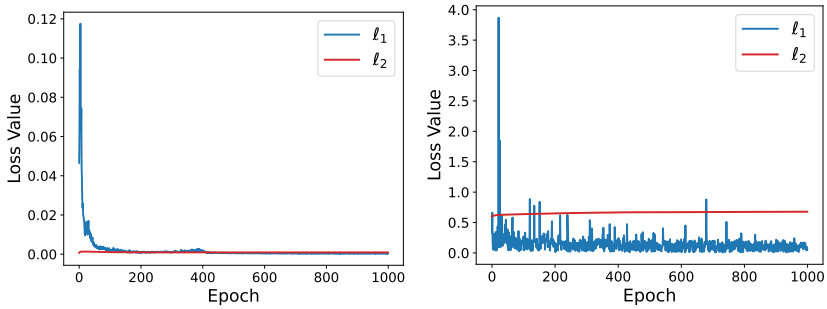
Table 3.3: Node classification accuracy (%) comparison. The numbers in parentheses indicate the running time for 100 epochs and r indicates the ratio of number of nodes in the condensed graph to that in the original graph.

	Cora, r=2.6%	Citeseer, r=1.8%	Pubmed, r=0.3%	Arxiv, r=0.25%	Flickr, r=0.1%
GCond	80.1 (75.9s)	70.6 (71.8s)	77.9 (51.7s)	59.2 (494.3s)	46.5 (51.9s)
DosCond	80.0 (3.5s)	71.0 (2.8s)	76.0 (1.3s)	59.0 (32.9s)	46.1 (14.3s)
Whole Dataset	81.5	71.7	79.3	71.4	47.2

3.4.3.2 Ablation Study

To examine how different model components affect the model performance, we perform ablation study on the proposed one-step gradient matching and regularization terms. We create an ablation of our method, namely *DosCond-Bi*, which adopts the vanilla gradient matching scheme that involves a bi-level optimization. Without loss of generality, we compare the training time and classification accuracy of *DosCond* and *DosCond-Bi* in the setting of learning 50 graphs/class synthetic graphs on CIFAR10 dataset. The results are summarized in Figure 3.1b and we can see that *DosCond* needs approximately 5 minutes to reach the performance of *DosCond-Bi* trained for 75 minutes, which indicates that *DosCond* only requires 6.7% training cost. It further demonstrates the efficiency of the proposed one-step gradient matching strategy.

Next we study the effect of sparsity regularization on *DosCond*. Specifically, we vary the sparsity coefficient β in the range of $\{0, 0.001, 0.01, 0.1, 1, 10\}$ and report the classification accuracy and graph sparsity on DD and NCI datasets in Figure 3.1c and 3.1d. Note that the graph sparsity is defined as the ratio of the number of edges to the square of the number of nodes. As shown in the figure, when β gets larger, we exert a stronger regularization on the learned graphs and the graphs become more sparse. Furthermore, the increased sparsity does not affect the classification performance. This is a desired property since sparse graphs can save much space for storage and reduce training cost for GNNs. We also remove the regularization of Eq. (3.14) for ogbg-molhiv, we obtain the performance of 0.724/ 0.727/0.731 for 1/10/50 graphs per class, which is slightly worse than the one with this regularization.



(a) DD (b) ogbg-molhiv

Figure 3.3: Scale of the two terms in Eq. (11).

3.4.3.3 Visualization

We further investigate whether GCN can learn discriminative representations from the synthetic graphs learned by *DosCond*. Specifically, we use t-SNE [137] to visualize the learned graph representation from GCN trained on different condensed graphs. We train a GCN on graphs produced by different methods and use it to extract the latent representation for real graphs from test set. Without loss of generality, we provide the t-SNE plots on DD dataset with 50 graphs per class in Figure A.2. It is observed that the graph representations learned with randomly selected graphs are mixed for different classes. This suggests that using randomly selected graphs cannot help GCN learn discriminative features. Similarly, DCG graphs also resulted in poorly trained GCN that outputs indistinguishable graph representations. By contrast, the representations are well separated for different classes when learned with *DosCond* graphs (Figure A.2c) and they are as discriminative as those learned on the whole training dataset (Figure A.2d). This demonstrates that the graphs learned by *DosCond* preserve sufficient information of the original dataset so as to recover the original performance.

3.4.3.4 Scale of the Two Terms in Eq. (3.11)

As mentioned earlier in Section 3.3.3, the scale of the first term is essentially larger than the second term in Eq. (3.11). We now perform empirical study to verify this statement. Since both terms contain the factor M, we simply drop it and focus on studying $\ell_1 = \sqrt{2}\|\nabla_\theta \ell_{\mathcal{T}}(\theta_0) - \nabla_\theta \ell_S(\theta_0)\|$ and $\ell_2 = \frac{3}{2\sqrt{T}} \cdot \frac{C-1}{CN'} \sqrt{\sum_i \gamma_i \|\mathbf{1}^\top \mathbf{A}'^K_{(i)} \mathbf{X}'_{(i)}\|^2}$. Specifically, we set T to 500 and N' to 50, and plot the

changes of these two terms during the training process of *DosCond*. The results on DD (with mean pooling) and ogbg-molhiv (with sum pooling) are shown in Figure 3.3. We can observe that the scale of ℓ_1 is much larger than ℓ_2 at the first few epochs when using mean pooling as shown in Figure 3.3a. By contrast, ℓ_2 is not negligible when using sum pooling as shown in Figure 3.3b and it is desired to include it as a regularization term in this case. These observations provide support for ours discussion of theoretical analysis in Section 3.3.3.

3.4.4 Node Classification

Next, we investigate whether the proposed method works well in node classification so as to support our analysis in Theorem 2 in Appendix B.1.2. Specifically, following *GCond* [72], a condensation method for node classification, we use 5 node classification datasets: Cora, Citeseer, Pubmed [78], ogbn-arxiv [60] and Flickr [177]. The dataset statistics are shown in B.2. We follow the settings in *GCond* to generate one condensed graph for each dataset, train a GCN on the condensed graph, and evaluate its classification performance on the original test nodes. To adopt *DosCond* into node classification, we replace the bi-level gradient matching scheme in *GCond* with our proposed one-step gradient matching. The results of classification accuracy and running time per epoch are summarized in Table 3.3. From the table, we make the following observations:

(a) The proposed *DosCond* achieves similar performance as *GCond* and the performance is also comparable to the original dataset. For example, we are able to approximate the original training performance by 99% with only 2.6% data on Cora. It demonstrates the effectiveness of *DosCond* in the node classification case and justifies Theorem 2 from an empirical perspective.

(b) The training cost of *DosCond* is essentially lower than *GCond* as *DosCond* avoids the expensive bi-level optimization. By examining their running time, we can see that *DosCond* is up to 40 times faster than *GCond*.

We further note that *GCond* produces weighted graphs which require storing the edge weights in float formats, while *DosCond* outputs discrete graph structure which can be stored as binary values. Hence, the graphs learned by *DosCond* are more memory-efficient.

3.5 Conclusion

Training graph neural networks on a large-scale graph dataset consumes high computational cost. One solution to alleviate this issue is to condense the large graph dataset into a small synthetic dataset. In this work, we propose a novel framework *DosCond* that adopts a one-step gradient matching strategy to efficiently condenses real graphs into a small number of informative graphs with discrete structures. We further justify the proposed method from both theoretical and empirical perspectives. Notably, our experiments show that we are able to reduce the dataset size by 90% while approximating up to 98% of the original performance. In the future, we plan to investigate interpretable condensation methods and diverse applications of the condensed graphs.

CHAPTER 4

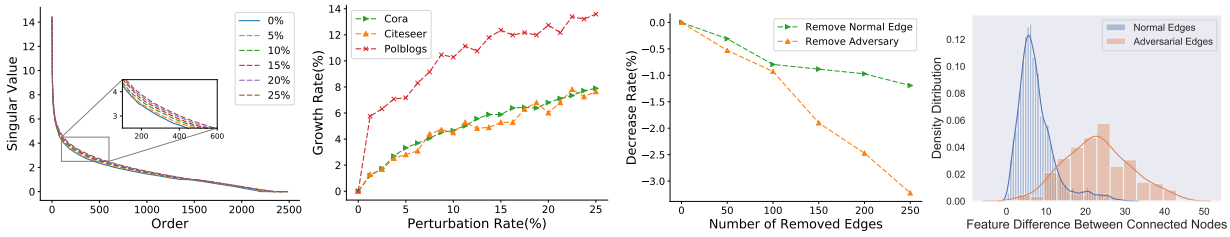
GRAPH STRUCTURE LEARNING FOR ROBUST GRAPH NEURAL NETWORKS

In this chapter, we investigate the training-time robustness of Graph Neural Networks (GNNs). Recent studies show that GNNs are vulnerable to carefully-crafted perturbations, called adversarial attacks. Adversarial attacks can easily fool GNNs into making predictions for downstream tasks. The vulnerability to adversarial attacks has raised increasing concerns about applying GNNs in safety-critical applications. Therefore, developing robust algorithms to defend adversarial attacks is of great significance. A natural idea to defend adversarial attacks is to clean the perturbed graph. It is evident that real-world graphs share some intrinsic properties. For example, many real-world graphs are low-rank and sparse, and the features of two adjacent nodes tend to be similar. In fact, we find that adversarial attacks are likely to violate these graph properties. Therefore, in this paper, we explore these properties to defend adversarial attacks on graphs. In particular, we propose a general framework Pro-GNN, which can jointly learn a structural graph and a robust graph neural network model from the perturbed graph guided by these properties. Extensive experiments on real-world graphs demonstrate that the proposed framework achieves significantly better performance compared with the state-of-the-art defense methods, even when the graph is heavily perturbed.

4.1 Introduction

Graphs are ubiquitous data structures in numerous domains, such as chemistry (molecules) [53], finance (trading networks) [184] and social media (the Facebook friend network) [122]. With their prevalence, it is particularly important to learn effective representations of graphs and then apply them to solve downstream tasks. Recent years have witnessed great success from Graph Neural Networks (GNNs) [86, 58, 78, 139] in representation learning of graphs. GNNs follow a message-passing scheme [50], where the node embedding is obtained by aggregating and transforming the embeddings of its neighbors. Due to the good performance, GNNs have been applied to various analytical tasks including node classification [78], link prediction [77], and recommender systems [168].

Although promising results have been achieved, recent studies have shown that GNNs are vulnerable to adversarial attacks [67, 209, 211, 22, 152]. In other words, the performance of GNNs can greatly degrade under an unnoticeable perturbation in graphs. The lack of robustness of these models can lead to severe consequences for critical applications pertaining to the safety and privacy. For example, in credit card fraud detection, fraudsters can create several transactions with only a few high-credit users to disguise themselves, thus escaping from the detection based on GNNs. Hence, developing robust GNN models to resist adversarial attacks is of significant importance. Modifying graph data can perturb either node features or graph structures. However, given the complexity of structural information, the majority of existing adversarial attacks on graph data have focused on modifying graph structure especially adding/deleting/rewiring edges [158]. Thus, in this work, we aim to defend against the most common setting of adversarial attacks on graph data, i.e., poisoning adversarial attacks on graph structure. Under this setting, the graph structure has already been perturbed by modifying edges before training GNNs while node features are not changed.



(a) Singular Values (b) Rank Growth (c) Rank Decrease Rate (d) Feature Smoothness

Figure 4.1: An illustrative example on the property changes of the adjacency matrix by adversarial attacks.

One perspective to design an effective defense algorithm is to clean the perturbed graph such as removing the adversarial edges and restoring the deleted edges [200, 133]. The key challenge from this perspective is what criteria we should follow to clean the perturbed graph. It is well known that real-world graphs often share certain properties. First, many real-world clean graphs are low-rank and sparse [199]. For instance, in a social network, most individuals are connected with only a small number of neighbors and there are only a few factors influencing the connections among users [199, 44]. Second, connected nodes in a clean graph are likely to share similar features

or attributes (or feature smoothness) [108]. For example, in a citation network, two connected publications often share similar topics [78]. Figure 4.1 demonstrates these properties of clean and poisoned graphs. Specifically, we apply the state-of-the-art graph poisoning attack, *metattack* [211], to perturb the graph data and visualize the graph properties before and after *mettack*. As shown in Figure 4.1a, *metattack* enlarges the singular values of the adjacency matrix and Figure 4.1b illustrates that *metattack* quickly increases the rank of adjacency matrix. Moreover, when we remove the adversarial and normal edges from the perturbed graph respectively, we observe that removing adversarial edges reduces the rank faster than removing normal edges as demonstrated in Figure 4.1c. In addition, we depict the density distribution of feature difference of connected nodes of the attacked graph in Figure 4.1d. It is observed that *metattack* tends to connect nodes with large feature difference. Observations from Figure 4.1 indicate that adversarial attacks could violate these properties. Thus, these properties have the potential to serve as the guidance to clean the perturbed graph. However, work of exploring these properties to build robust graph neural networks is rather limited.

In this chapter, we target on exploring graph properties of sparsity, low rank and feature smoothness to design robust graph neural networks. Note that there could be more properties to be explored and we would like to leave it as future work. In essence, we are faced with two challenges: (i) how to learn clean graph structure from poisoned graph data guided by these properties; and (ii) how to jointly learn parameters for robust graph neural network and the clean structure. To solve these two challenges, we propose a general framework Property GNN (Pro-GNN) to simultaneously learn the clean graph structure from perturbed graph and GNN parameters to defend against adversarial attacks. Extensive experiments on a variety of real-world graphs demonstrate that our proposed model can effectively defend against different types of adversarial attacks and outperforms the state-of-the-art defense methods.

4.2 Related Work

In line with the focus of our work, we briefly describe related work on GNNs, and adversarial attacks and defense for graph data.

4.2.1 Graph Neural Networks

Over the past few years, graph neural networks have achieved great success in solving machine learning problems on graph data. To learn effective representation of graph data, two main families of GNNs have been proposed, i.e., spectral methods and spatial methods. The first family learns node representation based on graph spectral theory [78, 10, 24]. Bruna et al. [10] generalize the convolution operation from Euclidean data to non-Euclidean data by using the Fourier basis of a given graph. To simplify spectral GNNs, Defferrard et al. [24] propose ChebNet and utilize Chebyshev polynomials as the convolution filter. Kipf et al. [78] propose GCN and simplify ChebNet by using its first-order approximation. Further, Simple Graph Convolution (SGC) [151] reduces the graph convolution to a linear model but still achieves competitive performance. The second family of models define graph convolutions in the spatial domain as aggregating and transforming local information [58, 50, 139]. For instance, DCNN [3] treats graph convolutions as a diffusion process and assigns a certain transition probability for information transferred from one node to the adjacent node. Hamilton et al. [58] propose to learn aggregators by sampling and aggregating neighbor information. Veličković et al. [139] propose graph attention network (GAT) to learn different attention scores for neighbors when aggregating information. To further improve the training efficiency, FastGCN [16] interprets graph convolutions as integral transforms of embedding functions under probability measures and performs importance sampling to sample a fixed number of nodes for each layer. For a thorough review, we please refer the reader to recent surveys [195, 156, 5].

4.2.2 Adversarial Attacks and Defense for GNNs

Extensive studies have demonstrated that deep learning models are vulnerable to adversarial attacks. In other words, slight or unnoticeable perturbations to the input can fool a neural network to output a wrong prediction. GNNs also suffer this problem [67, 209, 22, 211, 105, 98, 8, 152]. Different from image data, the graph structure is discrete and the nodes are dependent on each other, thus making it far more challenging. The *nettack* [209] generates unnoticeable perturbations by preserving degree distribution and imposing constraints on feature co-occurrence. RL-S2V [22] employs reinforcement learning to generate adversarial attacks. However, both of the two methods are

designed for targeted attack and can only degrade the performance of GNN on target nodes. To perturb the graph globally, *metattack* [211] is proposed to generate poisoning attacks based on meta-learning. Although increasing efforts have been devoted to developing adversarial attacks on graph data, the research about improving the robustness of GNNs has just started recently [200, 152, 133, 212]. One way to solve the problem is to learn a robust network by penalizing the attention scores of adversarial edges. RGCN [200] is to model Gaussian distributions as hidden layers to absorb the effects of adversarial attacks in the variances. PA-GNN [133] leverages supervision knowledge from clean graphs and applies a meta-optimization way to learn attention scores for robust graph neural networks. However, it requires additional graph data from similar domain. The other way is to preprocess the perturbed graphs to get clean graphs and train GNNs on the clean ones. Wu et. al [152] have found that attackers tend to connect to nodes with different features and they propose to remove the links between dissimilar nodes. Entezari et al. [36] have observed that *nettack* results in changes in high-rank spectrum of the graph and propose to preprocess the graph with its low-rank approximations. However, due to the simplicity of two-stage preprocessing methods, they may fail to counteract complex global attacks.

Different from the aforementioned defense methods, we aim to explore important graph properties to recover the clean graph while learning the GNN parameters simultaneously, which enables the proposed model to extract intrinsic structure from perturbed graph under different attacks.

4.3 Problem Statement

Before we present the problem statement, we first introduce some notations and basic concepts. The Frobenius norm of a matrix $\mathbf{S}$ is defined by $||\mathbf{S}||_F^2 = \Sigma_{ij}\mathbf{S}_{ij}^2$. The ℓ_1 norm of a matrix $\mathbf{S}$ is given by $||\mathbf{S}||_1 = \Sigma_{ij}|\mathbf{S}_{ij}|$ and the nuclear norm of a matrix $\mathbf{S}$ is defined as $||\mathbf{S}||_* = \Sigma_{i=1}^{rank(\mathbf{S})}\sigma_i$, where σ_i is the i-th singular value of $\mathbf{S}$. $(\mathbf{S})_+$ denotes the element-wise positive part of matrix $\mathbf{S}$ where $\mathbf{S}_{ij} = \max\{\mathbf{S}_{ij}, 0\}$ and $sgn(\mathbf{S})$ indicates the sign matrix of $\mathbf{S}$ where $sgn(\mathbf{S})_{ij} = 1, 0$, or -1 if $\mathbf{S}_{ij}$ >0, =0, or <0, respectively. We use $\odot$ to denote Hadamard product of matrices. Finally, we use $tr(\mathbf{S})$ to indicate the trace of matrix $\mathbf{S}$, i.e., $tr(\mathbf{S}) = \sum_i \mathbf{S}_{ii}$.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, where $\mathcal{V}$ is the set of N nodes $\{v_1, v_2, ..., v_N\}$ and $\mathcal{E}$ is the set of

edges. The edges describe the relations between nodes and can also be represented by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ where $\mathbf{A}_{ij}$ denotes the relation between nodes v_i and v_j. Furthermore, we use $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N] \in \mathbb{R}^{N\times d}$ to denote the node feature matrix where $\mathbf{x}_i$ is the feature vector of the node v_i. Thus a graph can also be denoted as $\mathcal{G} = (\mathbf{A}, \mathbf{X})$. Following the common node classification setting, only a part of nodes $\mathcal{V}_L = \{v_1, v_2, ..., v_l\}$ are associated with corresponding labels $\mathcal{Y}_L = \{y_1, y_2, ..., y_l\}$ where y_i denotes the label of v_i.

Given a graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$ and the partial labels $\mathcal{Y}_L$, the goal of node classification for GNN is to learn a function $f_\theta : \mathcal{V}_L \rightarrow \mathcal{Y}_L$ that maps the nodes to the set of labels so that f_θ can predict labels of unlabeled nodes. The objective function can be formulated as

$$\min_{\theta} \mathcal{L}_{GNN}(\theta, \mathbf{A}, \mathbf{X}, \mathcal{Y}_L) = \sum_{v_i \in \mathcal{V}_L} \ell\left(f_\theta(\mathbf{X}, \mathbf{A})_i, y_i\right), \tag{4.1}$$

where θ is the parameters of f_θ, $f_\theta(\mathbf{X}, \mathbf{A})_i$ is the prediction of node v_i and $\ell(\cdot, \cdot)$ is to measure the difference between prediction and true label such as cross entropy. Though there exist a number of different GNN methods, in this work, we focus on Graph Convolution Network (GCN) in [78]. Note that it is straightforward to extend the proposed framework to other GNN models. Specifically, a two-layer GCN with $\theta = (\mathbf{W}_1, \mathbf{W}_2)$ implements f_θ as

$$f_\theta(\mathbf{X}, \mathbf{A}) = \text{softmax}\left(\hat{\mathbf{A}}\ \sigma\left(\hat{\mathbf{A}}\,\mathbf{X}\,\mathbf{W}_1\right)\mathbf{W}_2\right), \tag{4.2}$$

where $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}(\mathbf{A} + \mathbf{I})\tilde{\mathbf{D}}^{-1/2}$ and $\tilde{\mathbf{D}}$ is the diagonal matrix of $\mathbf{A} + \mathbf{I}$ with $\tilde{\mathbf{D}}_{ii} = 1 + \sum_j \mathbf{A}_{ij}$. σ is the activation function such as ReLU.

With aforementioned notations and definitions, the problem we aim to study in this work can be formally stated as:

Given $\mathcal{G} = (\mathbf{A}, X)$ and partial node label $\mathcal{V}_L$ with $\mathbf{A}$ being poisoned by adversarial edges and feature matrix $\mathbf{X}$ unperturbed, simultaneously learn a clean graph structure with the graph adjacency matrix $\mathbf{S} \in \mathcal{S} = [0, 1]^{N\times N}$ and the GNN parameters θ to improve node classification performance for unlabeled nodes.

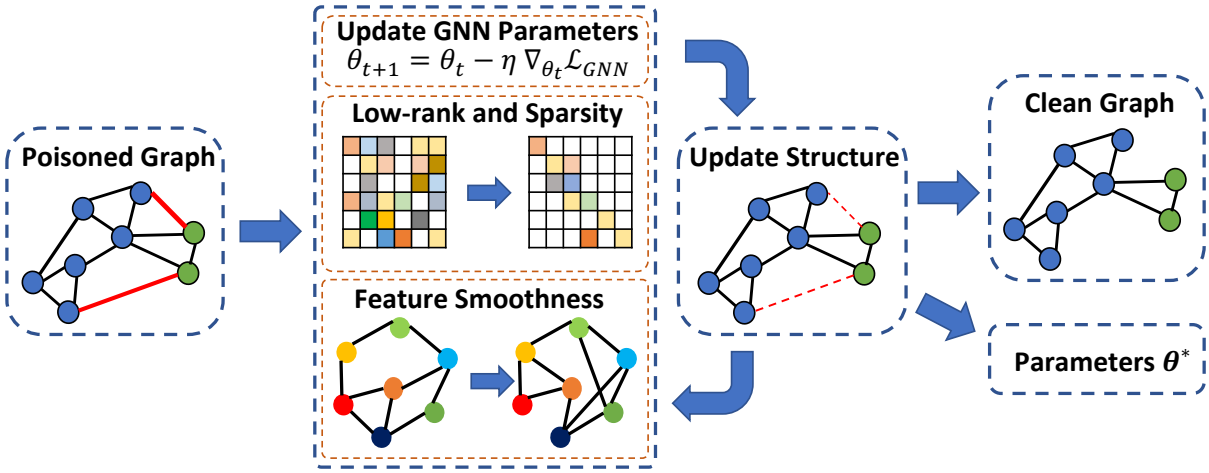


Figure 4.2: Overall framework of Pro-GNN. Dash lines indicate smaller weights.

4.4 The Proposed Framework

Adversarial attacks generate carefully-crafted perturbation on graph data. We refer to the carefully-crafted perturbation as adversarial structure. Adversarial structure can cause the performance of GNNs to drop rapidly. Thus, to defend adversarial attacks, one natural strategy is to eliminate the crafted adversarial structure, while maintaining the intrinsic graph structure. In this work, we aim to achieve the goal by exploring graph structure properties of low rank, sparsity and feature smoothness. The illustration of the framework is shown in Figure 4.2, where edges in black are normal edges and edges in red are adversarial edges introduced by an attacker to reduce the node classification performance. To defend against the attacks, Pro-GNN iteratively reconstructs the clean graph by preserving the low rank, sparsity, and feature smoothness properties of a graph so as to reduce the negative effects of adversarial structure. Meanwhile, to make sure that the reconstructed graph can help node classification, Pro-GNN simultaneously updates the GNN parameters on the reconstructed graph by solving the optimization problem in an alternating schema. In the following subsections, we will give the details of the proposed framework.

4.4.1 Exploring Low rank and Sparsity Properties

Many real-world graphs are naturally low-rank and sparse as the entities usually tend to form communities and would only be connected with a small number of neighbors [199]. Adversarial attacks on GCNs tend to add adversarial edges that link nodes of different communities as this is

more efficient to reduce node classification performance of GCN. Introducing links connecting nodes of different communities in a sparse graph can significantly increase the rank of the adjacency matrix and enlarge the singular values, thus damaging the low rank and sparsity properties of graphs, which is verified in Figure 4.1a and Figure 4.1b. Thus, to recover the clean graph structure from the noisy and perturbed graph, one potential way is to learn a clean adjacency matrix $\mathbf{S}$ close to the adjacency matrix of the poisoned graph by enforcing the new adjacency matrix with the properties of low rank and sparsity. As demonstrated in Figure4.1c, the rank decreases much faster by removing adversarial edges than by removing normal edges. This implies that the low rank and sparsity constraint can remove the adversarial edges instead of normal edges. Given the adjacency matrix $\mathbf{A}$ of a poisoned graph, we can formulate the above process as a structure learning problem [124, 64]:

$$\arg\min_{\mathbf{S}\in\mathcal{S}} \mathcal{L}_0 = \|\mathbf{A}-\mathbf{S}\|_F^2 + R(\mathbf{S}), \quad s.t., \mathbf{S} = \mathbf{S}^\top. \tag{4.3}$$

Since adversarial attacks target on performing unnoticeable perturbations to graphs, the first term $\|\mathbf{A}-\mathbf{S}\|_F^2$ ensures that the new adjacency matrix $\mathbf{S}$ should be close to $\mathbf{A}$. As we assume that the graph are undirected, the new adjacency matrix should be symmetric, i.e., $\mathbf{S} = \mathbf{S}^\top$. $R(\mathbf{S})$ denotes the constraints on $\mathbf{S}$ to enforce the properties of low rank and sparsity. According to [13, 80, 124], minimizing the ℓ_1 norm and the nuclear norm of a matrix can force the matrix to be sparse and low-rank, respectively. Hence, to ensure a sparse and low-rank graph, we want to minimize the ℓ_1 norm and the nuclear norm of $\mathbf{S}$. Eq. (4.3) can be rewritten as:

$$\arg\min_{\mathbf{S}\in\mathcal{S}} \mathcal{L}_0 = \|\mathbf{A}-\mathbf{S}\|_F^2 + \alpha\|\mathbf{S}\|_1 + \beta\|\mathbf{S}\|_*, \quad s.t., \mathbf{S} = \mathbf{S}^\top, \tag{4.4}$$

where α and β are predefined parameters that control the contributions of the properties of sparsity and low rank, respectively. One important benefit to minimize the nuclear norm $\|\mathbf{S}\|_*$ is that we can reduce every singular value, thus alleviating the impact of enlarging singular values from adversarial attacks.

4.4.2 Exploring Feature Smoothness

It is evident that connected nodes in a graph are likely to share similar features. In fact, this observation has been made on graphs from numerous domains. For example, two connected users

in a social graph are likely to share similar attributes [108], two linked web pages in the webpage graph tend to have similar contents [138] and two connected papers in the citation network usually have similar topics [78]. Meanwhile, recently it is demonstrated that adversarial attacks on graphs tend to connect nodes with distinct features [152]. Thus, we aim to ensure the feature smoothness in the learned graph. The feature smoothness can be captured by the following term $\mathcal{L}_s$:

$$\mathcal{L}_s = \frac{1}{2}\sum_{i,j=1}^{N} \mathbf{S}_{ij}(\mathbf{x}_i - \mathbf{x}_j)^2, \tag{4.5}$$

where $\mathbf{S}$ is the new adjacency matrix, $\mathbf{S}_{ij}$ indicates the connection of v_i and v_j in the learned graph and $(\mathbf{x}_i - \mathbf{x}_j)^2$ measures the feature difference between v_i and v_j. $\mathcal{L}_s$ can be rewritten as:

$$\mathcal{L}_s = tr(\mathbf{X}^\top \mathbf{L}\mathbf{X}), \tag{4.6}$$

where $\mathbf{L} = \mathbf{D} - \mathbf{S}$ is the graph Laplacian matrix of $\mathbf{S}$ and $\mathbf{D}$ is the diagonal matrix of $\mathbf{S}$. In this work, we use normalized Laplacian matrix $\hat{\mathbf{L}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$ instead of $\mathbf{L}$ to make feature smoothness independent on the degrees of the graph nodes [2], i.e.,

$$\mathcal{L}_s = tr(\mathbf{X}^T\hat{\mathbf{L}}\mathbf{X}) = \frac{1}{2}\sum_{i,j=1}^{N} \mathbf{S}_{ij}(\frac{\mathbf{x}_i}{\sqrt{d_i}} - \frac{\mathbf{x}_j}{\sqrt{d_j}})^2, \tag{4.7}$$

where d_i denotes the degree of v_i in the learned graph. In the learned graph, if v_i and v_j are connected (i.e., $\mathbf{S}_{ij} \neq 0$), we expect that the feature difference $(\mathbf{x}_i - \mathbf{x}_j)^2$ should be small. In other words, if the features between two connected node are quite different, $\mathcal{L}_s$ would be very large. Therefore, the smaller $\mathcal{L}_s$ is, the smoother features $\mathbf{X}$ are on the graph $\mathbf{S}$. Thus, to fulfill the feature smoothness in the learned graph, we should minimize $\mathcal{L}_s$. Therefore, we can add the feature smoothness term to the objective function of Eq. (4.4) to penalize rapid changes in features between adjacent nodes as:

$$\arg\min_{\mathbf{S}\in\mathcal{S}} \mathcal{L} = \mathcal{L}_0 + \lambda \cdot \mathcal{L}_s = \mathcal{L}_0 + \lambda\, tr(\mathbf{X}^T\hat{\mathbf{L}}\mathbf{X}),\ s.t.,\ \mathbf{S} = \mathbf{S}^\top, \tag{4.8}$$

where λ is a predefined parameter to control the contribution from feature smoothness.

4.4.3 Objective Function of Pro-GNN

Intuitively, we can follow the preprocessing strategy [152, 36] to defend against adversarial attacks – we first learn a graph from the poisoned graph via Eq. (4.8) and then train a GNN model

based on the learned graph. However, with such a two-stage strategy, the learned graph may be suboptimal for the GNN model on the given task. Thus, we propose a better strategy to jointly learn the graph structure and the GNN model for a specific downstream task. We empirically show that jointly learning GNN model and the adjacency matrix is better than two stage one in Sec 4.5.4.2. The final objective function of Pro-GNN is given as

$$\begin{aligned} &\arg\min_{\mathbf{S}\in\mathcal{S},\theta} \mathcal{L} = \mathcal{L}_0 + \lambda\mathcal{L}_s + \gamma\mathcal{L}_{GNN} \\ &= \|\mathbf{A}-\mathbf{S}\|_F^2 + \alpha\|\mathbf{S}\|_1 + \beta\|\mathbf{S}\|_* + \gamma\mathcal{L}_{GNN}(\theta,\mathbf{S},\mathbf{X},\mathcal{Y}_L) + \lambda tr(\mathbf{X}^T\hat{\mathbf{L}}\mathbf{X}) \\ &\quad s.t. \qquad \mathbf{S} = \mathbf{S}^\top, \end{aligned} \tag{4.9}$$

where $\mathcal{L}_{GNN}$ is a loss function for the GNN model that is controlled by a predefined parameter γ. Another benefit of this formulation is that the information from $\mathcal{L}_{GNN}$ can also guide the graph learning process to defend against adversarial attacks since the goal of graph adversarial attacks is to maximize $\mathcal{L}_{GNN}$.

4.4.4 An Optimization Algorithm

Jointly optimizing θ and $\mathbf{S}$ in Eq.(4.9) is challenging. The constraints on $\mathbf{S}$ further exacerbate the difficulty. Thus, in this work, we use an alternating optimization schema to iteratively update θ and $\mathbf{S}$.

Update $\boldsymbol{\theta}$. To update θ, we fix $\mathbf{S}$ and remove terms that are irrelevant to θ, then the objective function in Eq.(4.9) reduces to:

$$\min_{\theta} \mathcal{L}_{GNN}(\theta,\mathbf{S},\mathbf{X},\mathcal{Y}_L) = \sum_{u\in\mathcal{V}_L} \ell\left(f_\theta(\mathbf{X},\mathbf{S})_u, y_u\right), \tag{4.10}$$

which is a typical GNN optimization problem and we can learn θ via stochastic gradient descent.

Update S. Similarly, to update $\mathbf{S}$, we fix θ and arrive at

$$\min_{\mathbf{S}} \mathcal{L}(\mathbf{S},A) + \alpha\|\mathbf{S}\|_1 + \beta\|\mathbf{S}\|_* \quad s.t., \quad \mathbf{S} = \mathbf{S}^\top, \mathbf{S}\in\mathcal{S}, \tag{4.11}$$

where $\mathcal{L}(\mathbf{S},A)$ is defined as

$$\mathcal{L}(\mathbf{S},A) = \|\mathbf{A}-\mathbf{S}\|_F^2 + \mathcal{L}_{GNN}(\theta,\mathbf{S},\mathbf{X},Y) + \lambda tr(\mathbf{X}^T\hat{\mathbf{L}}\mathbf{X}). \tag{4.12}$$

Note that both ℓ_1 norm and nuclear norm are non-differentiable. For optimization problem with only one non-diffiential regularizer $R(S)$, we can use Forward-Backward splitting methods [20]. The idea is to alternate a gradient descent step and a proximal step as:

$$\mathbf{S}^{(k)} = \text{prox}_{\eta R}\left(\mathbf{S}^{(k-1)} - \eta \nabla_S \mathcal{L}(S, A)\right), \tag{4.13}$$

where η is the learning rate, prox_R is the proximal operator as:

$$\text{prox}_R(\mathbf{Z}) = \underset{\mathbf{S}\in\mathbb{R}^{N\times N}}{\arg\min} \frac{1}{2}\|\mathbf{S} - \mathbf{Z}\|_F^2 + R(\mathbf{S}). \tag{4.14}$$

In particular, the proximal operator of ℓ_1 norm and nuclear norm can be represented as [124, 6],

$$\text{prox}_{\alpha||.||_1}(\mathbf{Z}) = sgn(\mathbf{Z}) \odot (|\mathbf{Z}| - \alpha)_+, \tag{4.15}$$

$$\text{prox}_{\beta||.||_*}(\mathbf{Z}) = \mathbf{U}\, diag((\sigma_i - \beta)_+)_i \mathbf{V}^T, \tag{4.16}$$

where $\mathbf{Z} = \mathbf{U}\, diag(\sigma_1, ..., \sigma_n)\mathbf{V}^\top$ is the singular value decomposition of $\mathbf{Z}$. To optimize objective function with two non-differentiable regularizers, Richard et al. [117] introduce the Incremental Proximal Descent method based on the introduced proximal operators. By iterating the updating process in a cyclic manner, we can update $\mathbf{S}$ as follows,

$$\begin{cases} \mathbf{S}^{(k)} = \mathbf{S}^{(k-1)} - \eta \cdot \nabla_{\mathbf{S}} \left(\mathcal{L}(\mathbf{S}, \mathbf{A})\right), \\ \mathbf{S}^{(k)} = \text{prox}_{\eta\beta\|\cdot\|_*}\left(\mathbf{S}^{(k)}\right), \\ \mathbf{S}^{(k)} = \text{prox}_{\eta\alpha\|\cdot\|_1}\left(\mathbf{S}^{(k)}\right). \end{cases} \tag{4.17}$$

After we learn a relaxed $\mathbf{S}$, we project $\mathbf{S}$ to satisfy the constraints. For the symmetric constraint, we let $\mathbf{S} = \frac{\mathbf{S}+\mathbf{S}^\top}{2}$. For the constraint $\mathbf{S}_{ij} \in [0, 1]$, we project $\mathbf{S}_{ij} < 0$ to 0 and $\mathbf{S}_{ij} > 1$ to 1. We denote these projection operations as $P_{\mathcal{S}}(\mathbf{S})$.

Training Algorithm. With these updating and projection rules, the optimization algorithm is shown in Algorithm 1. In line 1, we first initialize the estimated graph $\mathbf{S}$ as the poisoned graph $\mathbf{A}$. In line 2, we randomly initialize the GNN parameters. From lines 3 to 10, we update $\mathbf{S}$ and the GNN parameters θ alternatively and iteratively. Specifically, we train the GNN parameters in each iteration while training the graph reconstruction model every τ iterations.

Algorithm 1 Pro-GNN

Data: Adjacency matrix $\mathbf{A}$, Attribute matrix $\mathbf{X}$, Labels $\mathcal{Y}_L$, Hyper-parameters $\alpha, \beta, \gamma, \lambda, \tau$, Learning rate η, η'

Result: Learned adjacency $\mathbf{S}$, GNN parameters θ

Initialize $\mathbf{S} \leftarrow \mathbf{A}$

Randomly initialize θ

while *Stopping condition is not met* **do**

  $\mathbf{S} \leftarrow \mathbf{S} - \eta \nabla_S (\|\mathbf{S} - \mathbf{A}\|_F^2 + \gamma \mathcal{L}_{GNN} + \lambda \mathcal{L}_s)$

  $\mathbf{S} \leftarrow \text{prox}_{\eta\beta||.||_*}(\mathbf{S})$

  $\mathbf{S} \leftarrow \text{prox}_{\eta\alpha||.||_1}(\mathbf{S})$

  $\mathbf{S} \leftarrow P_{\mathcal{S}}(\mathbf{S})$

  for *i=1 to* τ **do**

    $g \leftarrow \frac{\partial \mathcal{L}_{GNN}(\theta, \mathbf{S}, \mathbf{X}, \mathcal{Y}_L)}{\partial \theta}$

    $\theta \leftarrow \theta - \eta' g$

  end

end

Return $\mathbf{S}, \theta$

4.5 Experiments

In this section, we evaluate the effectiveness of Pro-GNN against different graph adversarial attacks. In particular, we aim to answer the following questions:

- **RQ1** How does Pro-GNN perform compared to the state-of-the-art defense methods under different adversarial attacks?
- **RQ2** Does the learned graph work as expected?
- **RQ3** How do different properties affect the performance of Pro-GNN.

Before presenting our experimental results and observations, we first introduce the experimental settings.

4.5.1 Experimental settings

4.5.1.1 Datasets

Following [209, 211], we validate the proposed approach on four benchmark datasets, including three citation graphs, i.e., Cora, Citeseer and Pubmed, and one blog graph, i.e., Polblogs. The statistics of the datasets are shown in Table 4.1. Note that in the Polblogs graph, node features are not available. In this case, we set the attribute matrix to $N \times N$ identity matrix.

Table 4.1: Dataset Statistics. Following [209, 211, 36], we only consider the largest connected component (LCC).

	$\mathbf{N_{LCC}}$	$\mathbf{E_{LCC}}$	**Classes**	**Features**
Cora	2,485	5,069	7	1,433
Citeseer	2,110	3,668	6	3,703
Polblogs	1,222	16,714	2	/
Pubmed	19,717	44,338	3	500

Table 4.2: Node classification performance (Accuracy±Std) under non-targeted attack (*metattack*).

Dataset	Ptb Rate (%)	GCN	GAT	RGCN	GCN-Jaccard[1]	GCN-SVD	Pro-GNN-fs	Pro-GNN[2]
Cora	0	83.50±0.44	**83.97±0.65**	83.09±0.44	82.05±0.51	80.63±0.45	83.42±0.52	82.98±0.23
	5	76.55±0.79	80.44±0.74	77.42±0.39	79.13±0.59	78.39±0.54	**82.78±0.39**	82.27±0.45
	10	70.39±1.28	75.61±0.59	72.22±0.38	75.16±0.76	71.47±0.83	77.91±0.86	**79.03±0.59**
	15	65.10±0.71	69.78±1.28	66.82±0.39	71.03±0.64	66.69±1.18	76.01±1.12	**76.40±1.27**
	20	59.56±2.72	59.94±0.92	59.27±0.37	65.71±0.89	58.94±1.13	68.78±5.84	**73.32±1.56**
	25	47.53±1.96	54.78±0.74	50.51±0.78	60.82±1.08	52.06±1.19	56.54±2.58	**69.72±1.69**
Citeseer	0	71.96±0.55	73.26±0.83	71.20±0.83	72.10±0.63	70.65±0.32	73.26±0.38	**73.28±0.69**
	5	70.88±0.62	72.89±0.83	70.50±0.43	70.51±0.97	68.84±0.72	**73.09±0.34**	72.93±0.57
	10	67.55±0.89	70.63±0.48	67.71±0.30	69.54±0.56	68.87±0.62	72.43±0.52	**72.51±0.75**
	15	64.52±1.11	69.02±1.09	65.69±0.37	65.95±0.94	63.26±0.96	70.82±0.87	**72.03±1.11**
	20	62.03±3.49	61.04±1.52	62.49±1.22	59.30±1.40	58.55±1.09	66.19±2.38	**70.02±2.28**
	25	56.94±2.09	61.85±1.12	55.35±0.66	59.89±1.47	57.18±1.87	66.40±2.57	**68.95±2.78**
Polblogs	0	**95.69±0.38**	95.35±0.20	95.22±0.14	-	95.31±0.18	93.20±0.64	-
	5	73.07±0.80	83.69±1.45	74.34±0.19	-	89.09±0.22	**93.29±0.18**	-
	10	70.72±1.13	76.32±0.85	71.04±0.34	-	81.24±0.49	**89.42±1.09**	-
	15	64.96±1.91	68.80±1.14	67.28±0.38	-	68.10±3.73	**86.04±2.21**	-
	20	51.27±1.23	51.50±1.63	59.89±0.34	-	57.33±3.15	**79.56±5.68**	-
	25	49.23±1.36	51.19±1.49	56.02±0.56	-	48.66±9.93	**63.18±4.40**	-
Pubmed	0	87.19±0.09	83.73±0.40	86.16±0.18	87.06±0.06	83.44±0.21	**87.33±0.18**	87.26±0.23
	5	83.09±0.13	78.00±0.44	81.08±0.20	86.39±0.06	83.41±0.15	**87.25±0.09**	87.23±0.13
	10	81.21±0.09	74.93±0.38	77.51±0.27	85.70±0.07	83.27±0.21	**87.25±0.09**	87.21±0.13
	15	78.66±0.12	71.13±0.51	73.91±0.25	84.76±0.08	83.10±0.18	**87.20±0.09**	87.20±0.15
	20	77.35±0.19	68.21±0.96	71.18±0.31	83.88±0.05	83.01±0.22	87.09±0.10	**87.15±0.15**
	25	75.50±0.17	65.41±0.77	67.95±0.15	83.66±0.06	82.72±0.18	86.71±0.09	**86.76±0.19**

[1] [2] JaccardGCN and Pro-GNN cannot be directly applied to datasets where node features are not available.

4.5.1.2 Baselines

To evaluate the effectiveness of Pro-GNN, we compare it with the state-of-the-art GNN and defense models by using the adversarial attack repository DeepRobust [85]:

- **GCN** [78]: while there exist a number of different Graph Convolutional Networks (GCN) models, we focus on the most representative one [78].
- **GAT** [139]: Graph Attention Netowork (GAT) is composed of attention layers which can learn different weights to different nodes in the neighborhood. It is often used as a baseline to defend against adversarial attacks.

- **RGCN** [200]: RGCN models node representations as gaussian distributions to absorb effects of adversarial attacks. It also employs attention mechanism to penalize nodes with high variance.
- **GCN-Jaccard** [152]: Since attackers tend to connect nodes with dissimilar features or different labels, GCN-Jaccard preprocesses the network by eliminating edges that connect nodes with jaccard similarity of features smaller than threshold τ. Note that this method only works when node features are available.
- **GCN-SVD** [36]: This is another preprocessing method to resist adversarial attacks. It is noted that *nettack* is a high-rank attack, thus GCN-SVD proposes to vaccinate GCN with the low-rank approximation of the perturbed graph. Note that it originally targets at defending against *nettack*, however, it is straightforward to extend it to non-targeted and random attacks.

In addition to representative baselines, we also include one variant of the proposed framework, Pro-GNN-fs, which is the variant by eliminating the feature smoothness term (or setting $\lambda = 0$).

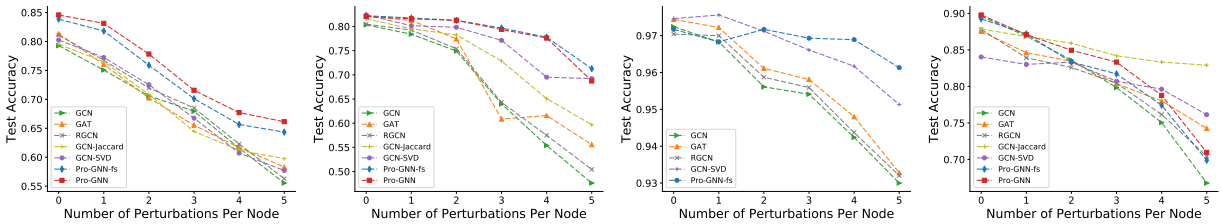
4.5.1.3 Parameter Settings

For each graph, we randomly choose 10% of nodes for training, 10% of nodes for validation and the remaining 80% of nodes for testing. For each experiment, we report the average performance of 10 runs. The hyper-parameters of all the models are tuned based on the loss and accuracy on validation set. For GCN and GAT, we adopt the default parameter setting in the author's implementation. For RGCN, the number of hidden units are tuned from $\{16, 32, 64, 128\}$. For GCN-Jaccard, the threshold of similarity for removing dissimilar edges is chosen from $\{0.01, 0.02, 0.03, 0.04, 0.05, 0.1\}$. For GCN-SVD , the reduced rank of the perturbed graph is tuned from $\{5, 10, 15, 50, 100, 200\}$.

4.5.2 Defense Performance

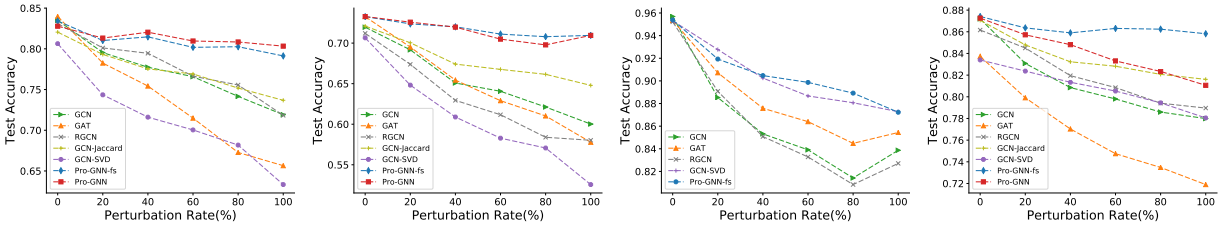
To answer the first question, we evaluate the node classification performance of Pro-GNN against three types of attacks, i.e., non-targeted attack, targeted attack and random attack:

- **Targeted Attack:** Targeted attack generates attacks on specific nodes and aims to fool GNNs on these target nodes. We adopt *nettack* [209] for the targeted attack method, which is the state-of-the-art targeted attack on graph data.
- **Non-targeted Attack:** Different from targeted attack, the goal of non-targeted attack is to degrade



(a) Cora (b) Citeseer (c) Polblogs (d) Pubmed

Figure 4.3: Results of different models under *nettack*



(a) Cora (b) Citeseer (c) Polblogs (d) Pubmed

Figure 4.4: Results of different models under random attack

the overall performance of GNNs on the whole graph. We adopt one representative non-targeted attack, *metattack* [211] .

- **Random Attack:** It randomly injects fake edges into the graph. It can also be viewed as adding random noise to the clean graph.

We first use the attack method to poison the graph. We then train Pro-GNN and baselines on the poisoned graph and evaluate the node classification performance achieved by these methods.

4.5.2.1 Against Non-targeted Adversarial Attacks

We first evaluate the node classification accuracy of different methods against non-targeted adversarial attack. Specifically, we adopt *metattack* and keep all the default parameter settings in the authors' original implementation. The *metattack* has several variants. For Cora, Citeseer and Polblogs datasets, we apply Meta-Self since it is the most destructive attack variant; while for Pubmed, the approximate version of Meta-Self, A-Meta-Self is applied to save memory and time. We vary the perturbation rate, i.e., the ratio of changed edges, from 0 to 25% with a step of 5%. As mentioned before, all the experiments are conducted 10 times and we report the average accuracy with standard deviation in Table 4.2. The best performance is highlighted in bold. From the table,

we make the following observations:

- Our method consistently outperforms other methods under different perturbation rates. For instance, on Polblogs dataset our model improves GCN over 20% at 5% perturbation rate. Even under large perturbation, our method outperforms other baselines by a larger margin. Specifically, under the 25% perturbation rate on the three datasets, vanilla GCN performs very poorly and our model improves GCN by 22%, 12% and 14%, respectively.
- Although GCN-SVD also employs SVD to get low-rank approximation of the graph, the performance of GCN-SVD drops rapidly. This is because GCN-SVD is designed for targeted attack, it cannot adapt well to the non-targeted adversarial attack. Similarly, GCN-Jaccard does not perform as well as Pro-GNN under different perturbation rates. This is because simply preprocessing the perturbed graph once cannot recover the complex intrinsic graph structure from the carefully-crafted adversarial noises. On the contrary, simultaneously updating the graph structure and GNN parameters with the low rank, sparsity and feature smoothness constraints helps recover better graph structure and learn robust GNN parameters.
- Pro-GNN achieves higher accuracy than Pro-GNN-fs especially when the perturbation rate is large, which demonstrates the effectiveness of feature smoothing in removing adversarial edges.

4.5.2.2 Against Targeted Adversarial Attack

In this experiment, *nettack* is adopted as the targeted-attack method and we use the default parameter settings in the authors' original implementation. Following [200], we vary the number of perturbations made on every targeted node from 1 to 5 with a step size of 1. The nodes in test set with degree larger than 10 are set as target nodes. For Pubmed dataset, we only sample 10% of them to reduce the running time of *nettack* while in other datasets we use all the target nodes. The node classification accuracy on target nodes is shown in Figure 4.3. From the figure, we can observe that when the number of perturbation increases, the performance of our method is better than other methods on the attacked target nodes in most cases. For instance, on Citeseer dataset at 5 perturbation per targeted node, our model improves vanilla GCN by 23% and outperforms other defense methods by 11%. It demonstrates that our method can also resist the targeted adversarial

attack.

4.5.2.3 Against Random Attack

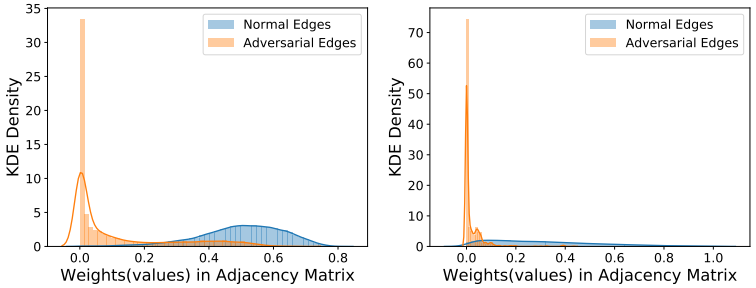
In this subsection, we evaluate how Pro-GNN behaves under different ratios of random noises from 0% to 100% with a step size of 20%. The results are reported in Figure 4.4. The figure shows that Pro-GNN consistently outperforms all other baselines and successfully resists random attack. Together with observations from Sections 4.5.2.1 and 4.5.2.2, we can conclude that Pro-GNN is able to defend various types of adversarial attacks. This is a desired property in practice since attackers can adopt any kinds of attacks to fool the system.

4.5.3 Importance of Graph Structure Learning

In the previous subsection, we have demonstrated the effectiveness of the proposed framework. In this section, we aim to understand the graph we learned and answer the second question.

4.5.3.1 Normal Edges Against Adversarial Edges

Based on the fact that adversary tends to add edges over delete edges [152, 211], if the model tends to learn a clean graph structure, the impact of the adversarial edges should be mitigated from the poisoned graph. Thus, we investigate the weights of normal and adversarial edges in the learned adjacency matrix $\mathbf{S}$. We visualize the weight density distribution of normal and perturbed edges of $\mathbf{S}$ in Figure 4.5. Due to the limit of space, we only show results on Pubmed and Polblogs under *metattack*. As we can see in the figure, in both datasets, the weights of adversarial edges are much smaller than those of normal edges, which shows that Pro-GNN can alleviate the effect of adversarial



(a) Pubmed (b) Polblogs

Figure 4.5: Weight density distributions of normal and adversarial edges on the learned graph.

Table 4.3: Node classification accuracy given the graph under 25% perturbation by *metattack*.

	GCN	GCN-NoGraph	Pro-GNN
Cora	47.53±1.96	62.12±1.55	**69.72±1.69**
Citeseer	56.94±2.09	63.75±3.23	**68.95±2.78**
Polblogs	49.23±1.36	51.79±0.62	**63.18±4.40**
Pubmed	75.50±0.17	84.14±0.11	**86.86±0.19**

edges and thus learn robust GNN parameters.

4.5.3.2 Performance on Heavily Poisoned Graph

In this subsection, we study the performance when the graph is heavily poisoned. In particular, we poison the graph with 25% perturbation by *metattack* . If a graph is heavily poisoned, the performance of GCN will degrade a lot. One straightforward solution is to remove the poisoned graph structure. Specifically, when removing the graph structure, the adjacency matrix will be all zeros and GCN normalizes the zero matrix into identity matrix and then makes prediction totally by node features. Under this circumstance, GCN actually becomes a feed-forward neural network. We denote it as GCN-NoGraph. We report the performance of GCN, GCN-NoGraph and Pro-GNN when the graph is heavily poisoned in Table 4.3.

From the table, we first observe that when the graph structure is heavily poisoned, by removing the graph structure, GCN-NoGraph outperforms GCN. This observation suggests the necessity to defend poisoning attacks on graphs because the poisoned graph structure are useless or even hurt the prediction performance. We also note that Pro-GNN obtains much better results than GCN-NoGraph. This observation suggests that Pro-GNN can learn useful graph structural information even when the graph is heavily poisoned.

4.5.4 Ablation Study

To get a better understanding of how different components help our model defend against adversarial attacks, we conduct ablation studies and answer the third question in this subsection.

Table 4.4: Classification performance of Pro-GNN-two and Pro-GNN on Cora dataset

Ptb Rate (%)	0	5	10	15	20	25
Pro-GNN-two	73.31±0.71	73.70±1.02	73.69±0.81	75.38±1.10	73.22±1.08	**70.57±0.61**
Pro-GNN	**82.98±0.23**	**82.27±0.45**	**79.03±0.59**	**76.40±1.27**	**73.32±1.56**	69.72±1.69

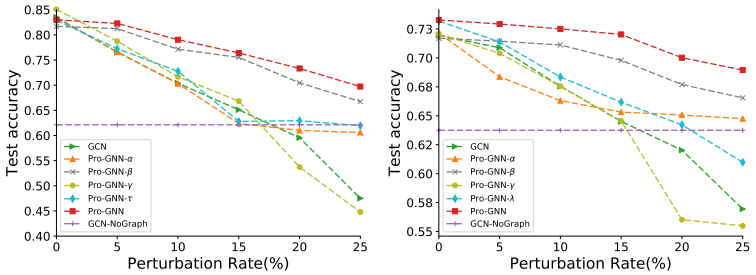
4.5.4.1 Regularizers

There are four key predefined parameters, i.e., α, β, γ and λ, which control the contributions for sparsity, low rank, GNN loss and feature smoothness, respectively. To understand the impact of each component, we vary the values of one parameter and set other parameters to zero, and then check how the performance changes. Correspondingly, four model variants are created: Pro-GNN-α, Pro-GNN-β, Pro-GNN-γ and Pro-GNN-λ. For example, Pro-GNN-α denotes that we vary the values of α while setting β, γ and λ to zero. We only report results on Cora and Citeseer, since similar patterns are observed in other cases, shown in Figure 4.6.

From the figure we can see Pro-GNN-α does not boost the model's performance too much with small perturbations. But when the perturbation becomes large, Pro-GNN-α outperforms vanilla GCN because it can learn a graph structure better than a heavily poisoned adjacency graph as shown in Section 4.5.3.2. Also, Pro-GNN-β and Pro-GNN-λ perform much better than vanilla GCN. It is worth noting that, Pro-GNN-β outperforms all other variants except Pro-GNN, indicating that nuclear norm is of great significance in reducing the impact of adversarial attacks. It is in line with our observation that adversarial attacks increase the rank of the graph and enlarge the singular values. Another observation from the figure is that, Pro-GNN-γ works better under small perturbation and when the perturbation rate increases, its performance degrades. From the above observations, different components play different roles in defending adversarial attacks. By incorporating these components, Pro-GNN can explore the graph properties and thus consistently outperform state-of-the-art baselines.

4.5.4.2 Two-Stage vs One-Stage

To study the contribution of jointly learning structure and GNN parameters, we conduct experiments with the variant Pro-GNN-two under *metattack* . Pro-GNN-two is the two stage variant



(a) Cora (b) Citeseer

Figure 4.6: Classification performance of Pro-GNN variants.

of Pro-GNN where we first obtain the clean graph and then train a GNN model based on it. We only show the results on Cora in Table 4.4 due to the page limitation. We can observe from the results that although Pro-GNN-two can achieve good performance under large perturbation, it fails to defend the attacks when the perturbation rate is relatively low. The results demonstrate that jointly learning structure and GNN parameters can actually help defend attacks.

4.5.5 Parameter Analysis

In this subsection, we explore the sensitivity of hyper-parameters α, β, γ and λ for Pro-GNN. In the experiments, we alter the value of α, β, γ and λ to see how they affect the performance of our model. More specifically, we vary α from 0.00025 to 0.064 in a log scale of base 2, β from 0 to 5, γ from 0.0625 to 16 in a log scale of base 2 and λ from 1.25 to 320 in a log scale of base 2. We only report the results on Cora dataset with the perturbation rate of 10% by *metattack* since similar observations are made in other settings.

The performance change of Pro-GNN is illustrated in Figure 4.7. As we can see, the accuracy of Pro-GNN can be boosted when choosing appropriate values for all the hyper-parameters. Different from γ, appropriate values of α and λ can boost the performance but large values will greatly hurt the performance. This is because focusing on sparsity and feature smoothness will result in inaccurate estimation on the graph structure. For example, if we set α and λ to $+\infty$, we will get a trivial solution of the new adjacency matrix, i.e, $\mathbf{S} = 0$. It is worth noting that, appropriate value of β can greatly increase the model's performance (more than 10%) compared with the variant without β, while too large or too small value of β will hurt the performance. This is also consistent with our observation

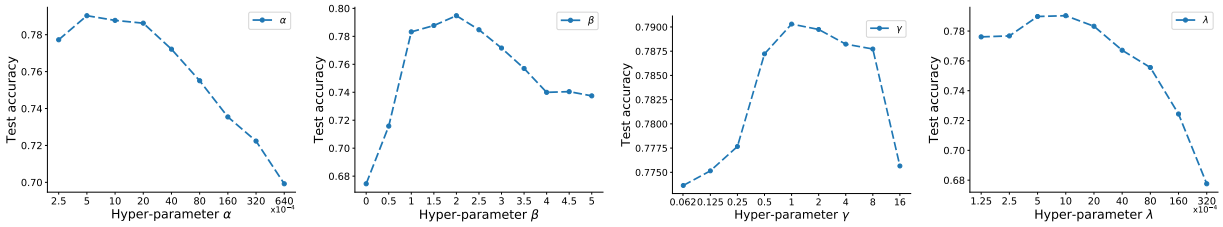


Figure 4.7: Results of parameter analysis on Cora dataset

in Section 4.5.4.1 that the low rank property plays an important role in defending adversarial attacks.

4.6 Conclusion

Graph neural networks can be easily fooled by graph adversarial attacks. To defend against different types of graph adversarial attacks, we introduced a novel defense approach Pro-GNN that learns the graph structure and the GNN parameters simultaneously. Our experiments show that our model consistently outperforms state-of-the-art baselines and improves the overall robustness under various adversarial attacks. In the future, we aim to explore more properties to further improve the robustness of GNNs.

CHAPTER 5

EMPOWERING GRAPH REPRESENTATION LEARNING WITH TEST-TIME GRAPH TRANSFORMATION

In this chapter, we study the test-time robustness of Graph Neural Networks (GNNs). The effectiveness of GNNs is immensely challenged by issues related to test data quality, such as distribution shifts, abnormal features and adversarial attacks. Recent efforts have been made on tackling these issues from a modeling perspective which requires additional cost of changing model architectures or re-training model parameters. In this work, we provide a data-centric view to tackle these issues and propose a graph transformation framework named GTrans which adapts and refines graph data at test time to achieve better performance. We provide theoretical analysis on the design of the framework and discuss why adapting graph data works better than adapting the model. Extensive experiments have demonstrated the effectiveness of GTrans on three distinct scenarios for eight benchmark datasets where suboptimal data is presented. Remarkably, GTrans performs the best in most cases with improvements up to 2.8%, 8.2% and 3.8% over the best baselines on three experimental settings.

5.1 Introduction

Graph representation learning has been at the center of various real-world applications, such as drug discovery [33, 53], recommender systems [169, 39, 123], forecasting [135, 27] and outlier detection [191, 25]. In recent years, there has been a surge of interest in developing graph neural networks (GNNs) as powerful tools for graph representation learning [78, 139, 57, 156]. Remarkably, GNNs have achieved state-of-the-art performance on numerous graph-related tasks including node classification, graph classification and link prediction [19, 171, 193].

Despite the enormous success of GNNs, recent studies have revealed that their generalization and robustness are immensely challenged by the data quality [67, 83]. In particular, GNNs can behave unreliably in scenarios where sub-optimal data is presented:

1. *Distribution shift* [154, 205]. GNNs tend to yield inferior performance when the distributions of training and test data are not aligned (due to corruption or inconsistent collection procedure of

test data).

2. *Abnormal features* [96]. GNNs suffer from high classification errors when data contains abnormal features, e.g., incorrect user profile information in social networks.
3. *Adversarial structure attack* [208, 84]. GNNs are vulnerable to imperceptible perturbations on the graph structure which can lead to severe performance degradation.

To tackle these problems, significant efforts have been made on developing new techniques from the *modeling perspective*, e.g., designing new architectures and employing adversarial training strategies [159, 154]. However, employing these methods in practice may be infeasible, as they require additional cost of changing model architectures or re-training model parameters, especially for well-trained large-scale models. The problem is further exacerbated when adopting these techniques for multiple architectures. By contrast, this paper seeks to investigate approaches that can be readily used with a wide variety of pre-trained models and test settings for improving model generalization and robustness. Essentially, we provide a *data-centric perspective* to address the aforementioned issues by modifying the graph data presented at test-time. Such modification aims to bridge the gap between training data and test data, and thus enable GNNs to achieve better generalization and robust performance on the new graph. Figure 5.1 visually describes this idea: we are originally given with a test graph with abnormal features where multiple GNN architectures yield poor performance; however, by transforming the graph prior to inference (at test-time), we enable these GNNs to achieve much higher accuracy.

In this work, we aim to develop a data-centric framework that transforms the test graph to enhance model generalization and robustness, without altering the pre-trained model. In essence, we are faced with two challenges: (1) how to model and optimize the transformed graph data, and (2) how to formulate an objective that can guide the transformation process. First, we model the graph transformation as injecting perturbation on the node features and graph structure, and optimize them alternatively via gradient descent. Second, inspired by the recent progress of contrastive learning, we propose a parameter-free surrogate loss which does not affect the pre-training process while effectively guiding the graph adaptation. Our contributions can be summarized as follows:

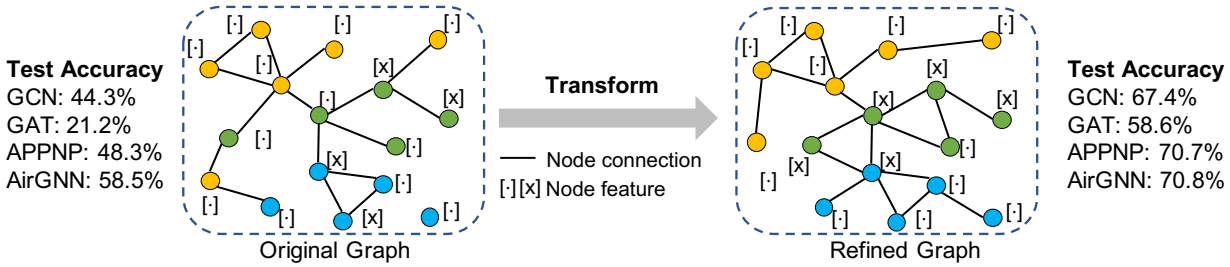


Figure 5.1: We study the test-time graph transformation problem, which seeks to learn a refined graph such that pre-trained GNNs can perform better on the new graph compared to the original. **Shown:** An illustration of our proposed approach's empirical performance on transforming a noisy graph.

1. For the first time, we provide a data-centric perspective to improve the generalization and robustness of GNNs with test-time graph transformation.
2. We establish a novel framework GTrans for test-time graph transformation by jointly learning the features and adjacency structure to minimize a proposed surrogate loss.
3. Our theoretical analysis provides insights on what surrogate losses we should use during test-time graph transformation and sheds light on the power of data-adaptation over model-adaptation.
4. Extensive experimental results on three settings (distribution shift, abnormal features and adversarial structure attacks) have demonstrated the superiority of test-time graph transformation. Particularly, GTrans performs the best in most cases with improvements up to 2.8%, 8.2% and 3.8% over the best baselines on three experimental settings.

Moreover, we note: (1) GTrans is flexible and versatile. It can be equipped with any pre-trained GNNs and the outcome (the refined graph data) can be deployed with any model given its favorable transferability. (2) GTrans provides a degree of interpretability, as it can show which kinds of graph modifications can help improve performance by visualizing the data.

5.2 Related Work

Distribution Shift in GNNs. GNNs have revolutionized graph representation learning and achieved state-of-the-art results on diverse graph-related tasks [78, 139, 57, 19, 79, 156]. However, recent studies have demonstrated that GNNs yield sub-optimal performance on out-of-distribution data for node classification [205, 154, 92, 107] and graph classification [17, 11, 52, 155]. These studies

have introduced solutions to tackle distribution shifts. For example, EERM [154] attributes the cause of distribution shifts to an unknown environmental variable, and trains GNNs under multiple environments produced by a generator. For a thorough review, we refer the readers to a recent survey [83]. Unlike existing works, we target modifying the inputs via test-time adaption.

Robustness of GNNs. Recent studies have demonstrated the vulnerability of GNNs to graph adversarial attacks [208, 210, 159, 21, 48], i.e., small perturbations on the input graph can mislead GNNs into making wrong predictions. Several works make efforts towards developing new GNNs or adversarial training strategies to defend against attacks [159, 201, 66, 67]. Instead of altering model training behavior, our work aims to modify the test graph to correct adversarial patterns.

Graph Structure Learning & Graph Data Augmentation. Graph structure learning and graph data augmentation both aim to improve GNNs' generalization performance by augmenting the (training) graph data, either learning the graph from scratch [45, 70, 18, 194, 121, 56, 41] or perturbing the graph in a rule-based way [119, 42, 28]. While our work also modifies the graph data, we focus on modifying the test data and not impacting the model training process.

Test-time Training. Our work is also related to test-time training [130, 142, 100, 181, 185], which has raised a surge of interest in computer vision recently. To handle out-of-distribution data, [130] propose the pioneer work of test-time training (TTT) by optimizing feature extractor via an auxiliary task loss. However, TTT alters training to jointly optimize the supervised loss and auxiliary task loss. To remove the need for training an auxiliary task, Tent [142] proposes to minimize the prediction entropy at test-time. It is worth noting that Tent works by adapting the statistics and parameters in batch normalization (BN) layers, which may not always be employed by modern GNNs. In this work, we focus on a novel perspective of adapting the test graph data instead of the model, which makes no assumptions about the particular training procedure or architecture.

5.3 The Proposed Framework

We start by introducing the general problem of test-time graph transformation (TTGT). While our discussion mainly focuses on the node classification task where the goal is to predict the labels of nodes in the graph, it can be easily extended to other tasks. Consider that we have a training

graph G_{Tr} and a test graph G_{Te}, and the corresponding set of node labels are denoted as $\mathcal{Y}_{\text{Tr}}$ and $\mathcal{Y}_{\text{Te}}$, respectively. Note that the node sets in G_{Tr} and G_{Te} can either be disjoint or overlapping, and they are not necessarily drawn from the same distribution. Further, let $f_\theta(\cdot)$ denote the mapping function of a GNN model parameterized by θ, which maps a graph into the space of node labels.

Definition 1 (Test-Time Graph Transformation (TTGT))**.** *TTGT requires to learn a graph transformation function $g(\cdot)$ which refines the test graph G_{Te} such that the pre-trained f_θ can yield better test performance on $g(G_{Te})$ than that on G_{Te}:*

$$\begin{gathered}\arg\min_{g} \mathcal{L}\left(f_{\theta^*}(g(G_{Te})), \mathcal{Y}_{Te}\right) \quad s.t. \quad g(G_{Te}) \in \mathcal{P}(G_{Te}), \\ with\ \theta^* = \arg\min_{\theta} \mathcal{L}\left(f_\theta(G_{Tr}), \mathcal{Y}_{Tr}\right),\end{gathered} \tag{5.1}$$

where $\mathcal{L}$ denotes the loss function measuring downstream performance; $\mathcal{P}(G_{Te})$ is the space of the modified graph, e.g., we may constrain the change on the graph to be small.

To optimize the TTGT problem, we are faced with two critical challenges: (1) how to parameterize and optimize the graph transformation function $g(\cdot)$; and (2) how to formulate a surrogate loss to guide the learning process, since we do not have access to the ground-truth labels of test nodes. Therefore, we propose GTrans and elaborate on how it addresses these challenges as follows.

5.3.1 Constructing Graph Transformation

In this subsection, we introduce how to construct the graph transformation function and detail its optimization process. Let $G_{\text{Te}} = \{\mathbf{A}, \mathbf{X}\}$ denote the test graph, where $\mathbf{A} \in \{0,1\}^{N\times N}$ is the adjacency matrix, N is the number of nodes, and $\mathbf{X} \in \mathbb{R}^{N\times d}$ is the d-dimensional node feature matrix. Since the pre-trained GNN parameters are fixed at test time and we only care about the test graph, we drop the subscript in G_{Te} and $\mathcal{Y}_{\text{Te}}$ to simplify notations in the rest of the paper.

Construction. We are interested in obtaining the transformed test graph $G'_{\text{Te}} = g(\mathbf{A}, \mathbf{X}) = (\mathbf{A}', \mathbf{X}')$. Specifically, we model feature modification as an additive function which injects perturbation to node features, i.e., $\mathbf{X}' = \mathbf{X} + \Delta_{\mathbf{X}}$; we model the structure modification as $\mathbf{A}' = \mathbf{A} \oplus \Delta_{\mathbf{A}}$[1], where $\oplus$ stands for an element-wise exclusive OR operation and $\Delta_{\mathbf{A}} \in \{0,1\}^{N\times N}$ is a binary matrix.

[1] $(\mathbf{A} \oplus \Delta_{\mathbf{A}})_{ij}$ can be implemented as $2 - (\mathbf{A} + \Delta_{\mathbf{A}})_{ij}$ if $(\mathbf{A} + \Delta_{\mathbf{A}})_{ij} \geq 1$, otherwise $(\mathbf{A} + \Delta_{\mathbf{A}})_{ij}$.

In other words, $(\Delta_{\mathbf{A}})_{ij} = 1$ indicates an edge flip. Formally, we seek to find $\Delta_{\mathbf{A}}$ and $\Delta_{\mathbf{X}}$ that can minimize the objective function:

$$\arg\min_{\Delta_{\mathbf{A}}, \Delta_{\mathbf{X}}} \mathcal{L}\left(f_\theta(\mathbf{A} \oplus \Delta_{\mathbf{A}}, \mathbf{X} + \Delta_{\mathbf{X}}), \mathcal{Y}\right) \quad \text{s.t.} \quad (\mathbf{A} \oplus \Delta_{\mathbf{A}}, \mathbf{X} + \Delta_{\mathbf{X}}) \in \mathcal{P}(\mathbf{A}, \mathbf{X}), \tag{5.2}$$

where $\Delta_{\mathbf{A}} \in \{0, 1\}^{N \times N}$ and $\Delta_{\mathbf{X}} \in \mathbb{R}^{N \times d}$ are treated as free parameters. Further, to ensure we do not heavily violate the original graph structure, we constrain the number of changed entries in the adjacency matrix to be smaller than a budget B on the graph structure, i.e., $\|\Delta_{\mathbf{A}}\|_F \leq B$. We do not impose constraints on the node features to ease optimization. In this context, $\mathcal{P}$ can be viewed as constraining $\Delta_{\mathbf{A}}$ to a binary space as well as restricting the number of changes.

Optimization. Jointly optimizing $\Delta_{\mathbf{X}}$ and $\Delta_{\mathbf{A}}$ is often challenging as they depend on each other. In practice, we alternatively optimize $\Delta_{\mathbf{X}}$ and $\Delta_{\mathbf{A}}$. Notably, the optimization for $\Delta_{\mathbf{X}}$ is easy since the node features are continuous. The optimization for $\Delta_{\mathbf{A}}$ is particularly difficult in that (1) $\Delta_{\mathbf{A}}$ is binary and constrained; and (2) the search space of N^2 entries is too large especially when we are learning on large-scale graphs.

To cope with the first challenge, we relax the binary space to $[0, 1]^{N \times N}$ and then we can employ projected gradient descent (PGD) [159, 48] to update $\Delta_{\mathbf{A}}$:

$$\Delta_{\mathbf{A}} \leftarrow \Pi_{\mathcal{P}}\left(\Delta_{\mathbf{A}} - \eta \nabla_{\Delta_{\mathbf{A}}} \mathcal{L}(\Delta_{\mathbf{A}})\right) \tag{5.3}$$

where we first perform gradient descent with step size η and call a projection $\Pi_{\mathcal{P}}$ to project the variable to the space $\mathcal{P}$. Specifically, given an input vector $\mathbf{p}$, $\Pi_{\mathcal{P}}(\cdot)$ is expressed as:

$$\Pi_{\mathcal{P}}(\mathbf{p}) \leftarrow \begin{cases} \Pi_{[0,1]}(\mathbf{p}), & \text{If } \mathbf{1}^\top \Pi_{[0,1]}(\mathbf{p}) \leq B; \\ \Pi_{[0,1]}(\mathbf{p} - \gamma \mathbf{1}) \text{ with } \mathbf{1}^\top \Pi_{[0,1]}(\mathbf{p} - \gamma \mathbf{1}) = B, & \text{otherwise,} \end{cases} \tag{5.4}$$

where $\Pi_{[0.1]}(\cdot)$ clamps the input values to $[0, 1]$, $\mathbf{1}$ stands for a vector of all ones, and γ is obtained by solving the equation $\mathbf{1}^\top \Pi_{[0,1]}(\mathbf{p} - \gamma \mathbf{1}) = B$ with the bisection method [95]. To keep the adjacency structure discrete and sparse, we view each entry in $\mathbf{A} \oplus \Delta_{\mathbf{A}}$ as a Bernoulli distribution and sample the learned graph as $\mathbf{A}' \sim \text{Bernoulli}(\mathbf{A} \oplus \Delta_{\mathbf{A}})$.

Furthermore, to enable efficient graph structure learning, it is desired to reduce the search space of updated adjacency matrix. One recent approach of graph adversarial attack [48] proposes to

sample a small block of possible entries from the adjacency matrix and update them at each iteration. This solution is still computationally intensive as it requires hundreds of steps to achieve a good performance. Instead, we constrain the search space to only the existing edges of the graph, which is typically sparse. Empirically, we observe that this simpler strategy still learns useful structure information when combined with feature modification.

5.3.2 Parameter-Free Surrogate Loss

As discussed earlier, the proposed framework GTrans aims to improve the model generalization and robustness by learning to transform the test graph. Ideally, when we have test ground-truth labels, the problem can be readily solved by adapting the graph to result in the minimum cross entropy loss on test samples. However, as we do not have such information at test-time, it motivates us to investigate feasible surrogate losses to guide the graph transformation process. In the absence of labeled data, recently emerging self-supervised learning techniques [157, 99] have paved the way for providing self-supervision for TTGT. However, not every surrogate self-supervised task and loss is suitable for our transformation process, as some tasks are more powerful and some are weaker. To choose a suitable surrogate loss, we provide the following theorem.

Theorem 3. *Let $\mathcal{L}_c$ denote the classification loss and $\mathcal{L}_s$ denote the surrogate loss, respectively. Let $\rho(G)$ denote the correlation between $\nabla_G \mathcal{L}_c(G, \mathcal{Y})$ and $\nabla_G \mathcal{L}_s(G)$, and let ϵ denote the learning rate for gradient descent. Assume that $\mathcal{L}_c$ is twice-differentiable and its Hessian matrix satisfies $\|\mathbf{H}(G, \mathcal{Y})\|_2 \leq M$ for all G. When $\rho(G) > 0$ and $\epsilon < \frac{2\rho(G)\|\nabla_G \mathcal{L}_c(G,\mathcal{Y})\|_2}{M\|\nabla_G \mathcal{L}_s(G)\|_2}$, we have*

$$\mathcal{L}_c\left(G - \epsilon \nabla_G \mathcal{L}_s\left(G\right), \mathcal{Y}\right) < \mathcal{L}_c\left(G, \mathcal{Y}\right). \tag{5.5}$$

The proof can be found in Section C.1.1. Theorem 3 suggests that when the gradients from classification loss and surrogate loss have a positive correlation, i.e., $\rho\left(G\right) > 0$, we can update the test graph by performing gradient descent with a sufficiently small learning rate such that the classification loss on the test samples is reduced. Hence, it is imperative to find a surrogate task that shares relevant information with the classification task. To empirically verify the effectiveness of Theorem 1, we adopt the surrogate loss in Equation (5.6) as $\mathcal{L}_s$ and plot the values of $\rho(G)$ and

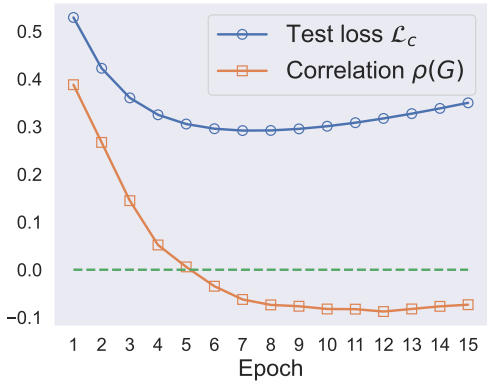


Figure 5.2: Positive $\rho(G)$ can help reduce test loss.

$\mathcal{L}_c$ on one test graph in Cora in Figure 5.2. We can observe that a positive $\rho(G)$ generally reduces the test classification loss. More results on different surrogate losses can be found in Figure C.4 in Section C.4.9 and similar patterns are exhibited.

Parameter-Free Surrogate Loss. As one popular self-supervised paradigm, graph contrastive learning has achieved promising performance in various graph-related tasks [59, 171, 207], which indicates that graph contrastive learning tasks are often highly correlated with downstream tasks. This property is desirable for guiding TTGT as suggested by Theorem 1. At its core lies the contrasting scheme, where the similarity between two augmented views from the same sample is maximized, while the similarity between views from two different samples is minimized.

However, the majority of existing graph contrastive learning methods cannot be directly applied to our scenario, as they often require a parameterized projection head to map augmented representations to another latent space, which inevitably alters the model architecture. To address this issue, we propose a parameter-free surrogate loss. Specifically, we apply an augmentation function $\mathcal{A}(\cdot)$ on input graph G and obtain an augmented graph $\mathcal{A}(G)$. The node representations obtained from the two graphs are denoted as Z and $\hat{Z}$, respectively; $\mathbf{z}_i$ and $\hat{\mathbf{z}}_i$ stand for the i-th node representation taken from Z and $\hat{Z}$, respectively. We adopt DropEdge [119] as the augmentation function $\mathcal{A}(\cdot)$, and the hidden presentations $\mathbf{z}_i$ and $\hat{\mathbf{z}}_i$ are taken from the output of the second last layer of the pre-trained model. Essentially, we maximize the cosine similarity between original nodes and their augmented

view while penalizing the similarity between the nodes and their negative samples:

$$\min \mathcal{L}_s = \sum_{i=1}^{N}(1 - \frac{\hat{\mathbf{z}}_i^\top \mathbf{z}_i}{\|\hat{\mathbf{z}}_i\|\|\mathbf{z}_i\|}) - \sum_{i=1}^{N}(1 - \frac{\tilde{\mathbf{z}}_i^\top \mathbf{z}_i}{\|\tilde{\mathbf{z}}_i\|\|\mathbf{z}_i\|}), \tag{5.6}$$

where $\{\tilde{\mathbf{z}}_i | i = 1, \ldots, N\}$ are the negative samples for corresponding nodes, which are generated by shuffling node features [140]. In Eq. (5.6), the first term encourages each node to be close while the second term pushes each node away from the corresponding negative sample. Note that (1) $\mathcal{L}_s$ is parameter-free and does not require modification of the model architecture, or affect the pre-training process; (2) there could be other self-supervised signals for guiding the graph transformation, and we empirically compare them with the contrastive loss in Section C.4.9. We also highlight that our unique contribution is not the loss in Eq. (5.6) but the proposed TTGT framework and the theoretical and empirical insights on how to choose a suitable surrogate loss. Furthermore, the algorithm of GTrans is provided in Section C.2.

5.3.3 Further Analysis

In this subsection, we study the theoretical property of the proposed surrogate loss and compare the strategy of adapting data versus that of adapting model. We first demonstrate the rationality of the proposed surrogate loss through the following theorem.

Theorem 4. *Assume that the augmentation function $\mathcal{A}(\cdot)$ generates a data view of the same class for the test nodes and the node classes are balanced. Assume for each class, the mean of the representations obtained from Z and $\hat{Z}$ are the same. Minimizing the first term in Eq. (5.6) is approximately minimizing the class-conditional entropy $H(Z|Y)$ between features Z and labels Y.*

The proof can be found in Section C.1.2. Theorem 4 indicates that minimizing the first term in Eq. (5.6) will approximately minimize $H(Z|Y)$, which encourages high intra-class compactness, i.e., learning a low-entropy cluster in the embedded space for each class. Notably, $H(Z|Y)$ can be rewritten as $H(Z|Y) = H(Z) - I(Z, Y)$. It indicates that minimizing Eq. (5.6) can also help promote $I(Z, Y)$, the mutual information between the hidden representation and downstream class. However, we note that only optimizing this term can result in collapse (mapping all data points to a single point in the embedded space), which stresses the necessity of the second term in Eq. (5.6).

Next, we use an illustrative example to show that adapting data at test-time can be more useful than adapting model in some cases. Given test samples $\{\mathbf{x}_i | i = 1, \ldots, K\}$, we consider a linearized GNN f_θ which first performs aggregation through a function $\mathrm{Agg}(\cdot, \cdot)$ and then transforms the aggregated features via a function $\mathrm{Trans}(\cdot)$. Hence, only the function $\mathrm{Trans}(\cdot)$ is parameterized by θ.

Example. Let $\mathcal{N}_i$ denote the neighbors for node $\mathbf{x}_i$. If there exist two nodes with the same aggregated features but different labels, i.e., $\mathrm{Agg}(\mathbf{x}_1, \{\mathbf{x}_i | i \in \mathcal{N}_1\}) = \mathrm{Agg}(\mathbf{x}_2, \{\mathbf{x}_j | j \in \mathcal{N}_2\})$, $y_1 \neq y_2$, adapting the data $\{\mathbf{x}_i | i = 1, \ldots, K\}$ can achieve lower classification error than adapting the model f_θ at test stage.

Illustration. Let $\bar{\mathbf{x}}_1 = \mathrm{Agg}(\mathbf{x}_1, \{\mathbf{x}_i | i \in \mathcal{N}_1\})$ and $\bar{\mathbf{x}}_2 = \mathrm{Agg}(\mathbf{x}_2, \{\mathbf{x}_j | j \in \mathcal{N}_2\})$. For simplicity, we consider the following mean square loss as the classification error:

$$\ell = \frac{1}{2}\left(\mathrm{Trans}(\bar{\mathbf{x}}_1)) - y_1)^2 + (\mathrm{Trans}(\bar{\mathbf{x}}_2) - y_2)^2\right). \tag{5.7}$$

It is easy to see that ℓ reaches its minimum when $\mathrm{Trans}(\bar{\mathbf{x}}_1) = y_1$ and $\mathrm{Trans}(\bar{\mathbf{x}}_2) = y_2$. In this context, it is impossible to find θ such that $\mathrm{Trans}(\cdot)$ can map $\mathbf{x}_1, \mathbf{x}_2$ to different labels since it is not a one-to-many function. However, since y_1 and y_2 are in the label space of training data, we can always modify the test graph to obtain newly aggregated features $\bar{\mathbf{x}}'_1, \bar{\mathbf{x}}'_2$ such that $\mathrm{Trans}(\bar{\mathbf{x}}'_1) = y_1$ and $\mathrm{Trans}(\bar{\mathbf{x}}'_2) = y_2$, which minimizes ℓ. In the extreme case, we may drop all node connections for the two nodes, and let $\mathbf{x}_1 \leftarrow \bar{\mathbf{x}}'_1$ and $\mathbf{x}_2 \leftarrow \bar{\mathbf{x}}'_2$ where $\bar{\mathbf{x}}'_1$ and $\bar{\mathbf{x}}'_2$ are the aggregated features taken from the training set. Hence, adapting data can achieve lower classification loss.

Remark 1. Note that the existence of two nodes with the same aggregated features but different labels is not rare when considering adversarial attack or abnormal features. We provide a figurative example in Figure C.1 in Section C.1.3: the attacker injects one adversarial edge into the graph and changes the aggregated features $\bar{\mathbf{x}}_1$ and $\bar{\mathbf{x}}_2$ to be the same.

Remark 2. When we consider $\bar{\mathbf{x}}_1 \neq \bar{\mathbf{x}}_2$, $y_1 \neq y_2$, whether we can find θ satisfying $\mathrm{Trans}(\bar{\mathbf{x}}_1) = y_1$ and $\mathrm{Trans}(\bar{\mathbf{x}}_2) = y_2$ depends on the expressiveness of the the transformation function. If it is not powerful enough (e.g., an under-parameterized neural network), it could fail to map different data points to different labels. On the contrary, adapting the data does not suffer this problem as we can always modify the test graph to satisfy $\mathrm{Trans}(\bar{\mathbf{x}}'_1) = y_1$ and $\mathrm{Trans}(\bar{\mathbf{x}}'_2) = y_2$.

Remark 3. The above discussion can be easily extended to nonlinear GNN by considering $\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2$ as the output before the last linear layer of GNN.

5.4 Experiment

5.4.1 Generalization on Out-of-distribution Data

Setup. Following the settings in EERM [154], which is designed for node-level tasks on OOD data, we validate GTrans on three types of distribution shifts with six benchmark datasets: (1) artificial transformation for Cora [166] and Amazon-Photo [127], (2) cross-domain transfers for Twitch-E and FB-100 [120] [89], and (3) temporal evolution for Elliptic [114] and Ogbn-arxiv [60]. Moreoever, Cora and Amazon-Photo have 1/1/8 graphs for training/validation/test sets. The splits are 1/1/5 on Twitch-E, 3/2/3 on FB-100, 5/5/33 on Elliptic, and 1/1/3 on Ogbn-arxiv. We compare GTrans with four baselines: empirical risk minimization (ERM, i.e., standard training), data augmentation technique DropEdge [119], test-time-training method Tent [142], and the recent SOTA method EERM [154] which is exclusively developed for graph OOD issue. Note that SR-GNN [205] is not included as a baseline because it specifically targets distribution shifts between the selection of training and testing nodes instead of the general OOD issue. By contrast, both Tent and EERM are designed to handle general distribution shifts. Further, we evaluate all the methods with four popular GNN backbones including GCN [78], GraphSAGE [57], GAT [139], and GPR [19]. Their default setup follows that in EERM[2]. Notably, all experiments in this paper are repeated 10 times with different random seeds.

Results. Table 5.1 reports the averaged performance over the test graphs for each dataset as well as the averaged rank of each algorithm. From the table, we make the following observations:

(a) Overall Performance. The proposed framework consistently achieves strong performance across the datasets: GTrans achieves average ranks of 1.0, 1.7, 2.0 and 1.7 with GCN, SAGE, GAT and GPR, respectively, while the corresponding ranks for the best baseline EERM are 2.9, 3.4, 3.0 and 2.0. Furthermore, in most of the cases, GTrans significantly improves the vanilla baseline

[2]We note that the GCN used in the experiments of EERM does not normalize the adjacency matrix according to its open-source code. Here we normalize the adjacency matrix to make it consistent with the original GCN.

Table 5.1: Average classification performance (%) on the test graphs. Rank indicates the average rank of each algorithm for each backbone. OOM indicates out-of-memory error on 32 GB GPU memory. The proposed GTrans consistently ranks the best compared with the baselines. */** indicates that GTrans outperforms ERM at the confidence level 0.1/0.05 from paired t-test.

Backbone	Method	Amz-Photo	Cora	Elliptic	FB-100	Ogbn-arxiv	Twitch-E	Rank
GCN	ERM	93.79±0.97	91.59±1.44	50.90±1.51	54.04±0.94	38.59±1.35	59.89±0.50	3.8
	DropEdge	92.11±0.31	81.01±1.33	53.96±4.91	53.00±0.50	41.26±0.92	59.95±0.39	3.6
	Tent	94.03±1.07	91.87±1.36	51.71±2.00	54.16±1.00	39.33±1.40	59.46±0.55	3.3
	EERM	94.05±0.40	87.21±0.53	53.96±0.65	54.24±0.55	OOM	59.85±0.85	2.9
	GTrans	**94.13±0.77***	**94.66±0.63****	**55.88±3.10****	**54.32±0.60**	**41.59±1.20****	**60.42±0.86***	**1.0**
SAGE	ERM	95.09±0.60	99.67±0.14	56.12±4.47	54.70±0.47	39.56±1.66	62.06±0.09	3.2
	DropEdge	92.61±0.56	95.85±0.30	52.38±3.11	54.51±0.69	38.89±1.74	62.14±0.12	4.2
	Tent	95.72±0.43	**99.80±0.10**	55.89±4.87	**54.86±0.34**	39.58±1.26	62.09±0.09	2.3
	EERM	95.57±0.13	98.77±0.14	58.20±3.55	54.28±0.97	OOM	62.11±0.12	3.4
	GTrans	**96.91±0.68****	99.45±0.13	**60.81±5.19****	54.64±0.62	**40.39±1.45****	**62.15±0.13***	**1.7**
GAT	ERM	96.30±0.79	94.81±1.28	65.36±2.70	51.77±1.41	40.63±1.57	58.53±1.00	3.0
	DropEdge	90.70±0.29	76.91±1.55	63.78±2.39	52.65±0.88	42.48±0.93	58.89±1.01	3.3
	Tent	95.99±0.46	95.91±1.14	66.07±1.66	51.47±1.70	40.06±1.19	58.33±1.18	3.3
	EERM	95.57±1.32	85.00±0.96	58.14±4.71	**53.30±0.77**	OOM	**59.84±0.71**	3.0
	GTrans	**96.67±0.74****	**96.37±1.00****	**66.43±2.57****	51.16±1.72	**43.76±1.25****	58.59±1.07	**2.0**
GPR	ERM	91.87±0.65	93.00±2.17	64.59±3.52	54.51±0.33	44.38±0.59	59.72±0.40	2.7
	DropEdge	88.81±1.48	79.27±1.39	61.02±1.78	55.04±0.33	43.65±0.77	59.89±0.05	3.3
	Tent[3]	-	-	-	-	-	-	-
	EERM	90.78±0.52	88.82±3.10	67.27±0.98	**55.95±0.03**	OOM	**61.57±0.12**	2.0
	GTrans	**91.93±0.73**	**93.05±2.02**	**69.03±2.33****	54.38±0.31	**46.00±0.46****	60.11±0.53**	**1.7**

[3] Tent cannot be applied to models which do not contain batch normalization layers.

(ERM) by a large margin. Particularly, when using GCN as backbone, GTrans outperforms ERM by 3.1%, 5.0% and 2.0% on Cora, Elliptic and Ogbn-arxiv, respectively. These results demonstrate the effectiveness of GTrans in tackling diverse types of distribution shifts.

(b) Comparison to other baselines. Both DropEdge and EERM modify the training process to improve model generalization. Nonetheless, they are less effective than GTrans, as GTrans takes advantage of the information from test graphs. As a test-time training method, Tent also performs well in some cases, but Tent only adapts the parameters in batch normalization layers and cannot be applied to models without batch normalization.

We further show the performance on each test graph on Cora with GCN in Figure 5.3. We observe that GTrans generally improves over individual test graphs within each dataset, which validates the effectiveness of GTrans.

Efficiency Comparison. Since EERM performs the best among baselines, Table 5.2 showcases

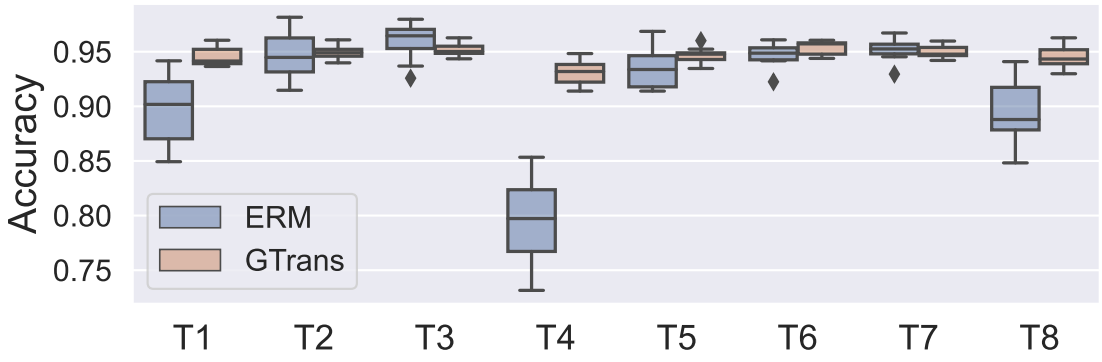


Figure 5.3: Results on Cora under OOD. GTrans improves GCN on most test graphs.

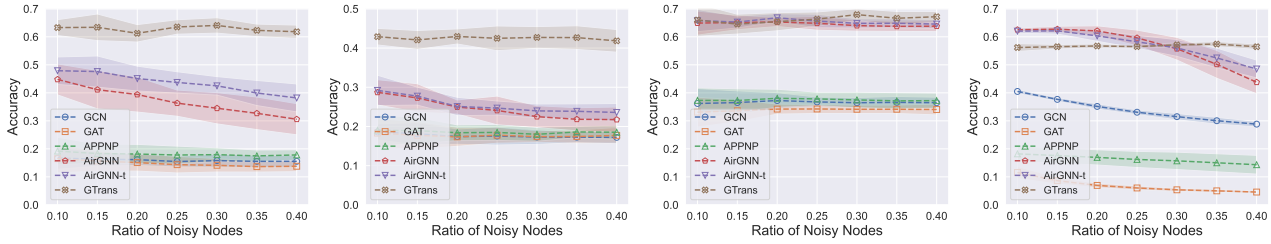
Table 5.2: Efficiency comparison. GTrans is more time- and memory-efficient than EERM.

	Extra Running Time (s)				Total GPU Memory (GB)			
	Cora	Photo	Ellip.	Arxiv	Cora	Photo	Ellip.	Arxiv
EERM	25.9	396.4	607.9	-	2.5	10.5	12.8	>32
GTrans	0.3	0.5	0.6	2.6	1.4	1.5	1.3	3.9

the efficiency comparison between our proposed GTrans and EERM on the largest test graph in each dataset. The additional running time of GTrans majorly depends on the number of gradient descent steps. As we only use a small number (5 or 10) throughout all the experiments, the time overhead brought by GTrans is negligible. Compared with the re-training method EERM, GTrans avoids the complex bilevel optimization and thus is significantly more efficient. Furthermore, EERM imposes a considerably heavier memory burden.

5.4.2 Robustness to Abnormal Features

Setup. Following the setup in AirGNN [96], we evaluate the robustness in the case of abnormal features. Specifically, we simulate abnormal features by assigning random features taken from a multivariate standard Gaussian distribution to a portion of randomly selected test nodes. Note that the abnormal features are injected after model training (at test time) and we vary the ratio of noisy nodes from 0.1 to 0.4 with a step size of 0.05. This process is performed for four datasets: the original version of Cora, Citeseer, Pubmed, and Ogbn-arxiv. In these four datasets, the training graph and the test graph have the same graph structure but the node features are different. Hence, we use the training classification loss combined with the proposed contrastive loss to optimize

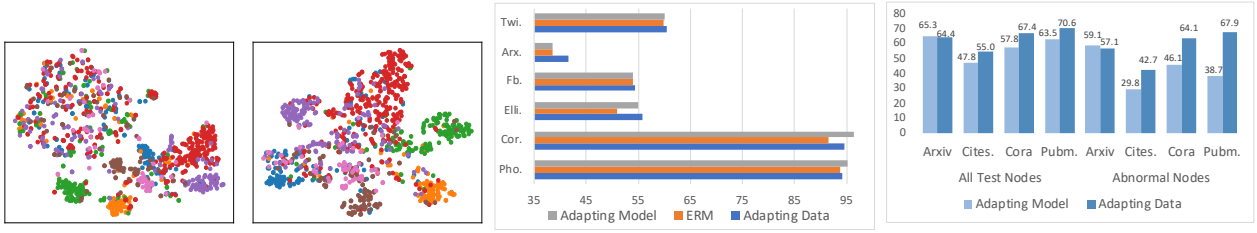


(a) Cora (b) Citeseer (c) Pubmed (d) Ogbn-arxiv

Figure 5.4: Node classification accuracy on abnormal (noisy) nodes.

GTrans. We use GCN as the backbone model and adopt four GNNs as the baselines including GAT [139], APPNP [79], AirGNN and AirGNN-t. Note that AirGNN-t tunes the message-passing hyper-parameter in AirGNN at test time. For a fair comparison, we tune AirGNN-t based on the performance on both training and validation nodes.

Results. For each model, we present the node classification accuracy on both abnormal nodes and all test nodes (i.e., both normal and abnormal ones) in Figure 5.4 and Figure C.3 (See Section C.4.5), respectively. From these figures, we have two observations. First, GTrans significantly improves GCN in terms of the performance on abnormal nodes and all test nodes for all datasets across all noise ratios. For example, on Cora with 30% noisy nodes, GTrans improves GCN by 48.2% on abnormal nodes and 31.0% on overall test accuracy. This demonstrates the effectiveness of the graph transformation process in GTrans in alleviating the effect of abnormal features. Second, GTrans shows comparable or better performance with AirGNNs, which are the SOTA defense methods for tackling abnormal features. It is worth mentioning that AirGNN-t improves AirGNN by tuning its hyper-parameter at test time, which aligns with our motivation that test-time adaptation can enhance model test performance. To further understand the effect of graph transformation, we provide the visualization of the test node embeddings obtained from abnormal graph (0.3 noise ratio) and transformed graph for Cora in Figures 5.5a and 5.5b, respectively. We observe that the transformed graph results in well-clustered node representations, which indicates that GTrans can promote intra-class compactness and counteract the effect of abnormal patterns.



(a) Abnormal Graph (b) Refined Graph (c) OOD Setting (d) Abnormal Features

Figure 5.5: **(a)(b)** T-SNE visualizations of embedding obtained from abnormal graph and transformed graph on Cora. **(c)(d)** Comparison between adapting data and adapting model at test time.

Table 5.3: Node classification accuracy (%) under different perturbation (Ptb.) rates of structure attack.

Ptb. Rate	GCN	GAT	RobustGCN	SimPGCN	GCNJaccard	GTrans
5%	57.47±0.54	64.56±0.43	61.55±1.20	61.30±0.42	65.01±0.26	**66.29±0.25**
10%	47.97±0.65	61.20±0.70	58.15±1.55	57.01±0.70	63.25±0.30	**65.16±0.52**
15%	38.04±1.22	58.96±0.59	55.91±1.27	54.13±0.73	61.83±0.29	**64.40±0.38**
20%	29.05±0.73	57.29±0.49	54.39±1.09	52.26±0.87	60.57±0.34	**63.44±0.50**
25%	19.58±2.32	55.86±0.53	52.76±1.44	50.46±0.85	59.17±0.39	**62.95±0.67**

5.4.3 Robustness to Adversarial Attack

Setup. We further evaluate GTrans under the setting of adversarial attack where we perturb the test graph, i.e., evasion attack. Specifically, we use PR-BCD [48], a scalable attack method, to attack the test graph in Ogbn-arxiv. We focus on structural attacks, and vary the perturbation rate, i.e., the ratio of changed edges, from 5% to 25% with a step of 5%. Similar to Section 5.4.2, we adopt the training classification loss together with the proposed contrastive loss to optimize GTrans. We use GCN as the backbone and employ four robust baselines including GAT [139], RobustGCN [201], SimPGCN [66] and GCNJaccard [159] as comparisons. Among them, GCNJaccard pre-processes the attacked graph by removing edges where the similarities of connected nodes are less than a threshold; we tune this threshold at test time based on the performance on both training and validation nodes.

Results. Table 5.3 reports the performances under structural evasion attack. We observe that GTrans consistently improves the performance of GCN under different perturbation rates of adversarial attack. Particularly, GTrans improves GCN by a larger margin when the perturbation rate is higher. For example, GTrans outperforms GCN by over 40% under the 25% perturbation

rate. Such observation suggests that GTrans can counteract the devastating effect of adversarial attacks. In addition, the best performing baseline GCNJaccard also modifies the graph at test time, which demonstrates the importance of test-time graph adaptation. Nonetheless, it consistently underperforms our proposed GTrans, indicating that a learnable transformation function is needed to achieve better robustness under adversarial attacks, which GCNJaccard does not employ.

Interpretation. To understand the modifications made on the graph, we compare several properties among clean graph, attacked graph (20% perturbation rate), graph obtained by GCNJaccard, and graph obtained by GTrans in Table C.9 in Appendix C.4.6. First, adversarial attack decreases homophily and feature similarity, but GTrans and GCNJaccard promote such information to alleivate the adversarial patterns. Our experiment also shows that GTrans removes 77% adversarial edges while removing 30% existing edges from the attacked graph. Second, both GTrans and GCNJaccard focus on deleting edges from the attacked graph, but GCNJaccard removes a substantially larger amount of edges, which may destroy clean graph structure and lead to sub-optimal performance.

5.4.4 Further Analysis

Cross-Architecture Transferability. Since the outcome of GTrans is a refined graph, it can conceptually be employed by any GNN model. In other words, we can transform the graph based on one pre-trained GNN and test the transformed graph on another pre-trained GNN. To examine such transferability, we perform experiments on four GNNs including GCN, APPNP, AirGNN and GAT under the abnormal feature setting with 30% noisy nodes on Cora. The results on all test nodes in Table 5.4. Note that “Tr" stands for GNNs used in TTGT while “Te” denotes GNNs used for obtaining final predictions on the transformed graph; “Noisy" indicates the performance on the noisy

Table 5.4: Cross-Architecture Analysis.

Tr\Te	GCN	APPNP	AirGNN	GAT
GCN	67.36	70.65	70.84	58.62
APPNP	67.87	70.39	69.59	64.46
AirGNN	68.00	70.37	72.68	64.93
GAT	54.85	60.37	65.22	54.60
Noisy	44.29	48.26	58.51	21.23

graph. We observe that the transformed graph yields good performance even outside the scope it was optimized for. We anticipate that such transferability can alleviate the need for costly re-training on new GNNs.

Adapting Model vs. Adapting Data. We empirically compare the performance between adapting data and adapting model and consider the OOD and abnormal feature settings. Specifically, we use GCN as the backbone and adapt the model parameters by optimizing the same loss function as used in GTrans. The results are shown in Figures 5.5c and 5.5d. In OOD setting, both adapting model and adapting data can generally improve GCN's performance. Since their performances are still close, it is hard to give a definite answer on which strategy is better. However, we can observe significant performance differences when the graph contains abnormal features: adapting data outperforms adapting model on 3 out of 4 datasets. This suggests that adapting data can be more powerful when the data is perturbed, which aligns with our analysis in Section 5.3.3.

Effect of Surrogate Loss. Recall that we used a combined loss of contrastive loss and training loss in the settings of abnormal features and adversarial attack. We now examine the effect of using them alone and show the results in Tables C.10 and C.11 in Section C.4.7. We observe that (1) both of them bring improvement over vanilla GCN, while optimizing the training loss alone improves more than optimizing the other; and (2) combining them always yields a better or comparable performance.

Learning Features v.s. Learning Structure. Since our framework learns both node features and graph structure, we investigate when one component plays a more important role than the other. Our results are shown in Tables C.12 and C.13 in Section C.4.8. From the tables, we observe that (1) while each component can improve the vanilla performance, feature learning is more crucial for counteracting feature corruption and structure learning is more important for defending structure corruption; and (2) combining them generally yields a better or comparable performance.

5.5 Conclusion

GNNs tend to yield unsatisfying performance when the presented data is sub-optimal. To tackle this issue, we seek to enhance GNNs from a data-centric perspective by transforming the graph data

at test time. We propose GTrans which optimizes a contrastive surrogate loss to transform graph structure and node features, and provide theoretical analysis with deeper discussion to understand this framework. Experimental results on distribution shift, abnormal features and adversarial attack have demonstrated the effectiveness of our method. In the future, we plan to explore more applications of our framework such as mitigating degree bias and long-range dependency.

CHAPTER 6

CONCLUSION

In this chapter, we summarize the research results of this dissertation, discuss their broader impact, and highlight promising research directions.

6.1 Summary

Graphs are pivotal data structures describing the relationships between entities in various domains such as social media, biology, transportation and financial systems [156, 4]. Due to their prevalence and rich descriptive capacity, graph machine learning is a prominent research area with powerful implications. As the generalization of deep neural networks on graph data, graph neural networks (GNNs) have proved to be powerful in learning representations for graphs and associated entities (nodes, edges, subgraphs), and they have been employed in various applications such as node classification [78, 139], node clustering [113, 69], computational biology [149], recommender systems [168, 38] and drug discovery [33]. Despite the tremendous success achieved by GNNs, recent studies have revealed that they are vulnerable to adversarial attacks and it is computationally expensive to train them on large-scale graph datasets [67, 57]. To address these concerns, the focus of many recent advances has been on the development of novel techniques from a modeling perspective, i.e., model-centric approaches [57, 66, 97], which revolve around changing the model while holding the dataset fixed. By contrast, despite its immense potential, the exploration of a data-centric approach remains significantly underdeveloped in GNN research. In this dissertation, we study the data-centric approaches to enhance the scalability and robustness of GNNs: (1) graph condensation for graph neural networks, (2) condensing graphs via one-step gradient matching, (3) graph structure learning for robust graph neural networks, and (4) empowering graph neural networks via test-time graph transformation.

As the application of large-scale graphs proliferates in real-world scenarios, concerns have emerged regarding the storage requirements and training times for graph neural networks. To mitigate these concerns, we introduce the novel concept of graph condensation [72] in the context of graph

neural networks. Graph condensation distills a large original graph into a more compact, synthetic graph, maintaining high levels of informativeness while delivering performance comparable to GNNs trained on the original graph. To solve the condensation problem, we introduce GCond, a method for learning synthetic graphs that replicate the model training trajectory of the original graph. This is achieved by simultaneously condensing both node features and structural information through the optimization of a gradient matching loss.

Despite GCond's promising performance, it suffers from two inherent limitations. First, the condensation process proves to be computationally demanding due to nested optimization requirements. Second, it fails to produce discrete graph structures, which could offer additional storage efficiency benefits. In response to these challenges, we delve into efficient dataset condensation, introducing a method called DosCond [71]. DosCond models the discrete graph structure as a probabilistic model and employs a one-step gradient matching scheme that conducts gradient matching in a single step without requiring the training of network weights. Our theoretical analysis illustrates that this approach can generate synthetic graphs that result in low classification loss on real graphs.

On a separate note, recent studies have highlighted the vulnerability of graph neural networks (GNNs) to adversarial attacks, which are deliberate perturbations. Such susceptibility raises significant concerns for the deployment of GNNs in safety-critical domains, making it imperative to mitigate the adverse effects of these attacks by purifying the data. We have observed that adversarial attacks during the training phase distort essential graph properties, such as low-rank and sparse structures and feature similarity among neighboring nodes. Notably, these intrinsic graph properties are often violated by adversarial attacks. Inspired by this observation, we put forward a unique framework, Pro-GNN [70], designed to learn a clean graph structure from a perturbed graph by recovering these inherent properties. Extensive experiments indicate that Pro-GNN outperforms contemporary defense methods significantly, even in scenarios with heavy perturbations.

In contrast to Pro-GNN, which aims to defend against training-time attacks, we propose a fresh data-centric approach, GTrans, focused on refining suboptimal test graphs to enhance the performance of pre-trained models. GTrans employs a test-time graph transformation framework

that minimizes a parameter-free surrogate loss, thereby improving the quality of the test graph data. This framework can be integrated with any pre-existing graph neural network (GNN) and, thanks to its excellent transferability, the refined graph data can be used with any model. An important characteristic of GTrans is its interpretability. By visualizing the data, GTrans can elucidate the types of graph modifications that lead to performance enhancements, providing valuable insights into the factors that influence the model's performance.

These methods pave the way for more efficient data handling and robust defenses against adversarial threats, leading to enhanced model performance with transferable outcomes. Furthermore, these contributions have deepened our understanding of Data-Centric AI in the graph domain. They demonstrate the considerable potential this approach holds in propelling GNN research and its applications forward, particularly in large-scale, safety-critical scenarios.

6.2 Future Work

In the future, we plan to further unleash the power of data-centric approaches in graph machine learning from the following perspectives:

- **Data-centric approaches for privacy protection.** Recent studies have shown that AI algorithms can carry the risk of disclosing users' private and sensitive information, such as medical records and financial transactions, etc [32, 186]. Thus, we hope to make GNNs privacy-preserving and we aim to develop data-centric approaches to encrypt the dataset to prevent user information leakage while not affecting the performance of models trained on it. To fulfill this need, we can inject small perturbations into the input graph dataset such that the users' privacy cannot be easily inferred by the attackers while the input data is not heavily modified so that downstream models can still effectively perform particular tasks.
- **Data-centric approaches for model explanation.** The interpretability of machine learning models has drawn increasing attention from both academic researchers and industrial practitioners [51, 175]. We aim to build graph machine learning systems that produce interpretable results such that users can fully trust them and take advantage of them. My prior work on graph condensation has shed light on learning synthetic graphs that can preserve sufficient information to train GNNs. These

synthetic graphs can serve as the model-level explanation for GNNs [112]. In addition, it is desired to incorporate graph rules (or prior knowledge) to encourage the graph explanation to be valid.

- **Graph dataset creation.** We also plan to investigate methodologies for graph dataset creation. As previously stated, data quality is crucial to the success of graph neural networks. However, existing benchmark datasets are often noisy, biased, and overly simplified, which may not be extended to the practical scenario and can hinder the evaluation of GNNs. Thus, to address these issues, we plan to construct real-world benchmark datasets that are large-scale and collected from diverse resources to ensure fairness. In fact, we are currently collaborating with Amazon researchers to build such real-world benchmark datasets from industry data. Furthermore, to make dataset creation more efficient, we will develop human-in-the-loop approaches, such as active learning algorithms, to prioritize the most valuable data for humans to annotate.

BIBLIOGRAPHY

[1] Abubakar Abid, Muhammad Fatih Balin, and James Zou. Concrete autoencoders for differentiable feature selection and reconstruction. *ArXiv preprint*, 2019.

[2] Rie Kubota Ando and Tong Zhang. Learning on graph with laplacian regularization. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, 2006.

[3] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016.

[4] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *ArXiv preprint*, 2018.

[5] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *ArXiv preprint*, 2018.

[6] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2009.

[7] Ondrej Bohdal, Yongxin Yang, and Timothy Hospedales. Flexible dataset distillation: Learn labels instead of images. *ArXiv preprint*, 2020.

[8] Aleksandar Bojchevski and Stephan Günnemann. Adversarial attacks on node embeddings via graph poisoning. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, Proceedings of Machine Learning Research, 2019.

[9] Malik Boudiaf, Jérôme Rony, Imtiaz Masud Ziko, Eric Granger, Marco Pedersoli, Pablo Piantanida, and Ismail Ben Ayed. A unifying mutual information view of metric learning: cross-entropy vs. pairwise losses. In *European conference on computer vision*. Springer, 2020.

[10] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[11] Davide Buffelli, Pietro Liò, and Fabio Vandin. Sizeshiftreg: a regularization method for improving size-generalization in graph neural networks. *Advances in Neural Information Processing Systems*, 2022.

[12] Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[13] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, (6), 2009.

[14] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, 2018.

[15] George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A Efros, and Jun-Yan Zhu. Dataset distillation by matching training trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[16] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[17] Yongqiang Chen, Yonggang Zhang, Han Yang, Kaili Ma, Binghui Xie, Tongliang Liu, Bo Han, and James Cheng. Invariance principle meets out-of-distribution generalization on graphs. *ArXiv preprint*, 2022.

[18] Yu Chen, Lingfei Wu, and Mohammed J. Zaki. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[19] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[20] Patrick L Combettes and Jean-Christophe Pesquet. Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*. 2011.

[21] Enyan Dai, Wei Jin, Hui Liu, and Suhang Wang. Towards robust graph neural networks for noisy graphs with sparse labels. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 181–191, 2022.

[22] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial

attack on graph structured data. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, Proceedings of Machine Learning Research, 2018.

[23] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016.

[24] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016.

[25] Ailin Deng and Bryan Hooi. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, number 5, 2021.

[26] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[27] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021.

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

[29] Mucong Ding, Tahseen Rabbani, Bang An, Evan Z Wang, and Furong Huang. Sketch-GNN: Scalable graph neural networks with sublinear training complexity. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[30] Tian Dong, Bo Zhao, and Lingjuan Lyu. Privacy for free: How does dataset condensation help privacy? In *ICML*, 2022.

[31] Xiaowen Dong, Dorina Thanou, Pascal Frossard, and Pierre Vandergheynst. Learning laplacian matrix in smooth graph signal representations. *IEEE Transactions on Signal Processing*, (23), 2016.

[32] Vasisht Duddu, Antoine Boutet, and Virat Shejwalkar. Quantifying privacy leakage in graph embedding. In *MobiQuitous 2020-17th EAI International Conference on Mobile and*

Ubiquitous Systems: Computing, Networking and Services, 2020.

[33] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2015.

[34] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *ArXiv preprint*, 2020.

[35] Hilmi E Egilmez, Eduardo Pavez, and Antonio Ortega. Graph learning from data under laplacian and structural constraints. *IEEE Journal of Selected Topics in Signal Processing*, (6), 2017.

[36] Negin Entezari, Saba A. Al-Sayouri, Amirali Darvishzadeh, and Evangelos E. Papalexakis. All you need is low (rank): Defending against adversarial attacks on graphs. In *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, 2020.

[37] Negin Entezari, Saba A. Al-Sayouri, Amirali Darvishzadeh, and Evangelos E. Papalexakis. All you need is low (rank): Defending against adversarial attacks on graphs. In *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, 2020.

[38] Wenqi Fan, Xiaorui Liu, Wei Jin, Xiangyu Zhao, Jiliang Tang, and Qing Li. Graph trend filtering networks for recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2022.

[39] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Yihong Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, 2019.

[40] Reza Zanjirani Farahani and Masoud Hekmatfar. *Facility location: concepts, models, algorithms and case studies*. 2009.

[41] Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. Slaps: Self-supervision improves structure learning for graph neural networks. *Advances in Neural Information Processing Systems*, 2021.

[42] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[43] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *ArXiv preprint*, 2019.

[44] Santo Fortunato. Community detection in graphs. *Physics reports*, 2010.

[45] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, Proceedings of Machine Learning Research, 2019.

[46] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, (1), 2016.

[47] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, Proceedings of Machine Learning Research, 2019.

[48] Simon Geisler, Tobias Schmidt, Hakan Şirin, Daniel Zügner, Aleksandar Bojchevski, and Stephan Günnemann. Robustness of graph neural networks at scale. *Advances in Neural Information Processing Systems*, 2021.

[49] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, Proceedings of Machine Learning Research, 2017.

[50] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, Proceedings of Machine Learning Research, 2017.

[51] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*. IEEE, 2018.

[52] Shurui Gui, Xiner Li, Limei Wang, and Shuiwang Ji. Good: A graph out-of-distribution benchmark. *ArXiv preprint*, 2022.

[53] Zhichun Guo, Bozhao Nan, Yijun Tian, Olaf Wiest, Chuxu Zhang, and Nitesh V Chawla. Graph-based molecular representation learning. *ArXiv preprint*, 2022.

[54] Zhichun Guo, William Shiao, Shichang Zhang, Yozen Liu, Nitesh Chawla, Neil Shah, and Tong Zhao. Linkless link prediction via relational distillation. *arXiv preprint arXiv:2210.05801*, 2022.

[55] Zhichun Guo, Chuxu Zhang, Wenhao Yu, John Herr, Olaf Wiest, Meng Jiang, and Nitesh V Chawla. Few-shot graph learning for molecular property prediction. In *Proceedings of the Web Conference 2021*, 2021.

[56] Jonathan Halcrow, Alexandru Mosoi, Sam Ruth, and Bryan Perozzi. Grale: Designing networks for graph learning. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, 2020.

[57] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017.

[58] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017.

[59] Kaveh Hassani and Amir Hosein Khas Ahmadi. Contrastive multi-view representation learning on graphs. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, Proceedings of Machine Learning Research, 2020.

[60] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[61] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. Combining label propagation and simple models out-performs graph neural networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[62] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. Scaling up graph neural networks via graph coarsening. In *In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21)*, 2021.

[63] Joseph J. Pfeiffer III, Sebastián Moreno, Timothy La Fond, Jennifer Neville, and Brian Gallagher. Attributed graph models: modeling network structure with correlated attributes. In *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, 2014.

[64] Bo Jiang, Ziyan Zhang, Doudou Lin, Jin Tang, and Bin Luo. Semi-supervised learning with graph learning-convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 2019.

[65] Wei Jin, Tyler Derr, Haochen Liu, Yiqi Wang, Suhang Wang, Zitao Liu, and Jiliang Tang. Self-supervised learning on graphs: Deep insights and new direction. *ArXiv preprint*, 2020.

[66] Wei Jin, Tyler Derr, Yiqi Wang, Yao Ma, Zitao Liu, and Jiliang Tang. Node similarity preserving graph convolutional networks. In *Proceedings of the 14th ACM international conference on web search and data mining*, 2021.

[67] Wei Jin, Yaxing Li, Han Xu, Yiqi Wang, Shuiwang Ji, Charu Aggarwal, and Jiliang Tang. Adversarial attacks and defenses on graphs. *ACM SIGKDD Explorations Newsletter*, (2), 2021.

[68] Wei Jin, Xiaorui Liu, Yao Ma, Charu Aggarwal, and Jiliang Tang. Feature overcorrelation in deep graph neural networks: A new perspective. In *KDD*, 2022.

[69] Wei Jin, Xiaorui Liu, Xiangyu Zhao, Yao Ma, Neil Shah, and Jiliang Tang. Automated self-supervised learning for graphs. In *International Conference on Learning Representations*, 2022.

[70] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, 2020.

[71] Wei Jin, Xianfeng Tang, Haoming Jiang, Zheng Li, Danqing Zhang, Jiliang Tang, and Bing Yin. Condensing graphs via one-step gradient matching. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

[72] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. Graph condensation for graph neural networks. In *ICLR 2022*, 2022.

[73] Wei Jin, Tong Zhao, Jiayuan Ding, Yozen Liu, Jiliang Tang, and Neil Shah. Empowering graph representation learning with test-time graph transformation. In *The Eleventh International Conference on Learning Representations*, 2023.

[74] David R Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, (2), 1999.

[75] KrishnaTeja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, Abir De, and Rishabh K. Iyer. GRAD-MATCH: gradient matching based data subset selection for efficient deep model training. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, Proceedings of Machine Learning Research, 2021.

[76] Jang-Hyun Kim, Jinuk Kim, Seong Joon Oh, Sangdoo Yun, Hwanjun Song, Joonhyun Jeong, Jung-Woo Ha, and Hyun Oh Song. Dataset condensation via efficient synthetic-data

parameterization. *ArXiv preprint*, 2022.

[77] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *ArXiv preprint*, 2016.

[78] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[79] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[80] Vladimir Koltchinskii, Karim Lounici, Alexandre B Tsybakov, et al. Nuclear-norm penalization and optimal rates for noisy low-rank matrix completion. *The Annals of Statistics*, (5), 2011.

[81] Saehyung Lee, Sanghyuk Chun, Sangwon Jung, Sangdoo Yun, and Sungroh Yoon. Dataset condensation with contrastive signals. In *ICML*, 2022.

[82] Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, 2019.

[83] Haoyang Li, Xin Wang, Ziwei Zhang, and Wenwu Zhu. Out-of-distribution generalization on graphs: A survey. *ArXiv preprint*, 2022.

[84] Jintang Li, Tao Xie, Chen Liang, Fenfang Xie, Xiangnan He, and Zibin Zheng. Adversarial attack on large scale graph. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[85] Yaxin Li, Wei Jin, Han Xu, and Jiliang Tang. Deeprobust: A pytorch library for adversarial attacks and defenses, 2020.

[86] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[87] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, (12), 2017.

[88] Weixin Liang, Girmaw Abebe Tadesse, Daniel Ho, L Fei-Fei, Matei Zaharia, Ce Zhang, and James Zou. Advances, challenges and opportunities in creating data for trustworthy ai. *Nature Machine Intelligence*, (8), 2022.

[89] Derek Lim, Xiuyu Li, Felix Hohne, and Ser-Nam Lim. New benchmarks for learning on non-homophilous graphs. *ArXiv preprint*, 2021.

[90] Gang Liu, Eric Inae, Tong Zhao, Jiaxin Xu, Tengfei Luo, and Meng Jiang. Data-centric learning from unlabeled graphs with diffusion model. *arXiv preprint arXiv:2303.10108*, 2023.

[91] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[92] Hongrui Liu, Binbin Hu, Xiao Wang, Chuan Shi, Zhiqiang Zhang, and Jun Zhou. Confidence may cheat: Self-training on graph neural networks under distribution shift. In *Proceedings of the ACM Web Conference 2022*, 2022.

[93] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, 2020.

[94] Meng Liu, Youzhi Luo, Kanji Uchino, Koji Maruhashi, and Shuiwang Ji. Generating 3d molecules for target protein binding. In *ICML*, 2022.

[95] Sijia Liu, Swarnendu Kar, Makan Fardad, and Pramod K Varshney. Sparsity-aware sensor collaboration for linear coherent estimation. *IEEE Transactions on Signal Processing*, (10), 2015.

[96] Xiaorui Liu, Jiayuan Ding, Wei Jin, Han Xu, Yao Ma, Zitao Liu, and Jiliang Tang. Graph neural networks with adaptive residual. In *Advances in Neural Information Processing Systems*, 2021.

[97] Xiaorui Liu, Wei Jin, Yao Ma, Yaxin Li, Hua Liu, Yiqi Wang, Ming Yan, and Jiliang Tang. Elastic graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, Proceedings of Machine Learning Research, 2021.

[98] Xuanqing Liu, Si Si, Jerry Zhu, Yang Li, and Cho-Jui Hsieh. A unified framework for data poisoning attack to graph-based semi-supervised learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 2019.

[99] Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip Yu. Graph self-supervised learning: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[100] Yuejiang Liu, Parth Kothari, Bastien van Delft, Baptiste Bellot-Gurlet, Taylor Mordan, and Alexandre Alahi. Ttt++: When does self-supervised test-time training fail or thrive? *Advances in Neural Information Processing Systems*, 2021.

[101] Andreas Loukas. Graph reduction with spectral and cut guarantees. *J. Mach. Learn. Res.*, (116), 2019.

[102] Andreas Loukas and Pierre Vandergheynst. Spectrally approximating large graphs with smaller graphs. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, Proceedings of Machine Learning Research, 2018.

[103] Xiaojun Ma, Ziyao Li, Lingjun Xu, Guojie Song, Yi Li, and Chuan Shi. Learning discrete adaptive receptive fields for graph convolutional networks, 2021.

[104] Yao Ma, Xiaorui Liu, Tong Zhao, Yozen Liu, Jiliang Tang, and Neil Shah. A unified view on graph neural networks as graph signal denoising. *ArXiv preprint*, 2020.

[105] Yao Ma, Suhang Wang, Lingfei Wu, and Jiliang Tang. Attacking graph convolutional networks via rewiring. *ArXiv preprint*, 2019.

[106] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[107] Haitao Mao, Lun Du, Yujia Zheng, Qiang Fu, Zelin Li, Xu Chen, Shi Han, and Dongmei Zhang. Source free unsupervised graph domain adaptation. *arXiv preprint arXiv:2112.00955*, 2021.

[108] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, (1), 2001.

[109] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *ArXiv preprint*, 2020.

[110] Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset meta-learning from kernel ridge-regression. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[111] Timothy Nguyen, Roman Novak, Lechao Xiao, and Jaehoon Lee. Dataset distillation with infinitely wide convolutional networks. *NeurIPS*, 2021.

[112] Yi Nian, Wei Jin, and Lu Lin. In-process global interpretation for graph learning via distribution matching. *arXiv preprint arXiv:2306.10447*, 2023.

[113] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. In *Proceedings of the Twenty-Seventh*

International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, 2018.

[114] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, 2020.

[115] David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of graph theory*, (1), 1989.

[116] Neoklis Polyzotis and Matei Zaharia. What can data-centric ai learn from data and ml engineering? *arXiv preprint arXiv:2112.06439*, 2021.

[117] Hugo Raguet, Jalal Fadili, and Gabriel Peyré. A generalized forward-backward splitting. *SIAM Journal on Imaging Sciences*, (3), 2013.

[118] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017.

[119] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[120] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, (2), 2021.

[121] Benedek Rozemberczki, Peter Englert, Amol Kapoor, Martin Blais, and Bryan Perozzi. Pathfinder discovery networks for neural message passing. In *Proceedings of the Web Conference 2021*, 2021.

[122] Alan Said, Ernesto W De Luca, and Sahin Albayrak. How social relationships affect user similarities.

[123] Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. Graph neural networks for friend ranking in large-scale social platforms. In *Proceedings of the Web Conference 2021*, 2021.

[124] Pierre-André Savalle, Emile Richard, and Nicolas Vayatis. Estimation of simultaneously sparse and low rank matrices. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012.

[125] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A

core-set approach. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[126] Cosma Rohilla Shalizi and Andrew C Thomas. Homophily and contagion are generically confounded in observational social network studies. *Sociological methods & research*, (2), 2011.

[127] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *ArXiv preprint*, 2018.

[128] Xiao Shen, Quanyu Dai, Fu-Lai Chung, Wei Lu, and Kup-Sze Choi. Adversarial deep network embedding for cross-network node classification. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, 2020.

[129] Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, (4), 2011.

[130] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, Proceedings of Machine Learning Research, 2020.

[131] Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. Adversarial graph augmentation to improve graph contrastive learning. *Advances in Neural Information Processing Systems*, 2021.

[132] Wenzhuo Tang, Hongzhi Wen, Renming Liu, Jiayuan Ding, Wei Jin, Yuying Xie, Hui Liu, and Jiliang Tang. Single-cell multimodal prediction via transformers. *arXiv preprint arXiv:2303.00233*, 2023.

[133] Xianfeng Tang, Yandong Li, Yiwei Sun, Huaxiu Yao, Prasenjit Mitra, and Suhang Wang. Robust graph neural network against poisoning attacks via transfer learning. *ArXiv preprint*, 2019.

[134] Xianfeng Tang, Yandong Li, Yiwei Sun, Huaxiu Yao, Prasenjit Mitra, and Suhang Wang. Transferring robustness for graph neural network against poisoning attacks. In *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, 2020.

[135] Xianfeng Tang, Yozen Liu, Neil Shah, Xiaolin Shi, Prasenjit Mitra, and Suhang Wang. Knowing your FATE: friendship, action and temporal explanations for user engagement prediction on social apps. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, 2020.

[136] Amanda L Traud, Peter J Mucha, and Mason A Porter. Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, (16), 2012.

[137] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, (11), 2008.

[138] Liwen Vaughan, Margaret EI Kipp, and Yijun Gao. Why are websites co-linked? the case of canadian universities. *Scientometrics*, (1), 2007.

[139] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[140] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[141] Daixin Wang, Jianbin Lin, Peng Cui, Quanhui Jia, Zhen Wang, Yanming Fang, Quan Yu, Jun Zhou, Shuang Yang, and Yuan Qi. A semi-supervised graph attentive network for financial fraud detection. In *ICDM*, 2019.

[142] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno A. Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[143] Kai Wang, Bo Zhao, Xiangyu Peng, Zheng Zhu, Shuo Yang, Shuo Wang, Guan Huang, Hakan Bilen, Xinchao Wang, and Yang You. Cafe: Learning to condense dataset by aligning features. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[144] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *ArXiv preprint*, 2018.

[145] Yu Wang, Wei Jin, and Tyler Derr. Graph neural networks: Self-supervised learning. In *Graph Neural Networks: Foundations, Frontiers, and Applications*. 2022.

[146] Yu Wang, Yuying Zhao, Yushun Dong, Huiyuan Chen, Jundong Li, and Tyler Derr. Improving fairness in graph neural networks via mitigating sensitive attribute leakage. In *KDD*, 2022.

[147] Jeremy Watt, Reza Borhani, and Aggelos K Katsaggelos. *Machine learning refined: Foundations, algorithms, and applications*. 2020.

[148] Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada,*

June 14-18, 2009, ACM International Conference Proceeding Series, 2009.

[149] Hongzhi Wen, Jiayuan Ding, Wei Jin, Yiqi Wang, Yuying Xie, and Jiliang Tang. Graph neural networks for multimodal single-cell data integration. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pages 4153–4163, 2022.

[150] Steven Euijong Whang, Yuji Roh, Hwanjun Song, and Jae-Gil Lee. Data collection and quality challenges in deep learning: A data-centric ai perspective. *The VLDB Journal*, 2023.

[151] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, Proceedings of Machine Learning Research, 2019.

[152] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples for graph data: Deep insights into attack and defense. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 2019.

[153] Man Wu, Shirui Pan, Chuan Zhou, Xiaojun Chang, and Xingquan Zhu. Unsupervised domain adaptive graph convolutional networks. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, 2020.

[154] Qitian Wu, Hengrui Zhang, Junchi Yan, and David Wipf. Handling distribution shifts on graphs: An invariance perspective. In *International Conference on Learning Representations*, 2022.

[155] Yingxin Wu, Xiang Wang, An Zhang, Xiangnan He, and Tat-Seng Chua. Discovering invariant rationales for graph neural networks. In *International Conference on Learning Representations*, 2022.

[156] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *ArXiv preprint*, 2019.

[157] Yaochen Xie, Zhao Xu, Jingtun Zhang, Zhengyang Wang, and Shuiwang Ji. Self-supervised learning of graph neural networks: A unified review. *ArXiv preprint*, 2021.

[158] Han Xu, Yao Ma, Haochen Liu, Debayan Deb, Hui Liu, Jiliang Tang, and Anil Jain. Adversarial attacks and defenses in images, graphs and text: A review. *ArXiv preprint*, 2019.

[159] Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. Topology attack and defense for graph neural networks: An optimization perspective. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 2019.

[160] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[161] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[162] Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. Generalized out-of-distribution detection: A survey. *ArXiv preprint*, 2021.

[163] Shuo Yang, Lu Liu, and Min Xu. Free lunch for few-shot learning: Distribution calibration. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[164] Shuo Yang, Songhua Wu, Tongliang Liu, and Min Xu. Bridging the gap between few-shot and many-shot learning via distribution calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (12), 2021.

[165] Shuo Yang, Zeke Xie, Hanyu Peng, Min Xu, Mingming Sun, and Ping Li. Dataset pruning: Reducing training data by examining generalization influence. *ArXiv preprint*, 2022.

[166] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, JMLR Workshop and Conference Proceedings, 2016.

[167] Rahul Yedida, Snehanshu Saha, and Tejas Prashanth. Lipschitzlr: Using theoretically computed adaptive learning rates for fast convergence. *Applied Intelligence*, (3), 2021.

[168] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, 2018.

[169] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, 2018.

[170] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2018.

[171] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, Proceedings of Machine Learning Research, 2021.

[172] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[173] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. When does self-supervision help graph convolutional networks? In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, Proceedings of Machine Learning Research, 2020.

[174] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. Graph domain adaptation via theory-grounded spectral regularization. In *International Conference on Learning Representations*, 2023.

[175] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in graph neural networks: A taxonomic survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[176] Werner Zellinger, Thomas Grubinger, Edwin Lughofer, Thomas Natschläger, and Susanne Saminger-Platz. Central moment discrepancy (CMD) for domain-invariant representation learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[177] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. Graphsaint: Graph sampling based inductive learning method. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[178] Daochen Zha, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, and Xia Hu. Data-centric ai: Perspectives and challenges. In *Proceedings of the 2023 SIAM International Conference on Data Mining (SDM)*, pages 945–948. SIAM, 2023.

[179] Daochen Zha, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, Zhimeng Jiang, Shaochen Zhong, and Xia Hu. Data-centric artificial intelligence: A survey. *arXiv preprint arXiv:2303.10158*, 2023.

[180] Angela Zhang, Lei Xing, James Zou, and Joseph C Wu. Shifting machine learning for healthcare from development to deployment and from models to data. *Nature Biomedical Engineering*, 2022.

[181] Marvin Zhang, Sergey Levine, and Chelsea Finn. Memo: Test time robustness via adaptation and augmentation. *ArXiv preprint*, 2021.

[182] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, 2018.

[183] Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less neural networks: Teaching old mlps new tricks via distillation. *arXiv preprint arXiv:2110.08727*, 2021.

[184] Yanci Zhang, Yutong Lu, Haitao Mao, Jiawei Huang, Cien Zhang, Xinyi Li, and Rui Dai. Company competition graph. *arXiv preprint arXiv:2304.00323*, 2023.

[185] Yifan Zhang, Bryan Hooi, Lanqing Hong, and Jiashi Feng. Self-supervised aggregation of diverse experts for test-agnostic long-tailed recognition. In *Advances in Neural Information Processing Systems*, 2022.

[186] Zhikun Zhang, Min Chen, Michael Backes, Yun Shen, and Yang Zhang. Inference attacks against graph neural networks. In *Proceedings of the 31th USENIX Security Symposium*, 2022.

[187] Bo Zhao and Hakan Bilen. Dataset condensation with differentiable siamese augmentation. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, Proceedings of Machine Learning Research, 2021.

[188] Bo Zhao and Hakan Bilen. Dataset condensation with distribution matching. *ArXiv preprint*, 2021.

[189] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[190] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *International Conference on Learning Representations*, 2022.

[191] Tong Zhao, Tianwen Jiang, Neil Shah, and Meng Jiang. A synergistic approach for graph anomaly detection with pattern mining and feature learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[192] Tong Zhao, Gang Liu, Stephan Günnemann, and Meng Jiang. Graph data augmentation for graph machine learning: A survey. *ArXiv preprint*, 2022.

[193] Tong Zhao, Gang Liu, Daheng Wang, Wenhao Yu, and Meng Jiang. Learning from counterfactual links for link prediction. In *International Conference on Machine Learning*. PMLR, 2022.

[194] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. In *AAAI*, 2021.

[195] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *ArXiv preprint*, 2018.

[196] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *ArXiv preprint*, 2018.

[197] Kaixiong Zhou, Ninghao Liu, Fan Yang, Zirui Liu, Rui Chen, Li Li, Soo-Hyun Choi, and Xia Hu. Adaptive label smoothing to regularize large-scale graph training. *ArXiv preprint*, 2021.

[198] Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. Auto-gnn: Neural architecture search of graph neural networks. *ArXiv preprint*, 2019.

[199] Ke Zhou, Hongyuan Zha, and Le Song. Learning social infectivity in sparse low-rank networks using multi-dimensional hawkes processes. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2013, Scottsdale, AZ, USA, April 29 - May 1, 2013*, JMLR Workshop and Conference Proceedings, 2013.

[200] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. Robust graph convolutional networks against adversarial attacks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, 2019.

[201] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. Robust graph convolutional networks against adversarial attacks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, 2019.

[202] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[203] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 2019.

[204] Meiqi Zhu, Xiao Wang, Chuan Shi, Houye Ji, and Peng Cui. Interpreting and unifying graph neural networks with an optimization framework. In *Proceedings of the Web Conference 2021*, 2021.

[205] Qi Zhu, Natalia Ponomareva, Jiawei Han, and Bryan Perozzi. Shift-robust gnns: Overcoming

the limitations of localized graph training data. *Advances in Neural Information Processing Systems*, 2021.

[206] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Qiang Liu, Shu Wu, and Liang Wang. Deep graph structure learning for robust representations: A survey. *ArXiv preprint*, 2021.

[207] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*, 2021.

[208] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, 2018.

[209] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, 2018.

[210] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[211] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[212] Daniel Zügner and Stephan Günnemann. Certifiable robustness and robust training for graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, 2019.

APPENDIX A

GRAPH CONDENSATION FOR GRAPH NEURAL NETWORKS

A.1 Datasets and Hyper-Parameters

A.1.1 Datasets

We evaluate the proposed framework on three transductive datasets, i.e., Cora, Citeseer [78] and Ogbn-arxiv [60], and two inductive datasets, i.e., Flickr [177] and Reddit [57]. Since all the datasets have public splits, we download them from PyTorch Geometric [43] and use those splits throughout the experiments. Dataset statistics are shown in Table A.1.

Table A.1: Dataset statistics. The first three are transductive datasets and the last two are inductive datasets.

Dataset	**#Nodes**	**#Edges**	**#Classes**	**#Features**	**Training/Validation/Test**
Cora	2,708	5,429	7	1,433	140/500/1000
Citeseer	3,327	4,732	6	3,703	120/500/1000
Ogbn-arxiv	169,343	1,166,243	40	128	90,941/29,799/48,603
Flickr	89,250	899,756	7	500	44,625/22312/22313
Reddit	232,965	57,307,946	210	602	15,3932/23,699/55,334

A.1.2 Hyper-Parameter Setting

Condensation Process. For DC, we tune the number of hidden layers in a range of $\{1, 2, 3\}$ and fix the number of hidden units to 256. We further tune the number of epochs for training DC in a range of $\{100, 200, 400\}$. For GCond, without specific mention, we adopt a 2-layer SGC [151] with 256 hidden units as the GNN used for gradient matching. The function g_Φ that models the relationship between $\mathbf{A}'$ and $\mathbf{X}'$ is implemented as a multi-layer perceptron (MLP). Specifically, we adopt a 3-layer MLP with 128 hidden units for small graphs (Cora and Citeseer) and 256 hidden units for large graphs (Flickr, Reddit and Ogbn-arxiv). We tune the training epoch for GCond in a range of $\{400, 600, 1000\}$. For GCond-X, we tune the number of hidden layers in a range of $\{1, 2, 3\}$ and fix the number of hidden units to 256. We further tune the number of epochs for training GCond-X in a range of $\{100, 200, 400\}$. We tune the learning rate for all the methods in

a range of $\{0.1, 0.01, 0.001, 0.0001\}$. Furthermore, we set δ to be $0.05, 0.05, 0.01, 0.01, 0.01$ for Citeseer, Cora, Ogbn-arxiv, Flickr and Reddit, respectively.

For the choices of condensation ratio r, we divide the discussion into two parts. The first part is about transductive datasets. For Cora and Citeseer, since their labeling rates are very small (5.2% and 3.6%, respectively), we choose r to be $\{25\%, 50\%, 100\%\}$ of the labeling rate. Thus, we finally choose $\{1.3\%, 2.6\%, 5.2\%\}$ for Cora and $\{0.9\%, 1.8\%, 3.6\%\}$ for Citeseer. For Ogbn-arxiv, we choose r to be $\{0.1\%, 0.5\%, 1\%\}$ of its labeling rate (53%), thus being $\{0.05\%, 0.25\%, 0.5\%\}$. The second part is about inductive datasets. As the nodes in the training graphs are all labeled in inductive datasets, we simply choose $\{0.1\%, 0.5\%, 0.1\%\}$ for Flickr and $0.05\%, 0.1\%, 0.2\%$ for Reddit.

Evaluation Process. During the evaluation process, we set dropout rate to be 0 and weight decay to be 0.0005 when training various GNNs. The number of epochs is set to 3000 for GAT while it is set to 600 for other models. The initial learning rate is set to 0.01.

Settings for Table 3 and Table 4. In both condensation stage and evaluation stage, we set the depth of GNNs to 2. During condensation stage, we set weight decay to 0, dropout to 0 and training epochs to 1000. During evaluation stage, we set weight decay to 0.0005, dropout to 0 and training epochs to 600.

A.1.3 Training Details of DC-Graph, GCond-X and GCond

DC-Graph: During the condensation stage, DC-Graph only leverages the node features to produce condensed node features $\mathbf{X}'$. During the training stage of evaluation, DC-Graph takes the condensed features $\mathbf{X}'$ together with an identity matrix as the graph structure to train a GNN. In the later test stage of evaluation, the GNN takes both test node features and test graph structure as input to make predictions for test nodes.

GCond-X: During the condensation stage, GCond-X leverages both the graph structure and node features to produce condensed node features $\mathbf{X}'$. During the training stage of evaluation, GCond-X takes the condensed features $\mathbf{X}'$ together with an identity matrix as the graph structure to train a GNN. In the later test stage of evaluation, the GNN takes both test node features and test

graph structure as input to make predictions for test nodes.

GCond: During the condensation stage, GCond leverages both the graph structure and node features to produce condensed graph data $(\mathbf{A}', \mathbf{X}')$. During the training stage of evaluation, GCond takes the condensed data $(\mathbf{A}', \mathbf{X}')$ to train a GNN. In the later test stage of evaluation, the GNN takes both test node features and test graph structure as input to make predictions for test nodes.

A.2 Algorithm

We show the detailed algorithm of GCond in Algorithm 1. In detail, we first set the condensed label set $\mathbf{Y}'$ to fixed values and initialize $\mathbf{X}'$ as node features randomly selected from each class. In each outer loop, we sample a GNN model initialization $\boldsymbol{\theta}$ from a distribution $P_{\boldsymbol{\theta}}$. Then, for each class we sample the corresponding node batches from $\mathcal{T}$ and $\mathcal{S}$, and calculate the gradient matching loss within each class. The sum of losses from different classes are used to update $\mathbf{X}'$ or Φ. After that we update the GNN parameters for $\tau_{\boldsymbol{\theta}}$ epochs. When finishing the updating of condensed graph parameters, we use $\mathbf{A}' = \mathrm{ReLU}(g_{\Phi}(\mathbf{X}') - \delta)$ to obtain the final sparsified graph structure.

A.3 More Related Work

Graph pooling. Graph pooling [182, 170, 47] also generates a coarsened graph with a smaller size. The work [182] is one the first to propose an end-to-end architecture for graph classification by incorporating graph pooling. Later, DiffPool [170] proposes to use GNNs to parameterize the process of node grouping. However, those methods are majorly tailored for the graph classification task and the coarsened graphs are a byproduct graph during the representation learning process.

A.4 More Experiments

A.4.1 Ablation Study

Different Parameterization. We study the effect of different parameterizations for modeling $\mathbf{A}'$ and compare GCond with modeling $\mathbf{A}'$ as free parameters in Table A.2. From the table, we observe a significant improvement by taking into account the relationship between $\mathbf{A}'$ and $\mathbf{X}'$. This suggests that directly modeling the structure as a function of features can ease the optimization and lead to better condensed graph data.

Algorithm 2 GCond for Graph Condensation

Input: Training data $\mathcal{T} = (\mathbf{A}, \mathbf{X}, \mathbf{Y})$, pre-defined condensed labels $\mathbf{Y}'$
Initialize $\mathbf{X}'$ as node features randomly selected from each class
for $k = 0, \ldots, K-1$ **do**
  Initialize $\boldsymbol{\theta}_0 \sim P_{\boldsymbol{\theta}_0}$
  for $t = 0, \ldots, T-1$ **do**
    $D' = 0$
    for $c = 0, \ldots, C-1$ **do**
      Compute $\mathbf{A}' = g_\Phi(\mathbf{X}')$; then $\mathcal{S} = \{\mathbf{A}', \mathbf{X}', \mathbf{Y}'\}$
      Sample $(\mathbf{A}_c, \mathbf{X}_c, \mathbf{Y}_c) \sim \mathcal{T}$ and $(\mathbf{A}'_c, \mathbf{X}'_c, \mathbf{Y}'_c) \sim \mathcal{S}$ ▷ detailed in Section 3.1
      Compute $\mathcal{L}^{\mathcal{T}} = \mathcal{L}\left(\mathrm{GNN}_{\boldsymbol{\theta}_t}(\mathbf{A}_c, \mathbf{X}_c), \mathbf{Y}_c\right)$ and $\mathcal{L}^{\mathcal{S}} = \mathcal{L}\left(\mathrm{GNN}_{\boldsymbol{\theta}_t}(\mathbf{A}'_c, \mathbf{X}'_c), \mathbf{Y}'_c\right)$
      $D' \leftarrow D' + D(\nabla_{\boldsymbol{\theta}_t}\mathcal{L}^{\mathcal{T}}, \nabla_{\boldsymbol{\theta}_t}\mathcal{L}^{\mathcal{S}})$
    end
    if $t\%(\tau_1 + \tau_2) < \tau_1$ **then**
      Update $\mathbf{X}' \leftarrow \mathbf{X}' - \eta_1 \nabla_{\mathbf{X}'} D'$
    end
    else
      Update $\Phi \leftarrow \Phi - \eta_2 \nabla_\Phi D'$
    end
    Update $\boldsymbol{\theta}_{t+1} \leftarrow \mathrm{opt}_{\boldsymbol{\theta}}(\boldsymbol{\theta}_t, \mathcal{S}, \tau_{\boldsymbol{\theta}})$ ▷ $\tau_{\boldsymbol{\theta}}$ is the number of steps for updating $\boldsymbol{\theta}$
  end
end
$\mathbf{A}' = g_\Phi(\mathbf{X}')$
$\mathbf{A}'_{ij} = \mathbf{A}'_{ij}$ if $\mathbf{A}'_{ij} > \delta$, otherwise 0
Return: $(\mathbf{A}', \mathbf{X}', \mathbf{Y}')$

Table A.2: Ablation study on different parametrizations.

Parameters	Citeseer, r=1.8%	Cora, r=2.6%	Ogbn-arxiv, r=0.25%
$\mathbf{A}'$, $\mathbf{X}'$	62.2±4.8	75.5±0.6	63.0±0.5
Φ, $\mathbf{X}'$	70.6±0.9	80.1±0.6	63.2±0.3

Joint optimization versus alternate optimization. We perform the ablation study on joint optimization and alternate optimization when updating Φ and $\mathbf{X}'$. The results are shown in Table A.3. From the table, we can observe that joint optimization always gives worse performance and the standard deviation is much higher than alternate optimization.

A.4.2 Neural Architecture Search

We focus on APPNP instead of GCN since the architecture of APPNP involves more hyper-parameters regarding its architecture setup. The detailed search space is shown as follows:

Table A.3: Ablation study on different optimization strategies.

	Citeseer, r=1.8%	Cora, r=2.6%	Ogbn-arxiv, r=0.25%	Flickr, r=0.5%	Reddit, r=0.1%
Joint	68.2±3.0	79.9±1.6	62.8±1.1	45.4±0.4	87.5±1.8
Alternate	70.6±0.9	80.1±0.6	63.2±0.3	47.1±0.1	89.5±0.8

(a) **Number of propagation K:** we search the number of propagation K in the range of $\{2, 4, 6, 8, 10\}$.

(b) **Residual coefficient α:** for the residual coefficient in APPNP, we search it in the range of $\{0.1, 0.2\}$.

(c) **Hidden dimension:** We collect the set of dimensions that are widely adopted by existing work as the candidates, i.e., {16,32,64,128,256,512}.

(d) **Activation function:** The set of available activation functions is listed as follows: {Sigmoid, Tanh, ReLU, Linear, Softplus, LeakyReLU, ReLU6, ELU}

In total, for each dataset we search 480 architectures of APPNP and we perform the search process on Cora, Citeseer and Ogbn-arxiv. Specifically, we train each architecture on the reduced graph for epochs on as the model converges faster on the smaller graph. We use the best validation accuracy to choose the final architecture. We report (1) the Pearson correlation between validation accuracies obtained by architectures trained on condensed graphs and those trained on original graphs, and (2) the average test accuracy of the searched architecture over 20 runs.

Table A.4: Neural Architecture Search. Methods are compared in validation accuracy correlation and test accuracy obtained by searched architecture. Whole means the architecture is searched using whole dataset.

	Pearson Correlation			Test Accuracy			
	Random	Herding	GCond	Random	Herding	GCond	Whole
Cora	0.40	0.21	**0.76**	82.9	82.9	83.1	82.6
Citeseer	0.56	0.29	**0.79**	71.4	71.3	71.3	71.6
Ogbn-arxiv	0.63	0.60	**0.64**	71.1	71.2	71.2	71.9

A.4.3 Time Complexity and Running Time

Time Complexity. We start from analyzing the time complexity of calculating gradient matching loss, i.e., line 8 to line 11 in Algorithm 1. Let the number of MLP layers in g_Φ be L and r be the

number of sampled neighbors per node. For simplicity, we assume all the hidden units are d for all layers and we use L-layer GCN for the analysis. The forward process of g_Φ has a complexity of $O(N'^2d^2)$. The forward process of GCN on the original graph has a complexity of $O(r^L Nd^2)$ and that on condensed graph has a complexity of $O(LN'^2d + LN'd)$. The complexity of calculating the second-order derivatives in backward propagation has an additional factor of $O(|\boldsymbol{\theta}_t||\mathbf{A}'| + |\boldsymbol{\theta}_t||\mathbf{X}'|)$, which can be reduced to $O(|\boldsymbol{\theta}_t| + |\mathbf{A}'| + |\mathbf{X}'|)$ with finite difference approximation. Although there are C iterations in line 7-11, we note that the process is easily parallelizable. Furthermore, the process of updating $\boldsymbol{\theta_t}$ in line 16 has a complexity of $\tau_{\boldsymbol{\theta}}(LN'^2d + LN'd)$. Considering there are T iterations and K different initializations, we multiply the aforementioned complexity by KT. To sum up, we can see that the time complexity linearly increases with number of nodes in the original graph.

Running Time. We now report the running time of the proposed GCond for different condensation rates. Specifically, we vary the condensation rates in the range of {0.1%, 0.5%, 1%} on Ogbn-arxiv and {1%, 5%, 10%} on Cora. The running time of 50 epochs on one single A100-SXM4 GPU is reported in Table A.5.The whole condensation process (1000 epochs) for generating 0.5% condensed graph of Ogbn-arxiv takes around 2.4 hours, which is an acceptable cost given the huge benefits of the condensed graph.

Table A.5: Running time of GCond for 50 epochs.

r	0.1%	0.5%	1%	r	1%	5%	10%
Ogbn-arxiv	348.6s	428.2s	609.8s	Cora	37.4s	43.9s	64.8s

A.4.4 Sparsification

In this subsection, we investigate the effect of threshold δ on the test accuracy and sparsity. In detail, we vary the values of the threshold δ used for truncating adjacency matrix in a range of $\{0.01, 0.05, 0.1, 0.2, 0.4, 0.6, 0.8\}$, and report the corresponding test accuracy and sparsity in Figure A.1. From the figure, we can see that increasing δ can effectively increase the sparsity of the obtained adjacency matrix without affecting the performance too much.

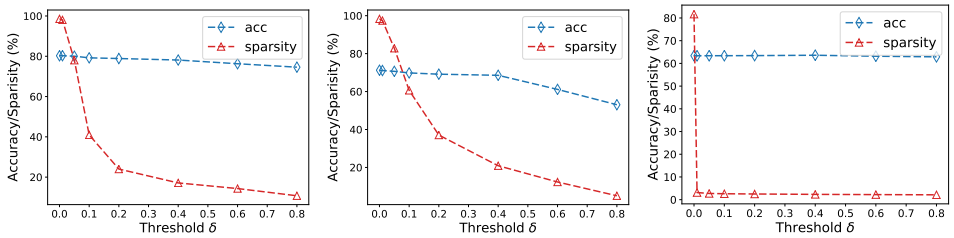
Table A.6: Test accuracy on different numbers of hidden units (H) and layers (L). When L=1, there is no hidden layer so the number of hidden units is meaningless.

(a) Cora, r=2.6%

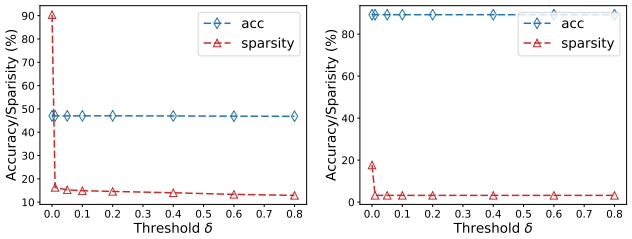
H\L	1	2	3	4
16	74.8±0.5	76.8±1.0	68.0±3.0	50.9±9.5
32	-	79.2±0.7	70.4±3.2	61.1±7.2
64	-	79.2±1.0	72.0±3.3	64.5±2.2
128	-	79.9±0.3	76.6±1.8	61.8±3.8
256	-	80.1±0.6	75.9±1.6	65.6±2.9

(b) Citeseer, r=1.8%

H\L	1	2	3	4
16	58.6±12.1	69.2±1.3	56.9±8.4	40.4±1.2
32	-	69.4±1.3	59.9±10.2	42.6±3.6
64	-	69.7±1.5	62.3±10.3	43.6±3.7
128	-	70.2±1.4	63.3±9.7	51.6±1.8
256	-	70.6±0.9	63.5±10.0	52.9±5.5



(a) Cora, r=2.5% (b) Citeseer, r=1.8% (c) Ogbn-arxiv, r=0.05%



(d) Flickr, r=0.1% (e) Reddit, r=0.1%

Figure A.1: Test accuracy and sparsity under different values of δ.

A.4.5 Different Depth and Hidden Units

Depth Versus Hidden Units. We vary the number of model layers (GCN) in a range of $\{1, 2, 3, 4\}$ and the number of hidden units in a range of $\{16, 32, 64, 128, 256\}$, and test them on the condensed graphs of Cora and Citeseer. The results are reported in Table A.6. From the table, we can observe that changing the number of layers impacts the model performance a lot while changing the number of units does not.

Propagation Versus Transformation. We further study the effect of propagation and transformation on the condensed graph. We use Cora as an example and use SGC as the test model due to its decoupled architecture. Specifically, we vary both the propagation layers and transformation layers of SGC in the range of $\{1, 2, 3, 4, 5\}$, and report the performance in Table A.7. As can be seen, the

condensed graph still achieves good performance with 3 and 4 layers of propagation. Although the condensed graph is generated under 2-layer SGC, it is able to generalize to 3-layer and 4-layer SGC. When increasing the propagation to 5, the performance degrades a lot which could be the cause of the oversmoothing issue. On the other hand, stacking more transformation layers can first help boost the performance but then hurt. Given the small scale of the graph, SGC suffers the overfitting issue in this case.

Table A.7: Test accuracy of SGC on different transformations and propagations for Cora, r=2.6%

Trans\Prop	1	2	3	4	5
1	77.09±0.43	79.02±1.17	78.12±2.13	74.04±3.60	61.19±7.73
2	76.94±0.50	79.01±0.57	79.11±1.15	77.57±1.03	72.37±4.25
3	75.28±0.58	77.95±0.67	74.16±1.50	70.58±3.71	58.28±8.90
4	66.87±0.73	66.54±0.82	59.24±1.60	43.94±6.33	30.45±9.67
5	46.44±0.91	37.29±3.23	16.05±2.74	15.33±2.79	15.33±2.79

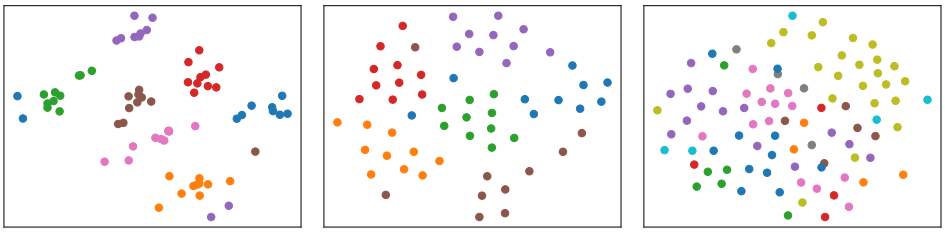
Table A.8: Cross-depth accuracy on Cora, r=2.6%

C\T	2	3	4	5	6
2	80.30	80.70	79.46	76.06	71.23
3	40.62	72.37	40.14	67.19	35.02
4	74.24	72.56	76.26	71.70	65.12
5	71.31	75.73	70.95	73.13	67.12
6	75.20	75.18	75.67	76.16	75.00

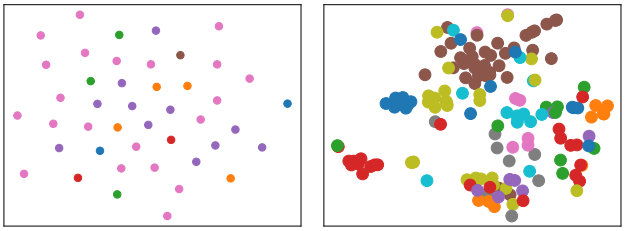
Cross-depth Performance. We show the cross-depth performance in Table A.8. Specifically, we use SGC of different depth in the condensation to generate condensed graphs and then use them to test on SGC of different depth. Note that in this table, we set weight decay to 0 and dropout to 0.5. We can observe that using a deeper GNN is not always helpful. Stacking more layers do not necessarily mean we can learn better condensed graphs since more nodes are involved during the optimization, and this makes optimization more difficult.

A.4.6 Performances on Original Graphs

We show the performances of various GNNs on original graphs in Table A.9 as references. Particularly, we report the performances of five GNN models including GAT, Cheby, SAGE, SGC, APPNP, and GCN, on the five datasets including Cora, Citeseer, Ogbn-arxiv, Flickr, and Reddit.



(a) Cora, r=2.5% (b) Citeseer, r=1.8% (c) Arxiv, r=0.05%



(d) Flickr, r=0.1% (e) Reddit, r=0.1%

Figure A.2: The t-SNE plots of node features in condensed graphs.

Table A.9: Performances of various GNNs on original graphs. SAGE: GraphSAGE.

	GAT	Cheby	SAGE	SGC	APPNP	GCN
Cora	83.1	81.4	81.2	81.4	83.1	81.2
Citeseer	70.8	70.2	70.1	71.3	71.8	71.7
Ogbn-arxiv	71.5	71.4	71.5	71.4	71.2	71.7
Flickr	44.3	47.0	46.1	46.2	47.3	47.1
Reddit	91.0	93.1	93.0	93.5	94.3	93.9

A.4.7 Visualization of Node Features

In addition, we provide the t-SNE [137] plots of condensed node features to further understand the condensed graphs. In Cora and Citeseer, the condensed node features are well clustered. For Ogbn-arxiv and Reddit, we also observe some clustered pattern for the nodes within the same class. In contrast, the condensed features are less discriminative in Flickr, which indicates that the condensed structure information can be essential in training GNN.

A.4.8 Experiments on Pubmed

We also show the experiments for Pubmed with a condensation ratio of 0.3% in Table A.10. From the table, we can observe that GCond approximates the original performance very well (77.92% vs. 79.32% on GCN). It also generalizes well to different architectures including APPNP, Cheby, GCN, GraphSage, and SGC. Furthermore, it outperforms DC-Graph, indicating that it is

important to leverage the graph structure information; it outperforms GCond-X, indicating that learning a condensed structure is necessary.

Table A.10: Performance of different GNNs on Pubmed (r=0.3%).

	APPNP	Cheby	GCN	GraphSage	SGC
DC-Graph	72.76±1.39	72.66±0.59	72.44±2.90	71.96±2.50	75.43±0.65
GCond-X	73.91±0.41	74.57±1.00	71.81±0.94	73.10±2.08	76.72±0.65
GCond	76.77±1.17	75.48±0.82	77.92±0.42	71.12±3.10	75.91±1.38

APPENDIX B

CONDENSING GRAPHS VIA ONE-STEP GRADIENT MATCHING

B.1 Proofs

B.1.1 Proof of Theorem 1

Let $\mathbf{A}_{(i)}, \mathbf{X}_{(i)}$ denote the adjacency matrix and the feature matrix of i-th real graph, respectively. We denote the cross entropy loss on the real samples as $\ell_{\mathcal{T}}(\theta) = \sum_i \ell_i(\mathbf{A}_{(i)}, \mathbf{X}_{(i)}, \theta)$ and denote that on synthetic samples as $\ell_S(\theta) = \ell_S(\mathbf{A}'_{(i)}, \mathbf{X}'_{(i)}, \theta)$. Let θ^* denote the optimal parameter and let θ_t be the parameter trained on condensed data at t-th epoch by optimizing $\ell_S(\theta)$. For simplicity of notations, we assume $\mathbf{A}$ and $\mathbf{A}'$ are already normalized. Part of the proof is inspired from the work [75].

Theorem 1. *When we use a linearized K-layer SGC as the GNN used in condensation, i.e., $f_\theta(\mathbf{A}_{(i)}, \mathbf{X}_{(i)}) = Pool(\mathbf{A}^K_{(i)}\mathbf{X}_{(i)}\mathbf{W}_1)\mathbf{W}_2$ with $\theta = [\mathbf{W}_1; \mathbf{W}_2]$ and assume that all network parameters satisfy $\|\theta\|^2 \leq M^2 (M > 0)$, we have*

$$\min_{t=0,1,\ldots,T-1} \ell_{\mathcal{T}}(\theta_t) - \ell_{\mathcal{T}}(\theta^*) \leq \sum_{t=0}^{T-1} \frac{\sqrt{2}M}{T} \|\nabla_\theta \ell_{\mathcal{T}}(\theta_t) - \nabla_\theta \ell_S(\theta_t)\| + \frac{3M}{2\sqrt{T}} \cdot \frac{C-1}{CN'} \sqrt{\sum_i \gamma_i \|\mathbf{1}^\top \mathbf{A}'^K_{(i)} \mathbf{X}'_{(i)}\|^2} \tag{B.1}$$

where $\gamma_i = 1$ if we use sum pooling in f_θ; $\gamma_i = \frac{1}{n_i}$ if we use mean pooling, with n_i being the number of nodes in i-th synthetic graph.

Proof. We start by proving that $\ell_{\mathcal{T}}(\theta)$ is convex and $\ell_S(\theta)$ is lipschitz continuous when we use $f_\theta(\mathbf{A}_{(i)}, \mathbf{X}_{(i)}) = \text{Pool}(\mathbf{A}^K_{(i)}\mathbf{X}_{(i)}\mathbf{W}_1)\mathbf{W}_2$ as the mapping function. Before proving these two properties, we first rewrite $f_\theta(\mathbf{A}_{(i)}, \mathbf{X}_{(i)})$ as:

$$f_\theta(\mathbf{A}_{(i)}, \mathbf{X}_{(i)}) = \begin{cases} \mathbf{1}^\top \mathbf{A}^K_{(i)} \mathbf{X}_{(i)} \mathbf{W}_1 \mathbf{W}_2 & \text{if use sum pooling,} \\ \frac{1}{n_i} \mathbf{1}^\top \mathbf{A}^K_{(i)} \mathbf{X}_{(i)} \mathbf{W}_1 \mathbf{W}_2 & \text{if use mean pooling,} \end{cases} \tag{B.2}$$

where n is the number of nodes in $\mathbf{A}_{(i)}$ and $\mathbf{1}$ is an $n_i \times 1$ matrix filled with constant one. From the above equation we can see that f_θ with different pooling methods only differ in a multiplication factor $\frac{1}{n_i}$. Thus, in the following we focus on f_θ with sum pooling to derive the major proof.

I. For f_θ with sum pooling:

Substitute $\mathbf{W}$ for $\mathbf{W}_1\mathbf{W}_2$ and we have $f_\theta(\mathbf{A}_{(i)}, \mathbf{X}_{(i)}) = \mathbf{1}^\top \mathbf{A}^K_{(i)} \mathbf{X}_{(i)} \mathbf{W}$ for the case with sum pooling. Next we show that $\ell_\mathcal{T}(\theta)$ is convex and $\ell_S(\theta)$ is lipschitz continuous when we use $f_\theta(\mathbf{A}_{(i)}, \mathbf{X}_{(i)}) = \mathbf{1}^\top \mathbf{A}^K_{(i)} \mathbf{X}_{(i)} \mathbf{W}$ with $\theta = \mathbf{W}$.

(a) Convexity of $\ell_\mathcal{T}(\theta)$. From chapter 4 of the book [147], we know that softmax classification $f(\mathbf{W}) = \mathbf{X}W$ with cross entropy loss is convex w.r.t. the parameters $\mathbf{W}$. In our case, the mapping function $f_\theta(\mathbf{A}_{(i)}, \mathbf{X}_{(i)}) = \mathbf{1}^\top \mathbf{A}^K_{(i)} \mathbf{X}_{(i)} \mathbf{W}$ applies an affine function on $\mathbf{X}W$. Given that applying affine function does not change the convexity, we know that $\ell_\mathcal{T}(\theta)$ is convex.

(b) Lipschitz continuity of $\ell_S(\theta)$. In [167], it shows that the lipschitz constant of softmax regression with cross entropy loss is $\frac{C-1}{Cm}\|\mathbf{X}\|$, where $\mathbf{X}$ is the input feature matrix, C is the number of classes and m is the number of samples. Since $\ell_S(\theta)$ is cross entropy loss and f_θ is linear, we know that the f_θ is lipschitz continuous and it satisfies:

$$\nabla_\theta \ell_S(\theta) \leq \frac{C-1}{CN'} \sqrt{\sum_i \|\mathbf{1}^\top \mathbf{A}'^K_{(i)} \mathbf{X}'_{(i)}\|^2} \tag{B.3}$$

With (a) and (b), we are able to proceed our proof. First, from the convexity of $\ell_\mathcal{T}(\theta)$ we have

$$\ell_\mathcal{T}(\theta_t) - \ell_\mathcal{T}(\theta^*) \leq \nabla_\theta \ell_\mathcal{T}(\theta_t)^T (\theta_t - \theta^*) \tag{B.4}$$

We can rewrite $\nabla_\theta \ell_\mathcal{T}(\theta_t)^T (\theta_t - \theta^*)$ as follows:

$$\begin{aligned} &\nabla_\theta \ell_\mathcal{T}(\theta_t)^T (\theta_t - \theta^*) = (\nabla_\theta \ell_\mathcal{T}(\theta_t)^T - \nabla_\theta \ell_S(\theta_t)^T + \nabla_\theta \ell_S(\theta_t)^T)(\theta_t - \theta^*) \\ &= (\nabla_\theta \ell_\mathcal{T}(\theta_t)^T - \nabla_\theta \ell_S(\theta_t)^T)(\theta_t - \theta^*) + \nabla_\theta \ell_S(\theta_t)^T (\theta_t - \theta^*) \end{aligned} \tag{B.5}$$

Given that we use gradient descent to update network parameters, we have $\nabla_\theta \ell_S(\theta_t) =$

$\frac{1}{\eta}(\theta_t - \theta_{t+1})$ where η is the learning rate. Then we have,

$$\begin{aligned}\nabla_\theta \ell_S(\theta_t)^T(\theta_t - \theta^*) &= \frac{1}{\eta}(\theta_t - \theta_{t+1})^T(\theta_t - \theta^*) \\ &= \frac{1}{2\eta}\left(\|\theta_t - \theta_{t+1}\|^2 + \|\theta_t - \theta^*\|^2 - \|\theta_{t+1} - \theta^*\|^2\right) \\ &= \frac{1}{2\eta}\left(\|\eta\nabla_\theta \ell_S(\theta_t)\|^2 + \|\theta_t - \theta^*\|^2 - \|\theta_{t+1} - \theta^*\|^2\right) \end{aligned} \tag{B.6}$$

Combining Eq. (B.4) and Eq. (B.6) we have,

$$\begin{aligned}\ell_{\mathcal{T}}(\theta_t) - \ell_{\mathcal{T}}(\theta^*) \leq{}& (\nabla_\theta \ell_{\mathcal{T}}(\theta_t)^T - \nabla_\theta \ell_S(\theta_t)^T)(\theta_t - \theta^*) \\ &+ \frac{1}{2\eta}\left(\|\eta\nabla_\theta \ell_S(\theta_t)\|^2 + \|\theta_t - \theta^*\|^2 - \|\theta_{t+1} - \theta^*\|^2\right)\end{aligned} \tag{B.7}$$

We sum up the two sides of the above inequality for different values of $t \in [0, T-1]$:

$$\begin{aligned}\sum_{t=0}^{T-1} \ell_{\mathcal{T}}(\theta_t) - \ell_{\mathcal{T}}(\theta^*) \leq{}& \sum_{t=0}^{T-1}(\nabla_\theta \ell_{\mathcal{T}}(\theta_t)^T - \nabla_\theta \ell_S(\theta_t)^T)(\theta_t - \theta^*) \\ &+ \frac{1}{2\eta}\sum_{t=0}^{T-1}\|\eta\nabla_\theta \ell_S(\theta_t)\|^2 + \frac{1}{2\eta}\|\theta_0 - \theta^*\|^2 - \frac{1}{2\eta}\|\theta_T - \theta^*\|^2\end{aligned} \tag{B.8}$$

Since $\frac{1}{2\eta}\|\theta_T - \theta^*\|^2 \geq 0$, we have

$$\begin{aligned}\sum_{t=0}^{T-1} \ell_{\mathcal{T}}(\theta_t) - \ell_{\mathcal{T}}(\theta^*) \leq{}& \sum_{t=0}^{T-1}(\nabla_\theta \ell_{\mathcal{T}}(\theta_t)^T - \nabla_\theta \ell_S(\theta_t)^T)(\theta_t - \theta^*) \\ &+ \frac{1}{2\eta}\sum_{t=0}^{T-1}\|\eta\nabla_\theta \ell_S(\theta_t)\|^2 + \frac{1}{2\eta}\|\theta_0 - \theta^*\|^2\end{aligned} \tag{B.9}$$

As we assume that $\|\theta\|^2 \leq M^2$, we have $\|\theta - \theta^*\|^2 \leq 2\|\theta\|^2 = 2M^2$. Then Eq. (B.9) can be rewritten as,

$$\begin{aligned}\sum_{t=0}^{T-1} \ell_T(\theta_t) - \ell_T(\theta^*) \leq{}& \sum_{t=0}^{T-1}\sqrt{2}M\|\nabla_\theta \ell_T(\theta_t) - \nabla_\theta \ell_S(\theta_t)\| \\ &+ \frac{1}{2\eta}\sum_{t=0}^{T-1}\|\eta\nabla_\theta \ell_S(\theta_t)\|^2 + \frac{M^2}{\eta}\end{aligned} \tag{B.10}$$

Recall that $\ell_S(\theta)$ is lipschitz continuous as shown in Eq. (B.3), and combine the inequality $\min_{t=0,1,\ldots,T-1}\left(\ell_{\mathcal{T}}\left(\theta_t\right)-\ell_{\mathcal{T}}\left(\theta^*\right)\right) \leq \frac{\sum_{t=0}^{T-1}\ell_{\mathcal{T}}(\theta_t)-\ell_{\mathcal{T}}(\theta^*)}{T}$:

$$\begin{aligned}\min_{t=0,1,\ldots,T-1}\ell_{\mathcal{T}}\left(\theta_t\right)-\ell_{\mathcal{T}}\left(\theta^*\right) \leq & \sum_{t=0}^{T-1}\frac{\sqrt{2}M}{T}\|\nabla_\theta\ell_{\mathcal{T}}\left(\theta_t\right)-\nabla_\theta\ell_S\left(\theta_t\right)\| \\ & +\frac{\eta(C-1)^2}{2C^2N'^2}\sum_i\|\mathbf{1}^\top\mathbf{A}'^K_{(i)}\mathbf{X}'_{(i)}\|^2+\frac{M^2}{T\eta}\end{aligned} \tag{B.11}$$

Then we choose $\eta=\frac{M}{\sqrt{T}\sqrt{\sum_i\|\mathbf{1}^\top\mathbf{A}'^K_{(i)}\mathbf{X}'_{(i)}\|^2}}$ and we can get:

$$\begin{aligned}\min_{t=0,1,\ldots,T-1}\ell_{\mathcal{T}}\left(\theta_t\right)-\ell_{\mathcal{T}}\left(\theta^*\right) \leq & \sum_{t=0}^{T-1}\frac{\sqrt{2}M}{T}\|\nabla_\theta\ell_{\mathcal{T}}\left(\theta_t\right)-\nabla_\theta\ell_S\left(\theta_t\right)\| \\ & +\frac{3M}{2\sqrt{T}}\cdot\frac{C-1}{CN'}\sqrt{\sum_i\|\mathbf{1}^\top\mathbf{A}'^K_{(i)}\mathbf{X}'_{(i)}\|^2}\end{aligned} \tag{B.12}$$

II. For f_θ with mean pooling:

Following similar derivation as in the case of sum pooling, we have

$$\begin{aligned}\min_{t=0,1,\ldots,T-1}\ell_{\mathcal{T}}\left(\theta_t\right)-\ell_{\mathcal{T}}\left(\theta^*\right) \leq & \sum_{t=0}^{T-1}\frac{\sqrt{2}M}{T}\|\nabla_\theta\ell_{\mathcal{T}}\left(\theta_t\right)-\nabla_\theta\ell_S\left(\theta_t\right)\| \\ & +\frac{3M}{2\sqrt{T}}\cdot\frac{C-1}{CN'}\sqrt{\sum_i\frac{1}{n_i}\|\mathbf{1}^\top\mathbf{A}'^K_{(i)}\mathbf{X}'_{(i)}\|^2}\end{aligned} \tag{B.13}$$

where n_i is the number of nodes in i-th synthetic graph. □

B.1.2 Theorem for Node Classification Case

We adopt similar notations for representing the data in node classification but note that there is only one graph for node classification task. Let $\mathbf{A}\in\{0,1\}^{N\times N}$, $\mathbf{A'}\in\{0,1\}^{N'\times N'}$ denote the adjacency matrix for real graph and synthetic graph, respectively. Let $\mathbf{X}\in\mathbb{R}^{N\times d}$, $\mathbf{X'}\in\mathbb{R}^{N'\times d}$ denote the feature matrix for real graph and synthetic graph, respectively. We denote the cross entropy loss on the real samples as $\ell_{\mathcal{T}}(\theta)$ and denote that on synthetic samples as $\ell_S(\theta)$.

Theorem 2. *When we use a K-layer SGC as the model used in condensation, i.e.,* $f_\theta(\mathbf{A},\mathbf{X},\theta)=$

$\mathbf{A}^K\mathbf{X}\mathbf{W}$ *with* $\theta = \mathbf{W}$ *and assume that all network parameters satisfy* $\|\theta\|^2 \leq M^2 (M > 0)$*, we have*

$$\min_{t=0,1,\ldots,T-1} \ell_{\mathcal{T}}(\theta_t) - \ell_{\mathcal{T}}(\theta^*) \leq \sum_{t=0}^{T-1} \frac{\sqrt{2}M}{T} \|\nabla_\theta \ell_{\mathcal{T}}(\theta_t) - \nabla_\theta \ell_S(\theta_t)\| + \frac{3M}{2\sqrt{T}} \cdot \frac{C-1}{CN'} \|\mathbf{A}'^K \mathbf{X}'\| \tag{B.14}$$

Proof. We start by proving that $\ell_{\mathcal{T}}(\theta)$ is convex and $\ell_S(\theta)$ is lipschitz continuous when $f_\theta(\mathbf{A}, \mathbf{X}, \theta) = \mathbf{A}^K\mathbf{X}\mathbf{W}$.

(a) Convexity of $\ell_{\mathcal{T}}(\theta)$: Similar to the graph classification case, the Hessian matrix of $\ell_{\mathcal{T}}(\theta)$ in node classification is positive semidefinite and thus $\ell_{\mathcal{T}}(\theta)$ is convex.

(b) Lipschitz continuity of $\ell_S(\theta)$: As shown in [167], the lipschitz constant of softmax regression with cross entropy loss is $\frac{C-1}{Cm}\|\mathbf{X}\|$ with C being the number of classes and m being the number of samples. Thus, we know that the lipschitz constant of $\ell_S(\theta)$ is $\frac{C-1}{CN'}\|\mathbf{A}'^K\mathbf{X}'\|$, which indicates $\nabla_\theta \ell_S(\theta) \leq \frac{C-1}{CN'}\|\mathbf{A}'^K\mathbf{X}'\|$.

From the convexity of $\ell_{\mathcal{T}}(\theta)$, we still have the following inequality (see Eq. (B.10)). Then recall that $\ell_S(\theta)$ is lipschitz continuous and $\nabla_\theta \ell_S(\theta) \leq \frac{C-1}{CN'}\|\mathbf{A}'^K\mathbf{X}'\|$, and combine $\min_t (\ell_{\mathcal{T}}(\theta_t) - \ell_{\mathcal{T}}(\theta^*)) \leq \frac{\sum_{t=0}^{T-1} \ell_{\mathcal{T}}(\theta_t) - \ell_{\mathcal{T}}(\theta^*)}{T}$:

$$\min_{t=0,1,\ldots,T-1} \ell_{\mathcal{T}}(\theta_t) - \ell_{\mathcal{T}}(\theta^*) \leq \sum_{t=0}^{T-1} \frac{\sqrt{2}M}{T} \|\nabla_\theta \ell_{\mathcal{T}}(\theta_t) - \nabla_\theta \ell_S(\theta_t)\| + \frac{\eta(C-1)^2}{2C^2N'^2}\|\mathbf{A}'^K\mathbf{X}'\|^2 + \frac{M^2}{T\eta} \tag{B.15}$$

Then we choose $\eta = \frac{M}{\sqrt{T}\|\mathbf{A}'^K\mathbf{X}'\|}$ and we can get:

$$\min_{t=0,1,\ldots,T-1} \ell_{\mathcal{T}}(\theta_t) - \ell_{\mathcal{T}}(\theta^*) \leq \sum_{t=0}^{T-1} \frac{\sqrt{2}M}{T} \|\nabla_\theta \ell_{\mathcal{T}}(\theta_t) - \nabla_\theta \ell_S(\theta_t)\| + \frac{3M}{2\sqrt{T}} \cdot \frac{C-1}{CN'} \|\mathbf{A}'^K \mathbf{X}'\| \tag{B.16}$$

□

B.2 Experimental Setup

B.2.1 Algorithm

In the following, we show the algorithm of the proposed *DosCond* for condensing graphs.

Algorithm 3 *DosCond* for Condensing Graphs

Input: Training data $\mathcal{T} = (\mathbf{A}, \mathbf{X}, \mathcal{Y})$
Required: Pre-defined condensed labels $\mathcal{Y}'$, graph neural network f_θ, temperature τ, desired sparsity ϵ, regularization coefficient β, learning rates η_1, η_2, number of epochs K_1.
Initialize $\mathbf{\Omega}, \mathbf{X}'$
for $k = 0, \ldots, K_1 - 1$ **do**
  Sample $\theta_0 \sim P_{\theta_0}$, Sample $\alpha \sim \text{Uniform}(0, 1)$
  Compute $\mathbf{A}' = \sigma\left(\left(\log \alpha - \log(1-\alpha) + \mathbf{\Omega}\right)/\tau\right)$
  for $c = 0, \ldots, C - 1$ **do**
    Sample $(\mathbf{A}_c, \mathbf{X}_c, \mathcal{Y}_c) \sim \mathcal{T}$ and $(\mathbf{A}'_c, \mathbf{X}'_c, \mathcal{Y}'_c) \sim \mathcal{S}$
    Compute $\ell_T = \ell\left(f_{\theta_0}(\mathbf{A}_c, \mathbf{X}_c), \mathcal{Y}_c\right)$
    Compute $\ell_S = \ell\left(f_{\theta_0}(\mathbf{A}'_c, \mathbf{X}'_c), \mathcal{Y}'_c\right)$
    Compute $\ell_{\text{reg}} = \max(\sum_{i,j} \sigma(\mathbf{\Omega}_{ij}) - \epsilon, 0)$
    Update $\mathbf{\Omega} \leftarrow \mathbf{\Omega} - \eta_1 \nabla_{\mathbf{\Omega}}(D(\nabla_{\theta_0}\ell_T, \nabla_{\theta_0}\ell_S) + \beta\ell_{\text{reg}})$
    Update $\mathbf{X}' \leftarrow \mathbf{X}' - \eta_2 \nabla_{\mathbf{X}'}(D(\nabla_{\theta_0}\ell_T, \nabla_{\theta_0}\ell_S) + \beta\ell_{\text{reg}})$
  end for
end for
Return: $(\mathbf{\Omega}, \mathbf{X}', \mathcal{Y}')$

B.2.2 Dataset Statistics and Code

Dataset statistics are shown in Table 4 and 5.

Table B.1: Graph classification dataset statistics.

Dataset	Type	#Clases	#Graphs	Avg. Nodes	Avg. Edges
CIFAR10	Superpixel	10	60,000	117.6	941.07
ogbg-molhiv	Molecule	2	41,127	25.5	54.9
ogbg-molbace	Molecule	2	1,513	34.1	36.9
ogbg-molbbbp	Molecule	2	2,039	24.1	26.0
MUTAG	Molecule	2	188	17.93	19.79
NCI1	Molecule	2	4,110	29.87	32.30
DD	Molecule	2	1,178	284.32	715.66
E-commerce	Transaction	2	1,109	33.7	56.3

Table B.2: Node classification dataset statistics.

Dataset	#Nodes	#Edges	#Classes	#Features
Cora	2,708	5,429	7	1,433
Citeseer	3,327	4,732	6	3,703
Pubmed	19,717	44,338	3	500
Arxiv	169,343	1,166,243	40	128
Flickr	89,250	899,756	7	500

APPENDIX C

EMPOWERING GRAPH NEURAL NETWORKS WITH TEST-TIME GRAPH TRANSFORMATION

C.1 Proofs

C.1.1 Thereom 3

Theorem 3. *Let $\mathcal{L}_c$ denote the classification loss and $\mathcal{L}_s$ denote the surrogate loss, respectively. Let $\rho(G)$ denote the correlation between $\nabla_G \mathcal{L}_c(G, \mathcal{Y})$ and $\nabla_G \mathcal{L}_s(G)$, and let ϵ denote the learning rate for gradient descent. Assume that $\mathcal{L}_c$ is twice-differentiable and its Hessian matrix satisfies $\|\mathbf{H}(G, \mathcal{Y})\|_2 \leq M$ for all G. When $\rho(G) > 0$ and $\epsilon < \frac{2\rho(G)\|\nabla_G \mathcal{L}_c(G,\mathcal{Y})\|_2}{M\|\nabla_G \mathcal{L}_s(G)\|_2}$, we have*

$$\mathcal{L}_c\left(G - \epsilon \nabla_G \mathcal{L}_s\left(G\right), \mathcal{Y}\right) < \mathcal{L}_c\left(G, \mathcal{Y}\right). \tag{C.1}$$

Proof. Given that $\mathcal{L}_c$ is differentiable and twice-differentiable, we perform first-order Taylor expansion with Lagrange form of remainder at $G - \epsilon \nabla_G \mathcal{L}_s(G)$:

$$\begin{aligned} &\mathcal{L}_c\left(G - \epsilon \nabla_G \mathcal{L}_s\left(G\right), \mathcal{Y}\right) \\ &= \mathcal{L}_c\left(G, \mathcal{Y}\right) - \epsilon \rho\left(G\right) \|\nabla_G \mathcal{L}_c\left(G, \mathcal{Y}\right)\|_2 \|\nabla_G \mathcal{L}_s\left(G\right)\|_2 + \frac{\epsilon^2}{2} \nabla_G \mathcal{L}_s(G)^\top \mathbf{H}(G - \epsilon\theta \nabla_G \mathcal{L}_s(G), \mathcal{Y}) \nabla_G \mathcal{L}_s(G), \end{aligned} \tag{C.2}$$

where $\theta \in (0, 1)$ is a constant given by Lagrange form of the Taylor's remainder (here we slightly abuse the notation), and $\rho\left(G\right)$ is the correlation between $\nabla_G \mathcal{L}_c(G, \mathcal{Y})$ and $\nabla_G \mathcal{L}_s(G)$:

$$\rho\left(G\right) = \frac{\nabla_G \mathcal{L}_c\left(G, \mathcal{Y}\right)^T \nabla_G \mathcal{L}_s\left(G\right)}{\|\nabla_G \mathcal{L}_c\left(G, \mathcal{Y}\right)\|_2 \|\nabla_G \mathcal{L}_s\left(G\right)\|_2}. \tag{C.3}$$

Before we proceed to the next steps, we first show that given a vector $\mathbf{p}$ and a symmetric matrix $\mathbf{A}$, the inequality $\mathbf{p}\top\mathbf{A}\mathbf{p} \leq \|\mathbf{p}\|_2^2 \|\mathbf{A}\|_2$ holds:

$$\begin{aligned} \mathbf{p}^\top \mathbf{A} \mathbf{p} &= \sum \sigma_i \mathbf{p}^\top \mathbf{u}_i \mathbf{u}_i^\top \mathbf{p} \quad \text{(Performing SVD on } \mathbf{A}\text{, i.e., } \mathbf{A} = \sum \sigma_i \mathbf{u}_i \mathbf{u}_i^\top\text{)} \\ &= \sum \sigma_i \mathbf{v}_i^\top \mathbf{v}_i \quad \text{(Let } \mathbf{v} = \mathbf{U}^\top p\text{, where } \mathbf{U} = [\mathbf{u}_1; \mathbf{u}_2; \ldots; \mathbf{u}_n]\text{)} \\ &\leq \sum \sigma_{\max} \mathbf{v}_i^\top \mathbf{v}_i = \sigma_{\max} \|\mathbf{v}\|_2^2 = \sigma_{\max} \|\mathbf{U}^\top p\|_2^2 \\ &= \sigma_{\max} \|\mathbf{p}\|_2^2 \quad (\mathbf{U} \text{ is an orthogonal matrix}) \\ &= \|\mathbf{p}\|_2^2 \|\mathbf{A}\|_2 \end{aligned} \tag{C.4}$$

Since the Hessian matrix is symmetric, we can use the above inequality to derive:

$$\begin{aligned}&\mathcal{L}_c\left(G-\epsilon\nabla_G\mathcal{L}_s\left(G\right),\mathcal{Y}\right)\\&\leq\mathcal{L}_c\left(G,\mathcal{Y}\right)-\epsilon\rho\left(G\right)\|\nabla_G\mathcal{L}_c\left(G,\mathcal{Y}\right)\|_2\|\nabla_G\mathcal{L}_s\left(G\right)\|_2+\frac{\epsilon^2}{2}\|\nabla_G\mathcal{L}_s(G)\|_2^2\|\mathbf{H}(G-\epsilon\theta\nabla_G\mathcal{L}_s(G),\mathcal{Y})\|_2.\end{aligned}\tag{C.5}$$

Then we rewrite Eq. (C.5) as:

$$\begin{aligned}&\mathcal{L}_c\left(G-\epsilon\nabla_G\mathcal{L}_s\left(G\right),\mathcal{Y}\right)-\mathcal{L}_c\left(G,\mathcal{Y}\right)\\&\leq\ -\epsilon\rho\left(G\right)\|\nabla_G\mathcal{L}_c\left(G,\mathcal{Y}\right)\|_2\|\nabla_G\mathcal{L}_s\left(G\right)\|_2+\frac{\epsilon^2}{2}\|\nabla_G\mathcal{L}_s(G)\|_2^2\|\mathbf{H}(G-\epsilon\theta\nabla_G\mathcal{L}_s(G),\mathcal{Y})\|_2.\end{aligned}\tag{C.6}$$

Given $\|\mathbf{H}(G,\mathcal{Y})\|_2\leq M$, we know

$$\begin{aligned}&\mathcal{L}_c\left(G-\epsilon\nabla_G\mathcal{L}_s\left(G\right),\mathcal{Y}\right)-\mathcal{L}_c\left(G,\mathcal{Y}\right)\\&\leq\ -\epsilon\rho\left(G\right)\|\nabla_G\mathcal{L}_c\left(G,\mathcal{Y}\right)\|_2\|\nabla_G\mathcal{L}_s\left(G\right)\|_2+\frac{\epsilon^2M}{2}\|\nabla_G\mathcal{L}_s(G)\|_2^2.\end{aligned}\tag{C.7}$$

By setting $\epsilon=\frac{2\rho(G)\|\nabla_G\mathcal{L}_c(G,\mathcal{Y})\|_2}{M\|\nabla_G\mathcal{L}_s(G)\|_2}$, we have

$$\mathcal{L}_c\left(G-\epsilon\nabla_G\mathcal{L}_s\left(G\right),\mathcal{Y}\right)-\mathcal{L}_c\left(G,\mathcal{Y}\right)=0.\tag{C.8}$$

Therefore, when $\epsilon<\frac{2\rho(G)\|\nabla_G\mathcal{L}_c(G,\mathcal{Y})\|_2}{M\|\nabla_G\mathcal{L}_s(G)\|_2}$ and $\rho(G)>0$, we have

$$\mathcal{L}_c\left(G-\epsilon\nabla_G\mathcal{L}_s\left(G\right),\mathcal{Y}\right)<\mathcal{L}_c\left(G,\mathcal{Y}\right).\tag{C.9}$$

□

C.1.2 Thereom 4

Theorem 4. *Assume that the augmentation function $\mathcal{A}(\cdot)$ generates a data view of the same class for the test nodes and the node classes are balanced. Assume for each class, the mean of the representations obtained from Z and $\hat{Z}$ are the same. Minimizing the first term in Eq. (5.6) is approximately minimizing the class-conditional entropy $H(Z|Y)$ between features Z and labels Y.*

Proof. For convenience, we slightly abuse the notations to replace $\frac{\mathbf{z}_i}{\|\mathbf{z}_i\|}$ and $\frac{\hat{\mathbf{z}}_i}{\|\hat{\mathbf{z}}_i\|}$ with $\mathbf{z}_i$ and $\hat{\mathbf{z}}_i$, respectively. Then we have $\|\mathbf{z}_i\|=\|\hat{\mathbf{z}}_i\|=1$. Let Z_k denote the set of test samples from class k; thus

$|Z_k| = \frac{N}{K}$. Let $\stackrel{c}{=}$ denote equality up to a multiplicative and/or additive constant. Then the first term in Eq. (5.6) can be rewritten as:

$$\sum_{i=1}^{N}(1 - \hat{\mathbf{z}}_i^\top \mathbf{z}_i) = \sum_{i=1}^{N}\left(1 - \hat{\mathbf{z}}_i^\top \mathbf{z}_i\right) \stackrel{c}{=} \sum_{k=1}^{K}\frac{1}{|Z_k|}\sum_{\mathbf{z}_i \in Z_k}\left(-\hat{\mathbf{z}}_i^\top \mathbf{z}_i\right) \tag{C.10}$$

Let $\mathbf{c}_k$ be the mean of hidden representations from class k; then we have $\mathbf{c}_k = \frac{1}{|Z_k|}\sum_{\mathbf{z}_i \in Z_k}\mathbf{z}_i = \frac{1}{|Z_k|}\sum_{\hat{\mathbf{z}}_i \in \hat{Z}_k}\hat{\mathbf{z}}_i$. Now we build the connection between Eq. (C.10) and $\sum_{i=1}^{K}\sum_{\mathbf{z}_i \in Z_k}\|\mathbf{z}_i - \mathbf{c}_k\|^2$:

$$\sum_{i=1}^{K}\sum_{\mathbf{z}_i \in Z_k}\|\mathbf{z}_i - \mathbf{c}_k\|^2 \tag{C.11}$$

$$= \sum_{i=1}^{K}\left(\sum_{\mathbf{z}_i \in Z_k}\|\mathbf{z}_i\|^2 - 2\sum_{\mathbf{z}_i \in Z_k}\mathbf{z}_i^\top \mathbf{c}_k + |Z_k|\|\mathbf{c}_k\|^2\right)$$

$$= \sum_{i=1}^{K}\left(\sum_{\mathbf{z}_i \in Z_k}\|\mathbf{z}_i\|^2 - 2\frac{1}{|Z_k|}\sum_{\mathbf{z}_i \in Z_k}\sum_{\hat{\mathbf{z}}_i \in \hat{Z}_k}\hat{\mathbf{z}}_i^\top \mathbf{z}_i + \frac{1}{|Z_k|}\sum_{\mathbf{z}_i \in Z_k}\sum_{\hat{\mathbf{z}}_i \in \hat{Z}_k}\hat{\mathbf{z}}_i^\top \mathbf{z}_i\right)$$

$$= \sum_{i=1}^{K}\left(\sum_{\mathbf{z}_i \in Z_k}\|\mathbf{z}_i\|^2 - \frac{1}{|Z_k|}\sum_{\mathbf{z}_i \in Z_k}\sum_{\hat{\mathbf{z}}_i \in Z_k}\hat{\mathbf{z}}_i^\top \mathbf{z}_i\right)$$

$$\stackrel{c}{=} \sum_{i=1}^{K}\left(\frac{1}{|Z_k|}\sum_{\mathbf{z}_i \in Z_k}\sum_{\hat{\mathbf{z}}_i \in \hat{Z}_k}\|\mathbf{z}_i\|^2 - \frac{1}{|Z_k|}\sum_{\mathbf{z}_i \in Z_k}\sum_{\hat{\mathbf{z}}_i \in Z_k}\hat{\mathbf{z}}_i^\top \mathbf{z}_i\right)$$

$$= \sum_{i=1}^{K}\left(\frac{1}{|Z_k|}\sum_{\mathbf{z}_i \in Z_k}\sum_{\hat{\mathbf{z}}_i \in \hat{Z}_k}\left(\|\mathbf{z}_i\|^2 - \hat{\mathbf{z}}_i^\top \mathbf{z}_i\right)\right)$$

$$\stackrel{c}{=} \sum_{i=1}^{K}\left(\frac{1}{|Z_k|}\sum_{\mathbf{z}_i \in Z_k}\sum_{\hat{\mathbf{z}}_i \in \hat{Z}_k}\left(-\hat{\mathbf{z}}_i^\top \mathbf{z}_i\right)\right) \tag{C.12}$$

By comparing Eq. (C.10) and Eq. (C.12), the only difference is that Eq. (C.12) includes more positive pairs for loss calculation. Hence, minimizing Eq. (C.10) can be viewed as approximately minimizing Eq. (C.12) or Eq. (C.11) through sampling positive pairs. As demonstrated in the work [9], Eq. (C.11) can be interpreted as a conditional cross-entropy between Z and another random variable $\bar{Z}$, whose conditional distribution given Y is a standard Gaussian centered around $\mathbf{c}_Y : Z \mid Y \sim \mathcal{N}(\mathbf{c}_Y, \mathbf{I})$:

$$\sum_{\mathbf{z}_i \in Z_k}\|\mathbf{z}_i - \mathbf{c}_k\|^2 = \mathcal{H}(Z \mid Y) + \mathcal{D}_{KL}(Z||\bar{Z} \mid Y) \geq \mathcal{H}(Z \mid Y) \tag{C.13}$$

Hence, minimizing the first term in Eq. (5.6) is approximately minimizing $H(Z|Y)$. □

Discussion: We note that the assumption "the mean of the representations obtained from Z and $\hat{Z}$ are the same" can be inferred by the first assumption about data augmentation. Let $p_k(X)$ denote the distribution of samples with class k and let $x \sim p_k(X)$ denote the sample with class k. Recall that we assume the data augmentation function $\mathcal{A}(\cdot)$ is strong enough to generate a data view that can simulate the test data from the same class. In this regard, the new data view can be regarded as an independent sample from the same class, i.e., $\mathcal{A}(x) \sim p_k(X)$. Hence, the expectation of Z and $\hat{Z}$ is the same and we would approximately have that "the mean of Z and $\hat{Z}$ is the same for each class". Particularly, when the number of samples is relatively large, the mean of Z ($\hat{Z}$) would be close to the true distribution mean. For example, on one graph of Cora, the mean absolute difference between the two mean representations of Z and $\hat{Z}$ are [0.018, 0.009, 0.021, 0.024, 0.016, 0.014, 0.0, 0.016, 0.023] for each class, which are actually very small.

C.1.3 A Figurative Example

In Figure C.1, we show an example of adversarial attack which causes the aggregated features for two nodes to be the same. Given two nodes $\mathbf{x}_1$ and $\mathbf{x}_2$ and their connections, we are interested in predicting their labels. Assume a mean aggregator is used for aggregating features from the neighbors. Before attack, the aggregated features for them are $\bar{\mathbf{x}}_1 = [0.45]$ and $\bar{\mathbf{x}}_2 = [0.53]$ while after attack the aggregated features become the same $\bar{\mathbf{x}}_1 = \bar{\mathbf{x}}_2 = [0.45]$. In this context, it is impossible to learn a classifier that can distinguish the two nodes.

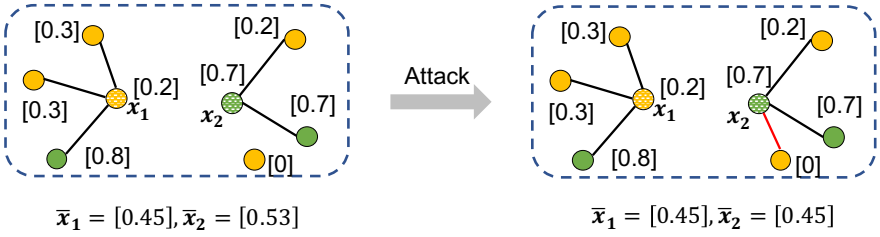


Figure C.1: Given two nodes $\mathbf{x}_1$ and $\mathbf{x}_2$ and their connections, we are interested in predicting their labels. The color indicates the node label and "[0.3]" suggests that the associated node feature is 0.3. Assume a mean aggregator is used for aggregating features from the neighbors. **Left:** we show the clean graph without adversarial attack. The aggregated features for the two center nodes are $\bar{\mathbf{x}}_1 = [0.45]$ and $\bar{\mathbf{x}}_2 = [0.53]$. **Right:** we show the attacked graph where the red edge indicates the adversarial edge injected by the attacker. The aggregated features for the two center nodes become $\bar{\mathbf{x}}_1 = [0.45]$ and $\bar{\mathbf{x}}_2 = [0.45]$.

C.2 Algorithm

We show the detailed algorithm of GCond in Algorithm 4. In detail, we first initialize $\Delta_{\mathbf{A}}$ and $\Delta_{\mathbf{X}'}$ as zero matrices and calculate $\mathcal{L}_s$ based on Eq. (5.6). Since we alternatively optimize $\Delta_{\mathbf{A}}$ and $\Delta_{\mathbf{X}'}$, we update $\Delta_{\mathbf{X}'}$ every τ_1 epochs and update $\Delta_{\mathbf{A}}$ every τ_2 epochs. When the optimization is done, we sample the discrete graph structure for K times and select the one that results in the smallest $\mathcal{L}_s$ as the final adjacency matrix.

Algorithm 4 GCond for Test-Time Graph Transformation

Input: Pre-trained model f_θ and test graph dataset $G_{\text{Te}} = (\mathbf{A}, \mathbf{X}')$.
Output: Model prediction $\hat{\mathbf{Y}}$ and transformed graph $G' = (\mathbf{A}', \mathbf{X}'')$.
Initialize $\Delta_{\mathbf{A}}$ and $\Delta_{\mathbf{X}}$ as zero matrices
for $t = 0, \ldots, T-1$ **do**
  Compute $\mathbf{A}' = \mathbf{A} \oplus \Delta_{\mathbf{A}}$ and $\mathbf{X}' = \mathbf{X} + \Delta_{\mathbf{X}}$
  Compute $\mathcal{L}_s(\Delta_{\mathbf{A}}, \Delta_{\mathbf{X}})$ as shown in Eq. (5.6)
  if $t\%(\tau_1 + \tau_2) < \tau_1$ **then**
    Update $\Delta_{\mathbf{X}} \leftarrow \Delta_{\mathbf{X}} - \eta_1 \nabla_{\Delta_{\mathbf{X}}} \mathcal{L}_s$
  else
    Update $\Delta_{\mathbf{A}} \leftarrow \Pi_{\mathcal{P}} \left(\Delta_{\mathbf{A}} - \eta \nabla_{\Delta_{\mathbf{A}}} \mathcal{L}_s\right)$
$\ell_{\text{best}} = \infty$ # store the best loss
for $k = 0, \ldots, K-1$ **do**
  Sample $\mathbf{A}'_0 \sim \text{Bernoulli}(\mathbf{A} \oplus \Delta_{\mathbf{A}})$
  Calculate $\mathcal{L}_s$ with $\mathbf{A}'_0$ as input
  if $\mathcal{L}_s < \ell_{best}$ **then**
    $\ell_{\text{best}} = \mathcal{L}_s$
    $\mathbf{A}' = \mathbf{A}'_0$
$\mathbf{X}' = \mathbf{X} + \Delta_{\mathbf{X}}$
$\hat{\mathbf{Y}} = f_\theta(\mathbf{A}', \mathbf{X}')$
Return: $\hat{\mathbf{Y}}$, $(\mathbf{A}', \mathbf{X}')$

C.3 Datasets and Hyper-Parameters

In this section, we reveal the details of reproducing the results in the experiments. We will release the source code upon acceptance.

C.3.1 Out-of-Distribution (OOD) Setting

The out-of-distribution (OOD) problem indicates that the model does not generalize well to the test data due to the distribution gap between training data and test data [162], which is also referred

Table C.1: Summary of the experimental datasets that entail diverse distribution shifts.

Distribution Shift	Dataset	#Nodes	#Edges	#Classes	Train/Val/Test Split	Metric	Adapted From
Artificial Transformation	Cora	2,703	5,278	10	Domain-Level	Accuracy	[166]
	Amz-Photo	7,650	119,081	10	Domain-Level	Accuracy	[127]
Cross-Domain Transfers	Twitch-E	1,912 9,498	31,299 - 153,138	2	Domain-Level	ROC-AUC	[120]
	FB100	769 41,536	16,656 - 1,590,655	2	Domain-Level	Accuracy	[136]
Temporal Evolution	Elliptic	203,769	234,355	2	Time-Aware	F1 Score	[114]
	OGB-Arxiv	169,343	1,166,243	40	Time-Aware	Accuracy	[60]

to as distribution shifts. Numerous research studies have been conducted to explore this problem and propose potential solutions [46, 205, 163, 164, 154, 92, 17, 11, 52, 155, 174]. In the following, we introduce the datasets used for evaluating the methods that tackle the OOD issue in the graph domain.

Dataset Statistics. For the evaluation on OOD data, we use the datasets provided by [154]. The dataset statistics are shown in Table C.1, which includes three distinct type of distribution shifts: (1) artificial transformation which indicates the node features are replaced by synthetic spurious features; (2) cross-domain transfers which means that graphs in the dataset are from different domains and (3) temporal evolution where the dataset is a dynamic one with evolving nature. Notably, we use the datasets provided by [154], which were adopted from the aforementioned references with manually created distribution shifts. Note that there can be multiple training/validaiton/test graphs. Specifically, Cora and Amazon-Photo have 1/1/8 graphs for training/validation/test sets. Similarly, the splits are 1/1/5 on Twitch-E, 3/2/3 on FB-100, 5/5/33 on Elliptic, and 1/1/3 on Ogbn-arxiv.

Hyper-Parameter Setting. For the setup of backbone GNNs, we majorly followed [154]:

(a) GCN: the architecture setup is 5 layers with 32 hidden units for Elliptic and Ogbn-arxiv, and 2 layers with 32 hidden units for other datasets, and with batch normalization for all datasets. The learning rate is set to 0.001 for Cora and Amz-Photo, 0.01 for other datasets; the weight decay is set to 0 for Elliptic and Ogbn-arxiv, and 0.001 for other datasets.

(b) GraphSAGE: the architecture setup is 5 layers with 32 hidden units for Elliptic and Ogbn-arxiv, and 2 layers with 32 hidden units for other datasets, and with batch normalization for all datasets. The learning rate is set to 0.01 for all datasets; the weight decay is set to 0 for Elliptic and Ogbn-arxiv, and 0.001 for other datasets.

(c) GAT: the architecture setup is 5 layers for Elliptic and Ogbn-arxiv, and 2 layers for other datasets, and with batch normalization for all datasets. Each layer contains 4 attention heads and each head is associated with 32 hidden units. The learning rate is set to 0.01 for all datasets; the weight decay is set to 0 for Elliptic and Ogbn-arxiv, and 0.001 for other datasets.

(d) GPR: We use 10 propagation layers and 2 transformation layers with 32 hidden units. The learning rate is set to 0.01 for all datasets; the weight decay is set to 0 for Elliptic and Ogbn-arxiv, and 0.001 for other datasets. Note that GPR does not contain batch normalization layers.

For the baseline methods, we tuned their hyper-parameters based on the validation performance. For DropEdge, we search the drop ratio in the range of [0, 0.05, 0.1, 0.15, 0.2, 0.3, 0.5, 0.7]. For Tent, we search the learning rate in the range of [1e-2, 1e-3, 1e-4, 1e-5, 1e-6] and the running epochs in [1, 10, 20, 30]. For EERM, we followed the instruction provided by the original paper. For $GCond$, we alternatively optimize node features for $\tau_1 = 4$ epochs and optimize graph structure $\tau_2 = 1$ epoch. We adopt DropEdge as the augmentation function $\mathcal{A}(\cdot)$ and set the drop ratio to 0.5. We use Adam optimizer for both feature learning and structure learning. We further search the learning rate of feature adaptation η_1 in [5e-3, 1e-3, 1e-4, 1e-5, 1e-6], learning rate of structure adaptation η_2 in [0.5, 0.1, 0.01], the modification budget B in [0.5%, 1%, 5%] of the original edges, total epochs T in [5, 10]. We note that the process of tuning hyper-parameters is quick due to the high efficiency of test-time adaptation as we demonstrated in Section 5.4.1.

Evaluation Protocol. For ERM (standard training), we train all the GNN backbones using the common cross entropy loss. For DropEdge, we drop a certain amount of edges at each training epoch. For EERM, it optimizes a bi-level problem to obtain a trained classifier. Note that the aforementioned three methods do not perform any test-time adaptation and their model parameters are fixed during test. For the two test-time adaptation methods, Tent and GCond, we first obtain the GNN backbones pre-trained from ERM and adapt the model parameters or graph data at test time, respectively. Furthermore, Tent minimizes the entropy loss while GCond minimizes the contrastive surrogate loss.

Quantifying Distribution Shifts. Following SR-GNN [205], we adopt central moment

discrepancy (CMD) [176] as the measurement to quantify the distribution shifts in different graphs. Specifically, given a pre-trained model, we obtain its hidden representation on the training graph and test graphs, denoted as Z_{tr} and Z_{te}. Then we calculate their distance by the CMD metric, i.e., $\text{CMD}(Z_{\text{tr}}), Z_{\text{te}}$. We show the results in Table C.2 and we can observe certain distribution shifts as these values are not small. Let's take the Ogbn-arxiv dataset as an example, where we select papers published before 2011 for training, 2011-2014 for validation, and within 2014-2016/2016-2018/2018-2020 for test. In this context, the distribution shift is from the temporal change. In Table C.3, we show the CMD values, ERM performance and GCond performance. From the table, we can find that (1) the CMD value on the validation graph is essentially smaller than those on test graphs; and (2) GCN performances on test graphs (with larger shifts) are lower than that on the validation graph.

Table C.2: CMD values on each individual graph based on the pre-trained GCN.

GraphID	G_0	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8
Amz-Photo	6.4	5.1	5.5	3.7	2.8	3.7	3.9	6.6	-
Cora	5.4	4.2	4.8	6.3	5.5	4.8	4.6	5.4	-
Elliptic	80.2	90.8	114.3	86.5	789.3	781.6	99.4	100.4	150.6
Ogbn-arxiv	14.7	20.6	10.4	-	-	-	-	-	-
FB-100	29.7	16.9	32.9	-	-	-	-	-	-
Twitch-E	8.6	6.1	9.0	8.4	9.7	-	-	-	-

Table C.3: CMD values and the performances of ERM and GCond on Ogbn-arxiv

Method	2011-2014 (Val)	2014-2016	2014-2016	2016-2018	2018-2020
CMD	2.5	14.7	14.7	20.6	10.4
ERM	45.32±0.50	41.29±1.13	41.29±1.13	38.69±1.33	35.78±1.81
GCond	45.82±0.38	44.03±0.95	44.03±0.95	41.90±1.28	38.81±1.47

C.3.2 Abnormal Features

Dataset Statistics. In these two settings, we choose the original version of popular benchmark datasets Cora, Citeseer, Pubmed and Ogbn-arxiv. The statistics for these datasets are shown in Table C.4. Note that we only have one test graph, and the injection of abnormal features or adversarial attack happens after the training process of backbone model, which can be viewed as evasion attack.

Table C.4: Dataset statistics for experiments on abnormal features and adversarial attack.

Dataset	Classes	Nodes	Edges	Features	Training Nodes	Validation Nodes	Test Nodes
Cora	7	2708	5278	1433	20 per class	500	1000
Citeseer	6	3327	4552	3703	20 per class	500	1000
Pubmed	3	19717	44324	500	20 per class	500	1000
Ogbn-arxiv	40	169343	1166243	128	54%	18%	28%

Hyper-Parameter Settings. We closely followed AirGNN [96] to set up the hyper-parameters for the baselines:

(a) GCN: the architecture setup is 2 layers with 64 hidden units without batch normalization for Cora, Citeseer and Pubmed, and 3 layers with 256 hidden units with batch normalization for Ogbn-arxiv. The learning rate is set to 0.01.

(b) GAT: the architecture setup is 2 layers with 8 hidden units in each of the 8 heads without batch normalization for Cora, Citeseer and Pubmed, and 3 layers with 32 hidden units in each of the 8 heads with batch normalization for Ogbn-arxiv. The learning rate is set to 0.005.

(c) APPNP: the architecture setup is 2-layer transformation with 64 hidden units and 10-layer propagation without batch normalization for Cora, Citeseer and Pubmed; the architecture is set to 3-layer transformation with 256 hidden units and 10-layer propagation with batch normalization for Ogbn-arxiv. The learning rate is set to 0.01.

(d) AirGNN: The architecture setup is the same as APPNP and the hyper-parameter λ is set to 0.3 for Ogbn-arxiv and 0.5 for other datasets.

(e) AirGNN-t: The architecture setup is the same as AirGNN but we tune the hyper-parameter λ in AirGNN based on performance on the combination of training and validation nodes at test stage. This is because the test graph has the same graph structure as the training graph; thus we can take advantage of the label information of training nodes (as well as validation nodes) to tune the hyper-parameters. Specifically, we search λ in the range of [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9] for each noise ratio.

For the setup of $GCond$, we alternatively optimize node features for $\tau_1 = 4$ epochs and optimize graph structure $\tau_2 = 1$ epoch. We adopt DropEdge as the augmentation function $\mathcal{A}(\cdot)$ and set the

drop ratio to 0.5. We use Adam optimizer for both feature learning and structure learning. We further search the learning rate of feature adaptation η_1 in [1, 1e-1, 1e-2], total epochs T in [10, 20]. The modification budget B to 5% of the original edges and the learning rate of structure adaptation η_2 is set to 0.1. It is worth noting that *we use a weighted combination of contrastive loss and training classification loss*, i.e., $\mathcal{L}_{\text{train}} + \lambda\mathcal{L}_s$, instead of optimizing the contrastive loss alone. We adaopt this strategy because that the training graph and the test graph were the same graph before the injection of abnormal features. Here the λ is tuned in the range of [1e-2, 1e-3, 1e-4]. We study the effects of contrastive loss and training classification loss in Appendix C.4.7.

C.3.3 Adversarial Attack

Dataset Statistics. We used Ogbn-arxiv for the adversarial attack experiments and the dataset statistics can be found in Table C.4. Again, we only have one test graph for this dataset.

Hyper-Parameter Settings. The setup of GCN and GAT is the same as that in the setting of abnormal features. For the defense methods including SimPGCN, RobustGCN and GCNJaccard, we use the DeepRobust [85] library to implement them. For GCNJaccard, we tune its threshold hyper-parameter in the range of [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]. The hyper-parameter is also tuned based on the performance of training and validation nodes (same as Appendix C.3.2). Note that the popular defenses ProGNN [70] and GCNSVD [37] were not included because they throw OOM error due to the expensive eigen-decomposition operation.

We use the official implementation of the scalable attack PR-BCD [48] to attack the test graph. We note that when performing adversarial attacks, the setting is more like transductive setting where the training graph and test graph are the same. However, the test graph becomes different from the training graph after the attack. Since the training graph and test graph were originally the same graph, *we use a weighted combination of contrastive loss and training classification loss*, i.e., $\mathcal{L}_{\text{train}} + \lambda\mathcal{L}_s$, instead of optimizing the contrastive loss alone. For the setup of $GCond$, we alternatively optimize node features for $\tau_1 = 1$ epoch and optimize graph structure $\tau_2 = 4$ epochs. We fix the learning rate of feature adaptation η_1 to 1e-3, learning rate of structure adaptation η_2 to 0.1, λ to 1, total epochs T to 50 and modification budget B to 30% of the original edges.

C.3.4 Hardware and Software Configurations.

We perform experiments on NVIDIA Tesla V100 GPUs. The GPU memory and running time reported in Table 5.2 are measured on one single V100 GPU. Additionally, we use eight CPUs, with the model name as Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz. The operating system we use is CentOS Linux 7 (Core).

C.4 More Experimental Results

C.4.1 Comparison to Graph Domain Adaptation

Our work is related to graph domain adaptation (GraphDA) [128, 153, 205], but they are also **highly different**. In Table C.5, we summarize the differences between GraphDA and GCond. In detail, there are the following differences:

(a) **Data and losses**: GraphDA methods optimize the loss function based on both labeled source data (training data) and unlabeled target data (test data), while GCond only requires target data during inference. Hence, GraphDA methods are infeasible when access to the source data is prohibited such as online service.

(b) **Parameter**: To our best knowledge, existing GraphDA methods are model-centric approaches while GCond is a data-centric approach. GCond adapts the data instead of the model, which can be more useful in some settings as we showed in the Example of Section 5.3.3.

(c) **Efficiency**: GraphDA is indeed a training-time adaptation and for each given test graph, it would require training the model on the source and target data. Thus, it is much less efficient than GCond, especially when we have multiple test graphs (e.g., 33 test graphs for Elliptic).

Table C.5: Comparison between GraphDA and GCond. They differ by their data and losses.

Setting	Source	Target	Train Loss	Test Loss	Parameter	Efficiency
GraphDA	G_{tr}	G_{te}	$\mathcal{L}(G_{\text{tr}}, \mathcal{Y}_{\text{tr}}) + \mathcal{L}(G_{\text{tr}}, G_{\text{te}})$	-	f_θ	Low
GCond	-	G_{te}	-	$\mathcal{L}(G_{\text{te}})$	G_{te}	High

To compare their empirical performance, we include two GraphDA methods (SR-GNN [205] and UDA-GCN [153]) and one general domain adaptation method (DANN [46]). SR-GNN regularizes

the model's performance on the source and target domains. Note that SR-GNN is originally developed under the transductive setting where the training graph and test graph are the same. To apply SR-GNN in our OOD setting, we assume the test graph is available during the training stage of SR-GNN, as typically done in domain adaptation methods. UDA-GCN is another work that tackles graph data domain adaptation, which exploits local and global information for different domains. In addition, we also include DANN, which adopts an adversarial domain classifier to promote the similarity of feature distributions between different domains. We followed the authors' suggestions in their paper to tune the hyper-parameters and the results are shown in Table C.6. On the one hand, we can observe that GraphDA methods generally improve the performance of GCN under distribution shift and SRGNN is the best performing baseline. On the other hand, GCond performs the best on all datasets except Amz-Photo. On Amz-Photo, GCond does not improve as much as SR-GNN, which indicates that joint optimization over source and target is necessary for this dataset. However, recall that domain adaptation methods are less efficient due to the joint optimization on source and target: the adaptation time of SR-GNN on the 8 graphs of Amz-Photo is 83.5s while that of GCond is 4.9s (plus pre-training time 10.1s). Overall, test-time graph transformation exhibits strong advantages of effectiveness and efficiency.

Table C.6: Performance comparison between GCond and graph domain adaptation methods.

Method	Amz-Photo	Cora	Elliptic	FB-100	Ogbn-arxiv	Twitch-E
ERM	93.79±0.97	91.59±1.44	50.90±1.51	54.04±0.94	38.59±1.35	59.89±0.50
UDA-GCN	91.70±0.35	92.65±0.46	51.57±1.31	54.11±0.54	39.43±0.71	52.12±0.38
DANN	94.08±0.21	92.89±0.64	53.00±0.97	51.53±1.47	36.60±1.26	60.13±0.53
SRGNN	**94.64±0.17**	94.08±0.28	51.94±0.81	54.08±1.10	38.92±0.65	59.21±0.51
GCond	94.13±0.77	**94.66±0.63**	**55.88±3.10**	**54.32±0.60**	**41.59±1.20**	**60.42±0.86**

C.4.2 Comparison to Graph Structure Learning

Our work is also relevant to graph structure learning (GSL) [45, 70, 18, 194, 121, 56, 41] which learns the graph structure during the training time while not adapting the graph structure at test stage. Our proposed test-time graph transform is essentially different from these works as we do not modify the training data but only the test data. It can be of interest to adopt GSL method at

test time by also adapting the test graph structure. However, most existing GSL methods optimize the cross entropy loss defined on the labels to update graph structure, thus not applicable in the absence of test labels. One exception is SLAPS [41] which utilizes a self-supervised loss together with the cross entropy loss to optimize the graph structure. However, the default setting in SLAPS is generating structure for raw data points (with no given graph structure). Hence, using SLAPS for our settings requires considerable changes. Furthermore, we highlight two more weaknesses of SLAPS compared to GCond.

(a) **Introducing additional parameters**. SLAPS uses a denoising loss as self-supervision. In detail, it first injects noise into node features and trains a denoising autoencoder to denoise the noisy features. This introduces additional parameters from the denoising autoencoder and inevitably changes the model architecture.

(b) **Not learning features**. As other GSL methods, SLAPS does not learn node features. We argue that feature learning is highly important under the abnormal feature setting as shown in Table C.12. For example, structure learning only improves GCN by 2% on Ogbn-arxiv while feature learning can improve GCN by 20%. Thus, without the feature learning component, the performance will significantly drop when encountering noisy features.

Since GCond is highly versatile and we can use any self-supervised loss as the surrogate loss, we can simply replace the contrastive loss in Eq. (5.6) with the denoising loss of SLAPS instead of paying considerable efforts in adjusting SLAPS. We refer to the loss used for denoising as SLAPS loss and adopt it for TTGT. Note that we first train the parameters of the DAE used for denoising while keeping the pre-trained model fixed. Then we fix both DAE and the pre-trained model and optimize the test graph for TTGT. The results are shown in Table C.7. From the table, we can observe that the SLAPS loss (or feature denoising loss) does not work as well as the contrastive loss.

Table C.7: Comparison between SLAPS loss and our contrastive loss.

	Amz-Photo	Cora	Elliptic	FB-100	Ogbn-arxiv	Twitch-E
None	93.79±0.97	91.59±1.44	50.90±1.51	54.04±0.94	38.59±1.35	59.89±0.50
SLAPS	93.97±1.04	91.41±1.23	50.54±1.81	54.08±0.76	41.38±1.35	59.85±0.68
$\mathcal{L}_s$ in Eq. (5.6)	**94.13±0.77**	**94.66±0.63**	**55.88±3.10**	**54.32±0.60**	**41.59±1.20**	**60.42±0.86**

Table C.8: Comparison between GCond and AD-GCL under the TTGT framework.

	Amz-Photo	Cora	Elliptic	FB-100	Ogbn-arxiv	Twitch-E
ERM	93.79±0.97	91.59±1.44	50.90±1.51	54.04±0.94	38.59±1.35	59.89±0.50
TTGT+AD-GCL	**94.96±0.52**	92.38±1.35	54.38±2.77	53.81±0.87	39.16±0.98	59.78±0.65
GCond	94.13±0.77	**94.66±0.63**	**55.88±3.10**	**54.32±0.60**	**41.59±1.20**	**60.42±0.86**

C.4.3 Comparison to AD-GCL

Next, we compare our method with a graph contrastive learning method with learnable augmentation AD-GCL [131]. Since AD-GCL is originally designed for graph classification as a pre-training strategy, the direct empirical comparison between AD-GCL and GCond is not easy. However, due to the flexibility of GCond, we can integrate AD-GCL into our TTGT framework, denoted as TTGT+AD-GCL. We present the empirical results in Table C.8. We can observe that TTGT+AD-GCL generally performs worse than GCond except on Amz-Photo, which indicates that GCond is a stronger realization of TTGT. Furthermore, we highlight some key differences between it and GCond.

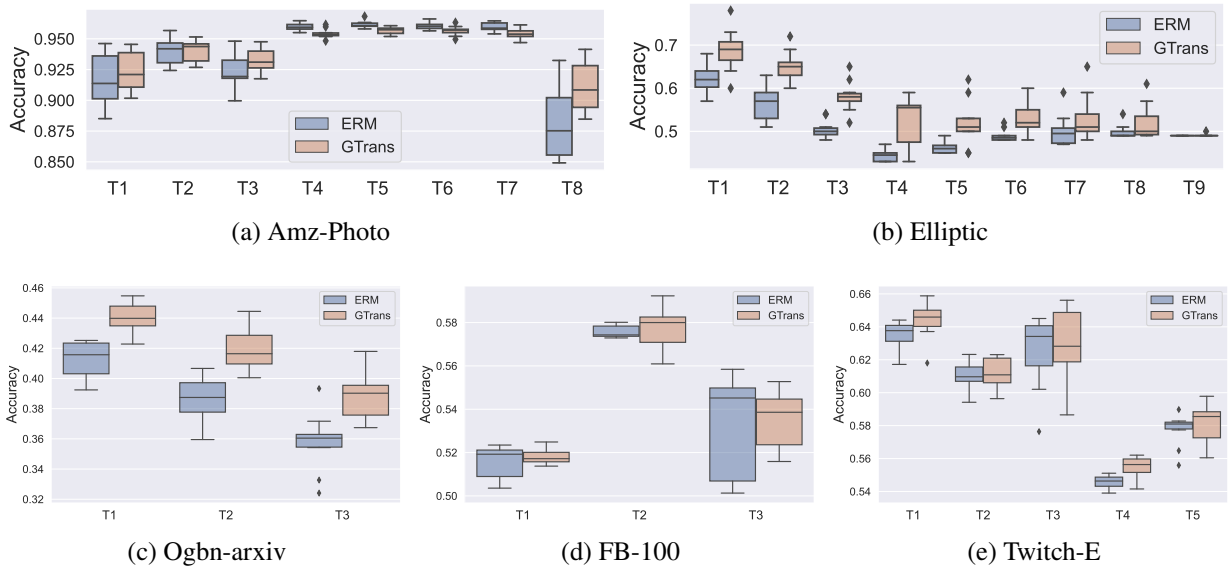
(a) AD-GCL requires optimization of a min-max problem which involves parameters of graph structure and model. Thus, adopting it for TTGT would change the pre-trained model architecture.

(b) AD-GCL only augments the graph structure while not learning the features. We argue that feature learning is highly important under the abnormal feature setting as shown in Table C.12. For example, structure learning only improves GCN by 2% on Ogbn-arxiv while feature learning can improve GCN by 20%. Thus, without the feature learning component, the performance will significantly drop when encountering noisy features.

(c) According to Eq. (9) in the AD-GCL paper, it calculates the similarities between all samples within each mini-batch. When we increase the batch size, we would easily get the out-of-memory issue while a small mini-batch will slow down the learning process. As a consequence, TTGT+AD-GCL is less efficient than GCond: the adaptation time of TTGT+AD-GCL on Ogbn-arxiv is 12.7s while that of GCond is 2.6s.

C.4.4 Out-of-Distribution (OOD) Setting

To show the performance on individual test graphs, we choose GCN as the backbone model and include the box plot on all test graphs within each dataset in Figure C.2. We observe that GCond generally improves over each test graph within each dataset, which validates the effectiveness of test-time graph transformation.



(a) Amz-Photo (b) Elliptic

(c) Ogbn-arxiv (d) FB-100 (e) Twitch-E

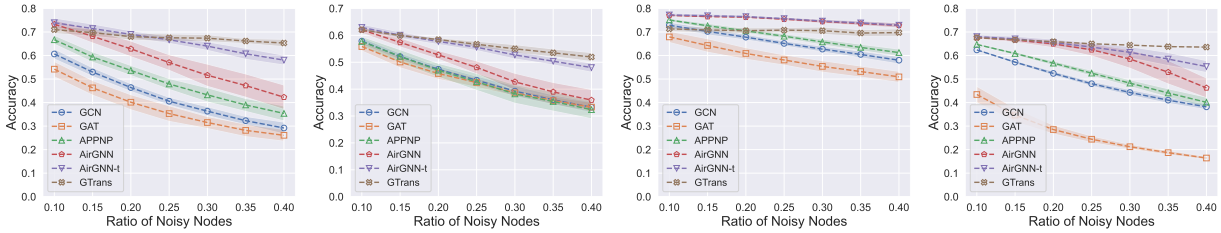
Figure C.2: Classification performance on individual test graphs within each dataset for OOD setting.

C.4.5 Abnormal Features

For each model, we present the node classification accuracy on all test nodes (i.e., both normal and abnormal ones) in Figure C.3. GCond significantly improves GCN in terms of the performance on all test nodes for all datasets across all noise ratios. For example, on Cora with 30% noisy nodes, GCond improves GCN by 31.0% on overall test accuracy. These results further validate that the proposed GCond can produce expressive and generalizable representations.

C.4.6 Interpretation on the Refined Graph for Adversarial Attack Setting

To understand the modifications made on the graph, we compare several properties among clean graph, attacked graph (20% perturbation rate), graph obtained by GCNJaccard, and graph obtained by GCond in Table C.9. We follow the definition in [202] to measure homophily; "Pairwise



(a) Cora (b) Citeseer (c) Pubmed (d) Ogbn-arxiv

Figure C.3: Overall node classification accuracy under the setting of abnormal features. GCond significantly improves the performance of GCN on both abnormal nodes and overall nodes.

Feature Similarity" is the averaged feature similarity among all pairs of connected nodes; "#Edge+/-" indicates the number of edges that the modified graph adds/deletes compared to the clean graph. From Table C.9, we observe that first, adversarial attack decreases homophily and feature similarity, but GCond and GCNJaccard promote such information to defend against adversarial patterns. Second, both GCond and GCNJaccard focus on deleting edges from the attacked graph, but GCNJaccard removes a substantially larger amount of edges, which may destroy clean graph structure and lead to suboptimal performance.

Table C.9: Statistics of modified graphs. GCond promotes homophily and feature similarity.

	GCond	GCNJaccard	Attacked	Clean
Homophily	0.689	0.636	0.548	0.654
Pairwise Feature Similarity	0.825	0.863	0.809	0.827
#Edges	1,945k	1,754k	2,778k	2,316k
#Edge+	108k	118k	463k	-
#Edge-	479k	679k	0.6k	-

C.4.7 Ablation Study on Surrogate Loss

Since we optimized a combined loss in the settings of abnormal features and adversarial attack, we now perform ablation study to examine the effect of each component. We choose GCN as the backbone model and choose 0.3 noise ratio for abnormal features. The results for abnormal features and adversarial attack are shown in Tables C.10 and C.11, respectively. "None" indicates the vanilla GCN without any test-time adaptation and "Combined" indicates jointly optimizing a combination of the two losses. From the two tables, we can conclude that (1) both $\mathcal{L}_s$ and $\mathcal{L}_{\text{train}}$ help

counteract abnormal features and adversarial attack; and (2) optimizing the combined loss generally outperforms optimizing $\mathcal{L}_s$ or $\mathcal{L}_{\text{train}}$ alone.

Table C.10: Performance comparison when optimizing different losses for abnormal feature setting. Both $\mathcal{L}_s$ and $\mathcal{L}_{\text{train}}$ help counteract abnormal features; optimizing the combined loss generally outperforms optimizing $\mathcal{L}_s$ or $\mathcal{L}_{\text{train}}$ alone.

	All Test Nodes				Abnormal Nodes			
Dataset	None	$\mathcal{L}_s$	$\mathcal{L}_{\text{train}}$	Combined	None	$\mathcal{L}_s$	$\mathcal{L}_{\text{train}}$	Combined
Ogbn-arxiv	44.29±1.20	46.70±1.20	64.60±0.22	64.64±0.24	31.50±1.12	35.22±1.17	57.54±0.93	57.69±0.93
Citeseer	39.26±2.02	45.41±2.71	54.97±1.55	52.54±1.08	17.30±1.86	32.93±2.81	42.67±2.78	44.10±2.97
Cora	36.35±1.87	48.71±3.02	66.77±2.54	67.29±1.44	15.80±2.33	35.40±4.05	61.67±3.64	63.90±2.55
Pubmed	62.72±1.20	65.49±1.65	66.56±0.64	70.55±1.55	36.47±1.85	56.77±3.60	60.20±1.97	67.93±2.11

Table C.11: Performance comparison when optimizing different losses for adversarial attack setting. Both $\mathcal{L}_s$ and $\mathcal{L}_{\text{train}}$ help counteract adversarial attack; optimizing the combined loss generally outperforms optimizing $\mathcal{L}_s$ or $\mathcal{L}_{\text{train}}$ alone. r denotes the perturbation rate.

Loss	r=5%	r=10%	r=15%	r=20%	r=25%
None	57.47±0.54	47.97±0.65	38.04±1.22	29.05±0.73	19.58±2.32
$\mathcal{L}_s$	62.40±0.45	59.76±0.93	57.85±1.03	55.26±1.35	52.64±2.35
$\mathcal{L}_{\text{train}}$	65.54±0.25	64.00±0.31	62.99±0.34	61.95±0.40	61.55±0.58
Combined	66.29±0.25	65.16±0.52	64.40±0.38	63.44±0.50	62.95±0.67

C.4.8 Ablation Study on Feature Learning and Structure Learning

In this subsection, we investigate the effects of the feature learning component and structure learning component. We show results for abnormal features and adversarial attack in Tables C.12 and C.13, respectively. Note that "None" indicates the vanilla GCN without any test-time adaptation; "$\mathbf{A}'$" or "$\mathbf{X}'$" is the variants of GCond which solely learns structure or node features; "Both" indicates the method GCond that learn both structure and node features. From Table C.12, we observe that (1) while both feature learning and structure learning can improve the vanilla performance, feature learning is more powerful than structure learning; (2) combining them does not seem to further improve the performance but it achieves a comparable performance to sole feature learning. From Table C.13, we observe that (1) while both feature learning and structure learning can improve the vanilla performance, structure learning is more powerful than feature learning; and (2) combining them can further improve the performance. From these observations, we conclude that (1) feature

learning is more crucial for counteracting feature corruption and structure learning is more important for defending structure corruption; and (2) combining them always yields a better or comparable performance.

Table C.12: Ablation study on feature learning and structure learning for abnormal feature setting. While both feature learning and structure learning can improve the vanilla performance, feature learning is more powerful than structure learning. Combining them does not seem to further improve the performance but it achieves a comparable performance to sole feature learning.

	All Test Nodes				Abnormal Nodes			
Dataset	None	$\mathbf{A}'$	$\mathbf{X}'$	Both	None	$\mathbf{A}'$	$\mathbf{X}'$	Both
Ogbn-arxiv	44.29±1.20	46.02±1.09	64.88±0.23	64.64±0.24	31.50±1.12	31.96±1.05	58.12±0.83	57.69±0.93
Citeseer	39.26±2.02	39.67±1.96	54.99±1.55	54.97±1.55	17.30±1.86	17.13±1.81	42.73±2.81	42.67±2.78
Cora	36.35±1.87	37.02±1.82	67.40±1.62	67.29±1.44	15.80±2.33	15.67±2.15	64.17±3.18	63.90±2.55
Pubmed	62.72±1.20	62.50±1.21	70.53±1.52	70.55±1.55	36.47±1.85	36.57±1.96	67.90±2.07	67.93±2.11

Table C.13: Ablation study on feature learning and structure learning for adversarial structural attack setting. While both feature learning and structure learning can improve the vanilla performance, structure learning is more powerful than feature learning. Combining them can further improve the performance.

Param	r=5%	r=10%	r=15%	r=20%	r=25%
None	57.47±0.54	47.97±0.65	38.04±1.22	29.05±0.73	19.58±2.32
$\mathbf{X}'$	64.16±0.24	61.59±0.29	60.07±0.32	59.04±0.49	58.82±0.68
$\mathbf{A}'$	65.93±0.32	64.31±0.71	63.14±0.39	61.42±0.58	60.18±1.53
Both	66.29±0.25	65.16±0.52	64.40±0.38	63.44±0.50	62.95±0.67

C.4.9 Comparing Different Self-Supervised Signals

As there can be other choices to guide our test-time graph transformation process, we examine the effects of other self-supervised signals. We choose the OOD setting to perform experiments and consider the following two parameter-free self-supervised loss:

(a) **Reconstruction Loss.** Data reconstruction is considered as a good self-supervised signal and we can adopt link reconstruction [77] as the guidance. Minimizing the reconstruction loss is equivalent to maximizing the similarity for connected nodes, which encourages the connected nodes to have similar representations.

(b) **Entropy Loss.** Entropy loss calculates the entropy of the model prediction. Minimizing the entropy can force the model to be certain about the prediction. It has been demonstrated effective

in Tent [142] when adapting batch normalization parameters.

(c) **SLAPS Loss.** SLAPS [41] utilizes self-supervision to guide the graph structure learning process. Specifically, it injects random noise into node features and employs a denoising autoencoder (DAE) to denoise the node features. We refer to the loss used for denoising as SLAPS loss and adopt it for TTGT. Note that we first train the parameters of the DAE used for denoising while keeping the pre-trained model fixed. Then we fix both DAE and the pre-trained model and optimize the test graph for TTGT.

We summarize the results in Table C.14. From the table, we observe that in most of the cases, the above three losses underperform our proposed surrogate loss and even degrade the vanilla performance. It validates the effectiveness of our contrastive loss in guiding the test-time graph transformation.

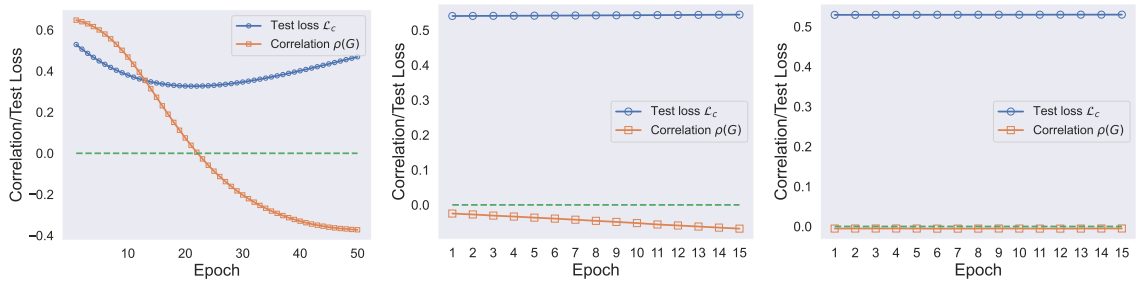
Table C.14: Comparison of different self-supervised signals for OOD setting. The reconstruction loss and entropy loss generally underperform our proposed loss.

	Amz-Photo	Cora	Elliptic	FB-100	Ogbn-arxiv	Twitch-E
None	93.79±0.97	91.59±1.44	50.90±1.51	54.04±0.94	38.59±1.35	59.89±0.50
Recon	93.77±1.01	91.37±1.41	49.33±1.37	53.94±1.03	**44.93±4.06**	59.17±0.77
Entropy	93.67±0.98	91.54±1.14	49.93±1.56	54.29±0.97	41.11±2.19	59.48±0.64
SLAPS	93.97±1.04	91.41±1.23	50.54±1.81	54.08±0.76	41.38±1.35	59.85±0.68
$\mathcal{L}_s$ in Eq. (5.6)	**94.13±0.77**	**94.66±0.63**	**55.88±3.10**	**54.32±0.60**	41.59±1.20	**60.42±0.86**

Gradient Correlation. In Figure 5.2, we have empirically verified the effectiveness of Theorem 1 when adopting the surrogate loss in Eq. (5.6) as $\mathcal{L}_s$. We further plot the values of $\rho(G)$ with different surrogate losses (i.e., entropy, reconstruction and SLAPS) and $\mathcal{L}_c$ on one test graph in Cora in Figure C.4. We can observe that a positive $\rho(G)$ generally reduces the test classification loss. For example, when using entropy loss, the test loss generally reduces when $\rho(G)$ is positive and starts to increase after $\rho(G)$ becomes negative.

C.4.10 Sensitivity to Hyper-Parameter B

In this subsection, we examine the sensitivity of GCond' performance with respect to the perturbation budget, i.e., hyper-parameter B. Specifically, we vary the value of B in the range of $\{0.5\%, 1\%, 5\%, 10\%, 20\%, 30\%\}$ and perform experiments on the Ogbn-arxiv dataset for the three



(a) Entropy Loss (b) Reconstruction Loss (c) SLAPS Loss

Figure C.4: The relationship between $\rho(G)$ and $\mathcal{L}_c$ when adopting different surrogate losses.

settings in Table C.15. Specifically, "Abn. Feat" stands for abnormal feature setting with 30% noisy feature while "Adv. Attack" stands for the adversarial attack setting with 20% perturbation rate. From the table, we can observe budget B has a smaller effect on OOD and abnormal feature settings while highly impacting the performance under structural adversarial attack. This is because most of the changes made by adversarial attack are edge injections as shown in Table C.9, and we need to use a large budget B to remove adversarial patterns. By contrast, GCond is much less sensitive to the value of B in the other two settings.

Table C.15: The change of model performance when varying budget B on Ogbn-arxiv.

Setting	B=0.5%	B=1%	B=5%	B=10%	B=20%	B=30%
OOD	40.52	40.69	41.32	41.40	41.70	41.65
Abn. Feat.	64.78	64.80	64.64	64.60	64.57	64.57
Adv. Attack	56.66	56.89	58.30	59.93	62.31	63.47

C.4.11 Different Augmentations Used in Contrastive Loss

In Eq. (5.6), we used DropEdge as the augmentation function $\mathcal{A}(\cdot)$ to obtain the augmented view. In practice, the choice of augmentation can be flexible and here we explore two other choices: node dropping [172] and subgraph sampling [207]. We perform experiments on OOD setting with GCN as the backbone model and report the results in Table C.16. Specifically, we adopt a ratio of 0.05 for node dropping, and ratios of 0.05 and 0.5 for DropEdge. From the table, we can observe that (1) GCond with any of the three augmentations can greatly improve the performance of GCN under distribution shift, and (2) different augmentations lead to slightly different performance on different

datasets.

Table C.16: Performance of GCond with different augmentation used in contrastive loss.

Augmentation	Amz-Photo	Cora	Elliptic	FB-100	Ogbn-arxiv	Twitch-E
Node Dropping	**94.45±0.70**	95.00±0.65	56.57±2.99	54.15±0.60	39.95±1.11	60.38±0.74
Subgraph Sampling	94.18±0.75	94.95±0.64	55.40±3.00	**54.51±0.56**	41.44±1.17	**60.52±0.80**
DropEdge (0.05)	94.43±0.68	**95.10±0.66**	**56.78±2.86**	54.17±0.60	40.19±1.08	60.31±0.74
DropEdge (0.5)	94.13±0.77	94.66±0.63	55.88±3.10	54.32±0.60	**41.59±1.20**	60.42±0.86
ERM	93.79±0.97	91.59±1.44	50.90±1.51	54.04±0.94	38.59±1.35	59.89±0.50