Condensing Graphs via One-Step Gradient Matching

Wei Jin[1], **Xianfeng Tang**[2], **Haoming Jiang**[2], **Zheng Li**[2], **Danqing Zhang**[2],
Jiliang Tang[1], **Bing Yin**[2]
[1]Michigan State University, [2]Amazon
{jinwei2,tangjili}@msu.edu, {xianft,jhaoming,amzzhe,danqinz,alexbyin}@amazon.com

Abstract

As training deep learning models on large dataset takes a lot of time and resources, it is desired to construct a small synthetic dataset with which we can train deep learning models sufficiently. There are recent works that have explored solutions on condensing image datasets through complex bi-level optimization. For instance, dataset condensation (DC) matches network gradients w.r.t. large-real data and small-synthetic data, where the network weights are optimized for multiple steps at each outer iteration. However, existing approaches have their inherent limitations: (1) they are not directly applicable to graphs where the data is discrete; and (2) the condensation process is computationally expensive due to the involved nested optimization. To bridge the gap, we investigate efficient dataset condensation tailored for graph datasets 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). Our code is available at `https://github.com/amazon-science/doscond`.

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 [1], detect fraud activities in a financial transaction graph [2], or recommend new friends to users in a social network [3]. Due to its prevalence, graph neural networks (GNNs) [4, 5, 6, 7] 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 [8] and continual learning [9].

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 [10, 11, 12, 13, 14, 15, 16]. In particular, one of the representative methods, DC [11], 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

NeurIPS 2022 New Frontiers in Graph Learning Workshop (NeurIPS GLFrontiers 2022)

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 [10, 11, 12, 13] 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.

To address the aforementioned challenges, we propose an efficient condensation method for graphs, where we follow DC [11] 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. 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.
3. 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).

2 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 the 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 $\mathcal{T}$.

2.1 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 [11, 12] introduces a gradient matching scheme to achieve this goal. Concretely, it tries

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

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:

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

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. (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 model parameters P_{θ_0}, we sample $\theta_0 \sim P_{\theta_0}$ and rewrite Eq. (1) as:

$$\min_{\mathcal{S}} \underset{\theta_0 \sim P_{\theta_0}}{E} \left[\sum_{t=0}^{T-1} D\left(\nabla_\theta \ell\left(f_{\theta_t}(\mathcal{S}), \mathcal{Y}'\right), \nabla_\theta \ell\left(f_{\theta_t}(\mathcal{T}), \mathcal{Y}\right)\right) \right], \quad \text{s.t. } \theta_{t+1} = \text{opt}_\theta(\theta_t, \mathcal{S}). \tag{2}$$

Discussion. The aforementioned strategy has demonstrated promising performance on condensing image datasets [11, 12]. 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*.

2.2 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 $\mathbf{X}' \in R^{N' \times n \times d}$ 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. (2) as follows:

$$\min_{\mathbf{A}', \mathbf{X}'} \underset{\theta_0 \sim P_{\theta_0}}{E} \left[\sum_{t=0}^{T-1} D\left(\nabla_\theta \ell\left(f_{\theta_t}(\mathbf{A}', \mathbf{X}'), \mathcal{Y}'\right), \nabla_\theta \ell\left(f_{\theta_t}(\mathcal{T}), \mathcal{Y}\right)\right) \right], \quad \text{s.t. } \theta_{t+1} = \text{opt}_\theta(\theta_t, \mathcal{S}), \tag{3}$$

where we aim to learn both graph structure $\mathbf{A}'$ and node features $\mathbf{X}'$. However, Eq. (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: $P_{\mathbf{\Omega}_{ij}}(\mathbf{A}'_{ij}) = \mathbf{A}'_{ij}\sigma(\mathbf{\Omega}_{ij}) + (1 - \mathbf{A}'_{ij})\sigma(-\mathbf{\Omega}_{ij})$, where $\sigma(\cdot)$ is the sigmoid function; $\mathbf{\Omega}_{ij} \in R$ 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:$P_{\mathbf{\Omega}}(\mathbf{A}') = \prod_{i=1}^{N'} \prod_{j=1}^{n^2} P_{\mathbf{\Omega}_{ij}}\left(\mathbf{A}'_{ij}\right)$. Then, the objective in Eq. (2) needs to be modified to

$$\min_{\mathbf{A}', \mathbf{X}'} \underset{\theta_0 \sim P_{\theta_0}}{E} \left[\underset{\mathbf{A}' \sim P_{\mathbf{\Omega}}}{E} \left[\ell(\mathbf{A}'(\mathbf{\Omega}), \mathbf{X}', \theta_0)\right] \right]. \tag{4}$$

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 [17], 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:

$$\mathbf{A}'_{ij} = \sigma\left(\left(\log \alpha - \log(1-\alpha) + \mathbf{\Omega}_{ij}\right)/\tau\right), \tag{5}$$

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 [17]. In practice, we employ an annealing schedule [18] 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:

$$\min_{\mathbf{\Omega}, \mathbf{X}'} \underset{\theta_0 \sim P_{\theta_0}}{E} \left[\underset{\alpha \sim \text{Uniform}(0,1)}{E} \left[\ell(\mathbf{A}'(\mathbf{\Omega}), \mathbf{X}', \theta_0) \right] \right] = \tag{6}$$
$$\underset{\theta_0}{E} \left[\underset{\alpha}{E} \left[\sum_{t=0}^{T-1} D\left(\nabla_\theta \ell\left(f_{\theta_t}(\mathbf{A}'(\mathbf{\Omega}), \mathbf{X}'), \mathcal{Y}' \right), \nabla_\theta \ell\left(f_{\theta_t}(\mathcal{T}), \mathcal{Y} \right) \right) \right] \right], \quad \text{s.t. } \theta_{t+1} = \text{opt}_\theta(\theta_t, \mathcal{S}).$$

2.3 One-Step Gradient Matching

The vanilla gradient matching scheme in Eq. (2) 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. (6) and simplify Eq. (6) as follows:

$$\min_{\mathbf{\Omega}, \mathbf{X}'} \underset{\theta_0}{E} \left[\underset{\alpha}{E} \left[D\left(\nabla_\theta \ell\left(f_{\theta_0}(\mathbf{A}'(\mathbf{\Omega}), \mathbf{X}'), \mathcal{Y}' \right), \nabla_\theta \ell\left(f_{\theta_0}(\mathcal{T}), \mathcal{Y} \right) \right) \right] \right], \tag{7}$$

where we sample $\theta_0 \sim P_{\theta_0}$ and $\alpha \sim \text{Uniform}(0, 1)$. Compared with Eq. (6), 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 3.3, 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) [19] 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*

$$\min_t \ell_\mathcal{T}(\theta_t) - \ell_\mathcal{T}(\theta^*) \leq \sum_{t=0}^{T-1} \frac{\sqrt{2}M}{T} \|\nabla_\theta \ell_\mathcal{T}(\theta_t) - \nabla_\theta \ell_\mathcal{S}(\theta_t)\| \frac{3M}{2\sqrt{T}} \frac{C-1}{CN'} \sqrt{\sum_i \gamma_i \|\mathbf{1}^\top \mathbf{A}'^K_{(i)} \mathbf{X}'_{(i)}\|^2} \tag{8}$$

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 C.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.,* $\|\nabla_\theta \ell_{\mathcal{T}}(\theta_0) - \nabla_\theta \ell_S(\theta_0)\| = \max_t \|\nabla_\theta \ell_{\mathcal{T}}(\theta_t) - \nabla_\theta \ell_S(\theta_t)\|$ *with* $t = 0, 1, \ldots, T-1$*, we have*

$$\min_t \ell_{\mathcal{T}}(\theta_t) - \ell_{\mathcal{T}}(\theta^*) \leq \sqrt{2}M\|\nabla_\theta \ell_{\mathcal{T}}(\theta_0) - \nabla_\theta \ell_S(\theta_0)\| + \frac{3M}{2\sqrt{T}}\frac{C-1}{CN'}\sqrt{\sum_i \gamma_i \|\mathbf{1}^\top \mathbf{A}'^K_{(i)}\mathbf{X}'_{(i)}\|^2}. \quad (9)$$

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 B.1, when using mean pooling, the second term tend 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, when 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 theoretically.

Remark 1. Note that the spectral analysis from [19] 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 it 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. We obtain similar conclusions for node classification as shown in Theorem 2 in Appendix C.2 and achieve impressive empirical performance in Appendix B.2.

2.4 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. (6) involves two nested expectations, we adopt Monte Carlo to approximately optimize the objective function. Together with one-step gradient matching, we have

$$\min_{\mathbf{\Omega},\mathbf{X}'} \underset{\theta_0}{E} \underset{\alpha\sim\text{Uniform}(0,1)}{E} [[\ell(\mathbf{A}'(\mathbf{\Omega}),\mathbf{X}',\theta_0)]] \approx \sum_{k_1=1}^{K_1}\sum_{k_2=1}^{K_2} D\left(\nabla_\theta \ell\left(f_{\theta_0}(\mathbf{A}'(\mathbf{\Omega}),\mathbf{X}'),\mathcal{Y}'\right), \nabla_\theta \ell\left(f_{\theta_0}(\mathcal{T}),\mathcal{Y}\right)\right)$$

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

$$\ell_{\text{reg}} = \max(\frac{1}{|\mathbf{\Omega}|}\sum_{i,j}\sigma(\mathbf{\Omega}_{ij}) - \epsilon, 0). \quad (10)$$

We initialize $\sigma(\mathbf{\Omega})$ and $\mathbf{X}'$ as randomly sampled training graphs 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 2.3, we include the following regularization for GNNs with sum pooling:

$$\ell_{\text{reg2}} = \frac{3}{2\sqrt{2T}} \cdot \frac{C-1}{CN'}\sqrt{\sum_i \|\mathbf{1}^\top \mathbf{A}'^K_{(i)}\mathbf{X}'_{(i)}\|^2} \quad (11)$$

Training Algorithm. We provide the details of our proposed framework in Algorithm 1 in Appendix A.3. Specifically, we sample K_1 model initializations θ_0 to perform one-step gradient matching. Following the convention in DC [11], 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 save us enormous training time when learning larger synthetic sets.

Comparison with Coreset Methods. Coreset methods [20, 21] 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 3.2.

3 Experiment

3.1 Experimental settings

Datasets. To evaluate the performance of our method, we use multiple molecular datasets from Open Graph Benchmark (OGB) [22] and TU Datasets (DD, MUTAG and NCI1) [23] for graph-level property classification, and one superpixel dataset CIFAR10 [24]. 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 A.2.

Baselines. We compare our proposed methods with four baselines that produce discrete structures: three coreset methods (*Random*, *Herding* [20] and *K-Center* [25, 21]), and a *dataset condensation* method DCG [11]: (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 [26, 27]. (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 [11] 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 DC [11] 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. We report the mean and standard deviation of these results.

3.2 Performance with Condensed Graphs

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 [22], 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 1. Note that the *Ratio* column presents the ratio of synthetic graphs to original graphs and we name it as *condensation ratio*; the

Table 1: The classification performance comparison. 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**	

Whole Dataset column shows the GCN performance achieved by training on the original dataset. From the table, we make four 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 3.3.

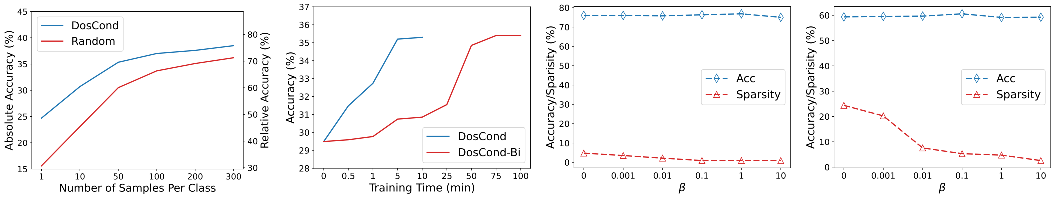
(c) The performance on CIFAR10 is less promising due to the limit number of synthetic graphs. We posit that the dataset has more complex topology and feature information and thus requires more parameters to preserve sufficient information. However, we note that our method still outperforms the baseline 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 3.3.

(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.

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 1: 1000 iterations for *DosCond*, i.e., $K_1 = 1000$, and 500 epochs (100 epochs 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

Table 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



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

Figure 1: Algorithm analysis and parameter analysis w.r.t. the sparsity regularization.

epoch while *DosCond* only processes a batch of graphs. In Table 2, we report the running time on an NVIDIA V100 GPU for CIFAR10, ogbg-molhiv and DD. We make three 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.

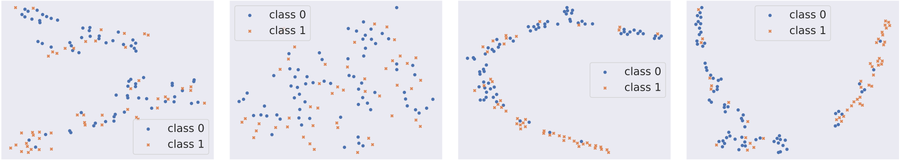
(c) The average nodes in synthetic graph n also impacts the training cost of *DosCond*. For instance, the training cost on ogbg-molhiv (n=26) is much lower than that on DD (n=285), and the gap of cost between the two methods on ogbg-molhiv and DD is very different. As mentioned earlier, it is because 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.

3.3 Further Investigation

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 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. Note 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 viewed as the upper bound of *DosCond*.

Ablation Study. 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 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 1c and 1d. 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



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

Figure 2: T-SNE visualizations of embedding learned with condensed graphs on DD.

property since sparse graphs can save much space for storage and reduce training cost for GNNs. We also remove the regularization of Eq. (11) for ogbg-molhiv, we obtain the performance of 0.724/ 0.727/0.731 for 1/10/50 graphs per class, slightly worse than the one with this regularization.

Visualization. We further investigate whether GCN can learn discriminative representations from the synthetic graphs learned by *DosCond*. Specifically, we use t-SNE [28] 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 2. It is observed that the graph representations learned with randomly selected graphs are mixed for different classes. 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 2c) and they are as discriminative as those learned on the whole dataset (Figure 2d). This demonstrates that the graphs learned by *DosCond* preserve sufficient information of the original dataset so as to recover the original performance.

4 Related Work

Graph Neural Networks. As the generalization of deep neural network to graph data, graph neural networks (GNNs) [4, 29, 5, 7, 19, 30, 31, 32, 33] 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 [34, 35], link prediction [3] and node classification [4]. Recent years have also witnessed their great success achieved in many real-world applications such as recommender systems [3, 36], computer vision [37], drug discovery [38] and single-cell analysis [39]. 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 [40] where each node first aggregates the information from its neighborhood and then transforms the aggregated information to update its representation.

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) [10] 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) [11, 12] 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 [11, 12, 41, 42, 43], neural architecture search [13, 14, 44] and privacy-preserving scenarios [45]. Recently, following the gradient matching scheme in DC, *GCond* [46] proposes to condense a large-scale graph to a small graph for node classification. Different from *GCond*, we aim to solve the challenge of learning discrete structure and we majorly target at graph classification. Our method avoids the costly bi-level optimization and is much more efficient than the previous work. A detailed comparison is included in Appendix B.2.

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.

References

[1] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

[2] 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.

[3] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Yihong Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *WWW*, 2019.

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

[5] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[6] 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.

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

[8] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *ICLR*, 2019.

[9] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

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

[11] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. In *ICLR*, 2021.

[12] Bo Zhao and Hakan Bilen. Dataset condensation with differentiable siamese augmentation. In *ICML*, Proceedings of Machine Learning Research, 2021.

[13] Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset meta-learning from kernel ridge-regression. In *ICLR*, 2021.

[14] Timothy Nguyen, Roman Novak, Lechao Xiao, and Jaehoon Lee. Dataset distillation with infinitely wide convolutional networks. *NeurIPS*, 34, 2021.

[15] George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A Efros, and Jun-Yan Zhu. Dataset distillation by matching training trajectories. In *CVPR*, 2022.

[16] 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 *CVPR*, 2022.

[17] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[18] Abubakar Abid, Muhammad Fatih Balin, and James Zou. Concrete autoencoders for differentiable feature selection and reconstruction. *arXiv preprint arXiv:1901.09346*, 2019.

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

[20] Max Welling. Herding dynamical weights to learn. In *ICML*, 2009.

[21] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *ICLR*, 2018.

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

[23] 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 arXiv:2007.08663*, 2020.

[24] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

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

[26] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. In *CVPR*, 2017.

[27] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *ECCV*, 2018.

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

[29] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR 2019*, 2019.

[30] Xianfeng Tang, Yandong Li, Yiwei Sun, Huaxiu Yao, Prasenjit Mitra, and Suhang Wang. Transferring robustness for graph neural network against poisoning attacks. In *WSDM*, 2020.

[31] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *KDD*, 2020.

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

[33] 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.

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

[35] 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*, pages 2559–2567, 2021.

[36] 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*, pages 112–121, 2022.

[37] Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, 2019.

[38] 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 *NeurIPS*, 2015.

[39] 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.

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

[41] 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:2205.14959*, 2022.

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

[43] Bo Zhao and Hakan Bilen. Dataset condensation with distribution matching. *arXiv preprint arXiv:2110.04181*, 2021.

[44] Shuo Yang, Zeke Xie, Hanyu Peng, Min Xu, Mingming Sun, and Ping Li. Dataset pruning: Reducing training data by examining generalization influence. *arXiv preprint arXiv:2205.09329*, 2022.

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

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

[47] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. Graphsaint: Graph sampling based inductive learning method. In *ICLR*, 2020.

[48] Krishnateja Killamsetty, Durga S, Ganesh Ramakrishnan, Abir De, and Rishabh Iyer. Grad-match: Gradient matching based data subset selection for efficient deep model training. In *ICML*. PMLR, 2021.

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

[50] Rahul Yedida, Snehanshu Saha, and Tejas Prashanth. Lipschitzlr: Using theoretically computed adaptive learning rates for fast convergence. *Applied Intelligence*, 51(3):1460–1478, 2021.

A Experimental Setup

A.1 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.

A.2 Dataset Statistics

Dataset statistics for node classification and graph classification are shown in Table 3 and 4, respectively.

Table 3: 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 4: 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

A.3 Algorithm

We provide the details of our proposed framework in Algorithm 1. Specifically, we sample K_1 model initializations θ_0 to perform one-step gradient matching. Following the convention in DC [11], 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.

B More Experiments

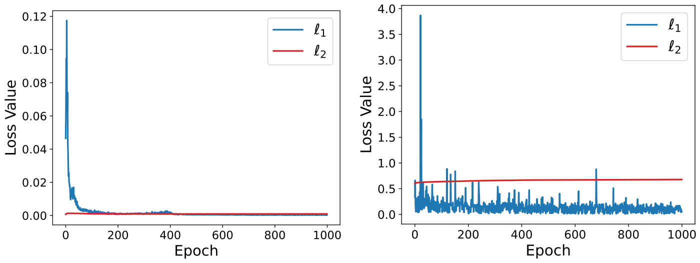
B.1 Scale of the two terms in Eq. (9).

As mentioned earlier in Section 2.3, the scale of the first term is essentially larger than the second term in Eq. (9). 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

Algorithm 1: *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 $f_\theta$, temperature $\tau$, desired sparsity $\epsilon$, regularization coefficient $\beta$, learning rates $\eta_1, \eta_2$, 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 $\alpha \sim \mathrm{Uniform}(0, 1)$
    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 $\mathbf{\Omega} \leftarrow \mathbf{\Omega} - \eta_1 \nabla_{\mathbf{\Omega}}(D(\nabla_{\boldsymbol{\theta}_0}\ell_T, \nabla_{\boldsymbol{\theta}_0}\ell_S) + \beta\ell_{\text{reg}})$
        Update $\mathbf{X}' \leftarrow \mathbf{X}' - \eta_2 \nabla_{\mathbf{X}'}(D(\nabla_{\boldsymbol{\theta}_0}\ell_T, \nabla_{\boldsymbol{\theta}_0}\ell_S) + \beta\ell_{\text{reg}})$
    end for
end for
Return: $(\mathbf{\Omega}, \mathbf{X}', \mathcal{Y}')$
```

$\ell_2 = \frac{3}{2\sqrt{T}} \cdot \frac{C-1}{CN'}\sqrt{\sum_i \gamma_i \|\mathbf{1}^\top \mathbf{A}'^{K}_{(i)} \mathbf{X}'_{(i)}\|^2}$. 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. 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 3a. By contrast, ℓ_2 is not negligible when using sum pooling as shown in Figure 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 2.3.



(a) DD

(b) ogbg-molhiv

Figure 3: Scale of the two terms in Eq. (11).

B.2 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 C.2. Specifically, following *GCond* [46], a condensation method for node classification, we use 5 node classification datasets: Cora, Citeseer, Pubmed [4], ogbn-arxiv [22] and Flickr [47]. The dataset statistics are shown in 4. 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 5. 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

Table 5: 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

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.

C Proofs

C.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 [48].

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*

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

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.

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:

$$f_\theta(\mathbf{A}_{(i)}, \mathbf{X}_{(i)}) = \begin{cases} \mathbf{1}^\top \mathbf{A}^K_{(i)} \mathbf{X}_{(i)} \mathbf{W}_1 \mathbf{W}_2 & \text{if use sum pooling,} \\ \frac{1}{n_i} \mathbf{1}^\top \mathbf{A}^K_{(i)} \mathbf{X}_{(i)} \mathbf{W}_1 \mathbf{W}_2 & \text{if use mean pooling,} \end{cases} \tag{13}$$

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 [49], we know that softmax classification $f(\mathbf{W}) = \mathbf{XW}$ with cross entropy loss is convex w.r.t. the parameters $\mathbf{W}$. In our case, the mapping function $f_\theta(\mathbf{A}_{(i)}, \mathbf{X}_{(i)}) = \mathbf{1}^\top \mathbf{A}_{(i)}^K \mathbf{X}_{(i)} \mathbf{W}$ applies an affine function on $\mathbf{XW}$. 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 [50], 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:

$$\nabla_\theta \ell_S(\theta) \leq \frac{C-1}{CN'} \sqrt{\sum_i \|\mathbf{1}^\top \mathbf{A}'^{K}_{(i)} \mathbf{X}'_{(i)}\|^2} \tag{14}$$

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

$$\ell_{\mathcal{T}}\left(\theta_t\right) - \ell_{\mathcal{T}}\left(\theta^*\right) \leq \nabla_\theta \ell_{\mathcal{T}}\left(\theta_t\right)^T \left(\theta_t - \theta^*\right) \tag{15}$$

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

$$\begin{aligned}\nabla_\theta \ell_{\mathcal{T}}\left(\theta_t\right)^T \left(\theta_t - \theta^*\right) &= \left(\nabla_\theta \ell_{\mathcal{T}}\left(\theta_t\right)^T - \nabla_\theta \ell_S\left(\theta_t\right)^T + \nabla_\theta \ell_S\left(\theta_t\right)^T\right)\left(\theta_t - \theta^*\right) \\ &= \left(\nabla_\theta \ell_{\mathcal{T}}\left(\theta_t\right)^T - \nabla_\theta \ell_S\left(\theta_t\right)^T\right)\left(\theta_t - \theta^*\right) + \nabla_\theta \ell_S\left(\theta_t\right)^T \left(\theta_t - \theta^*\right)\end{aligned} \tag{16}$$

Given that we use gradient descent to update network parameters, we have $\nabla_\theta \ell_S\left(\theta_t\right) = \frac{1}{\eta}(\theta_t - \theta_{t+1})$ where η is the learning rate. Then we have,

$$\begin{aligned}\nabla_\theta \ell_S\left(\theta_t\right)^T \left(\theta_t - \theta^*\right) &= \frac{1}{\eta}\left(\theta_t - \theta_{t+1}\right)^T \left(\theta_t - \theta^*\right) \\ &= \frac{1}{2\eta}\left(\left\|\theta_t - \theta_{t+1}\right\|^2 + \left\|\theta_t - \theta^*\right\|^2 - \left\|\theta_{t+1} - \theta^*\right\|^2\right) \\ &= \frac{1}{2\eta}\left(\left\|\eta \nabla_\theta \ell_S\left(\theta_t\right)\right\|^2 + \left\|\theta_t - \theta^*\right\|^2 - \left\|\theta_{t+1} - \theta^*\right\|^2\right)\end{aligned} \tag{17}$$

Combining Eq. (15) and Eq. (17) we have,

$$\begin{aligned}\ell_{\mathcal{T}}\left(\theta_t\right) - \ell_{\mathcal{T}}\left(\theta^*\right) &\leq \left(\nabla_\theta \ell_{\mathcal{T}}\left(\theta_t\right)^T - \nabla_\theta \ell_S\left(\theta_t\right)^T\right)\left(\theta_t - \theta^*\right) \\ &+ \frac{1}{2\eta}\left(\left\|\eta \nabla_\theta \ell_S\left(\theta_t\right)\right\|^2 + \left\|\theta_t - \theta^*\right\|^2 - \left\|\theta_{t+1} - \theta^*\right\|^2\right)\end{aligned} \tag{18}$$

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

$$\begin{aligned}\sum_{t=0}^{T-1} \ell_{\mathcal{T}}\left(\theta_t\right) - \ell_{\mathcal{T}}\left(\theta^*\right) &\leq \sum_{t=0}^{T-1}\left(\nabla_\theta \ell_{\mathcal{T}}\left(\theta_t\right)^T - \nabla_\theta \ell_S\left(\theta_t\right)^T\right)\left(\theta_t - \theta^*\right) \\ &+ \frac{1}{2\eta}\sum_{t=0}^{T-1}\left\|\eta \nabla_\theta \ell_S\left(\theta_t\right)\right\|^2 + \frac{1}{2\eta}\left\|\theta_0 - \theta^*\right\|^2 - \frac{1}{2\eta}\left\|\theta_T - \theta^*\right\|^2\end{aligned} \tag{19}$$

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

$$\begin{aligned}\sum_{t=0}^{T-1} \ell_{\mathcal{T}}\left(\theta_t\right) - \ell_{\mathcal{T}}\left(\theta^*\right) &\leq \sum_{t=0}^{T-1}\left(\nabla_\theta \ell_{\mathcal{T}}\left(\theta_t\right)^T - \nabla_\theta \ell_S\left(\theta_t\right)^T\right)\left(\theta_t - \theta^*\right) \\ &+ \frac{1}{2\eta}\sum_{t=0}^{T-1}\left\|\eta \nabla_\theta \ell_S\left(\theta_t\right)\right\|^2 + \frac{1}{2\eta}\left\|\theta_0 - \theta^*\right\|^2\end{aligned} \tag{20}$$

As we assume that $\|\theta\|^2 \leq M^2$, we have $\|\theta - \theta^*\|^2 \leq 2\|\theta\|^2 = 2M^2$. Then Eq. (20) can be rewritten as,

$$\begin{aligned}\sum_{t=0}^{T-1} \ell_T\left(\theta_t\right) - \ell_T\left(\theta^*\right) &\leq \sum_{t=0}^{T-1} \sqrt{2}M\|\nabla_\theta \ell_T\left(\theta_t\right) - \nabla_\theta \ell_S\left(\theta_t\right)\| \\ &+ \frac{1}{2\eta}\sum_{t=0}^{T-1}\left\|\eta \nabla_\theta \ell_S\left(\theta_t\right)\right\|^2 + \frac{M^2}{\eta}\end{aligned} \tag{21}$$

Recall that $\ell_S(\theta)$ is lipschitz continuous as shown in Eq. (14), and combine $\min_{t=0,1,\ldots,T-1}\left(\ell_{\mathcal{T}}\left(\theta_t\right)-\ell_{\mathcal{T}}\left(\theta^*\right)\right) \leq \frac{\sum_{t=0}^{T-1}\ell_{\mathcal{T}}(\theta_t)-\ell_{\mathcal{T}}(\theta^*)}{T}$:

$$\begin{aligned}\min_{t=0,1,\ldots,T-1} \ell_{\mathcal{T}}\left(\theta_t\right) - \ell_{\mathcal{T}}\left(\theta^*\right) \leq & \sum_{t=0}^{T-1} \frac{\sqrt{2}M}{T}\left\|\nabla_\theta \ell_{\mathcal{T}}\left(\theta_t\right)-\nabla_\theta \ell_S\left(\theta_t\right)\right\| \\ & + \frac{\eta(C-1)^2}{2C^2N'^2}\sum_i \|\mathbf{1}^\top \mathbf{A}'^K_{(i)}\mathbf{X}'_{(i)}\|^2 + \frac{M^2}{T\eta}\end{aligned} \tag{22}$$

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:

$$\begin{aligned}\min_{t=0,1,\ldots,T-1} \ell_{\mathcal{T}}\left(\theta_t\right) - \ell_{\mathcal{T}}\left(\theta^*\right) \leq & \sum_{t=0}^{T-1} \frac{\sqrt{2}M}{T}\left\|\nabla_\theta \ell_{\mathcal{T}}\left(\theta_t\right)-\nabla_\theta \ell_S\left(\theta_t\right)\right\| \\ & + \frac{3M}{2\sqrt{T}}\cdot\frac{C-1}{CN'}\sqrt{\sum_i \|\mathbf{1}^\top \mathbf{A}'^K_{(i)}\mathbf{X}'_{(i)}\|^2}\end{aligned} \tag{23}$$

II. For f_θ with mean pooling:

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

$$\begin{aligned}\min_{t=0,1,\ldots,T-1} \ell_{\mathcal{T}}\left(\theta_t\right) - \ell_{\mathcal{T}}\left(\theta^*\right) \leq & \sum_{t=0}^{T-1} \frac{\sqrt{2}M}{T}\left\|\nabla_\theta \ell_{\mathcal{T}}\left(\theta_t\right)-\nabla_\theta \ell_S\left(\theta_t\right)\right\| \\ & + \frac{3M}{2\sqrt{T}}\cdot\frac{C-1}{CN'}\sqrt{\sum_i \frac{1}{n_i}\|\mathbf{1}^\top \mathbf{A}'^K_{(i)}\mathbf{X}'_{(i)}\|^2}\end{aligned} \tag{24}$$

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

C.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 $\mathbf{X} \in R^{N\times d}$, $\mathbf{X}' \in R^{N'\times d}$ 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*

$$\begin{aligned}\min_{t=0,1,\ldots,T-1} \ell_{\mathcal{T}}\left(\theta_t\right) - \ell_{\mathcal{T}}\left(\theta^*\right) \leq & \sum_{t=0}^{T-1} \frac{\sqrt{2}M}{T}\left\|\nabla_\theta \ell_{\mathcal{T}}\left(\theta_t\right)-\nabla_\theta \ell_S\left(\theta_t\right)\right\| \\ & + \frac{3M}{2\sqrt{T}}\cdot\frac{C-1}{CN'}\|\mathbf{A}'^K\mathbf{X}'\|\end{aligned} \tag{25}$$

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 [50], 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. (21)). 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 $\min_t \left(\ell_{\mathcal{T}}\left(\theta_t\right) - \ell_{\mathcal{T}}\left(\theta^*\right)\right) \leq \frac{\sum_{t=0}^{T-1} \ell_{\mathcal{T}}(\theta_t) - \ell_{\mathcal{T}}(\theta^*)}{T}$:

$$\begin{aligned}\min_{t=0,1,\ldots,T-1} \ell_{\mathcal{T}}\left(\theta_t\right) - \ell_{\mathcal{T}}\left(\theta^*\right) \leq & \sum_{t=0}^{T-1} \frac{\sqrt{2}M}{T} \|\nabla_\theta \ell_{\mathcal{T}}\left(\theta_t\right) - \nabla_\theta \ell_S\left(\theta_t\right)\| \\ & + \frac{\eta(C-1)^2}{2C^2{N'}^2}\|\mathbf{A'}^K\mathbf{X}'\|^2 + \frac{M^2}{T\eta} \end{aligned} \tag{26}$$

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

$$\begin{aligned}\min_{t=0,1,\ldots,T-1} \ell_{\mathcal{T}}\left(\theta_t\right) - \ell_{\mathcal{T}}\left(\theta^*\right) \leq & \sum_{t=0}^{T-1} \frac{\sqrt{2}M}{T} \|\nabla_\theta \ell_{\mathcal{T}}\left(\theta_t\right) - \nabla_\theta \ell_S\left(\theta_t\right)\| \\ & + \frac{3M}{2\sqrt{T}} \cdot \frac{C-1}{CN'}\|\mathbf{A'}^K\mathbf{X}'\| \end{aligned} \tag{27}$$