Dink-Net: Neural Clustering on Large Graphs

Yue Liu [1 2] **Ke Liang** [1] **Jun Xia** [2] **Sihang Zhou** [1] **Xihong Yang** [1] **Xinwang Liu** [1 †] **Stan Z. Li** [2 †]

Abstract

Deep graph clustering, which aims to group the nodes of a graph into disjoint clusters with deep neural networks, has achieved promising progress in recent years. However, the existing methods fail to scale to the large graph with million nodes. To solve this problem, a scalable deep graph clustering method (*Dink-Net*) is proposed with the idea of dilation and shrink. Firstly, by discriminating nodes, whether being corrupted by augmentations, representations are learned in a self-supervised manner. Meanwhile, the cluster centers are initialized as learnable neural parameters. Subsequently, the clustering distribution is optimized by minimizing the proposed cluster dilation loss and cluster shrink loss in an adversarial manner. By these settings, we unify the two-step clustering, i.e., representation learning and clustering optimization, into an end-to-end framework, guiding the network to learn clustering-friendly features. Besides, *Dink-Net* scales well to large graphs since the designed loss functions adopt the mini-batch data to optimize the clustering distribution even without performance drops. Both experimental results and theoretical analyses demonstrate the superiority of our method. Compared to the runner-up, *Dink-Net* achieves 9.62% NMI improvement on the ogbn-papers100M dataset with 111 million nodes and 1.6 billion edges. The source code is released: *Dink-Net*[I]. Besides, a collection (papers, codes, and datasets) of deep graph clustering is shared on GitHub[II].

[†]Corresponding Author [1]National University of Defense Technology [2]Westlake University. Email: Yue Liu <yueliu19990731@163.com>, Xinwang Liu <xinwangliu@nudt.edu.cn>, Stan Z. Li <Stan.ZQ.Li@westlake.edu.cn>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

[I]https://github.com/yueliu1999/Dink-Net

[II]https://github.com/yueliu1999/Awesome-Deep-Graph-Clustering

1. Introduction

Attribute graph clustering, which aims to separate the nodes in an attribute graph into different groups, has become a fast-growing research direction in recent years. As a pure unsupervised mission, promising achievements are made by models based on deep neural networks, especially graph neural networks (GNN) (Kipf & Welling, 2017; 2016; Veličković et al., 2018). Multiple models are proposed (Wang et al., 2017; Hassani & Khasahmadi, 2020; Peng et al., 2021; Liu et al., 2022b; Gong et al., 2022; Devvrit et al., 2022), which generally first embed nodes into the hidden space and then perform clustering algorithms on them. Although proven effective, most existing models in such a manner suffer from scalability issues, i.e., poor scalability on large graphs. Such a problem is one of the most critical challenging tasks in deep graph clustering (Liu et al., 2022c), and our work attempts to seek a breakthrough according to it.

Based on our observations, there are three main reasons for the poor scalability of existing models on large graphs with millions of nodes. Firstly, some methods (Cui et al., 2020; Hassani & Khasahmadi, 2020; Liu et al., 2022b; Gong et al., 2022) need to process the $N \times N$ dense graph diffusion matrix (Klicpera et al., 2019), where N is the number of nodes. Secondly, unlike other tasks like node classification or link prediction, clustering requires the method to estimate the whole sample distribution at once. Therefore, when the node number grows to a considerable value, e.g., 100 million, it easily leads to out-of-memory failure or long-running time problems. Thirdly, current attempts for this problem, such as S^3GC (Devvrit et al., 2022), usually separate the optimization of representation learning and clustering, leading to sub-optimal performance.

To solve this problem, we present a scalable deep graph clustering method termed dilation shrink network (*Dink-Net*). The guidance idea is motivated by the dilation and shrink of many galaxies in the universe. Concretely, it mainly consists of the node discriminate module and the neural clustering module. At first, in the node discriminate module, node representations are learned by telling apart the original samples and augmented samples. After the pre-training, the cluster centers are initialized and assigned as optimizable neural parameters with gradients. Additionally, at fine-tuning stage, the neural clustering module optimizes the clustering dis-

tribution by minimizing the proposed dilation and shrink loss functions. Concretely, in an adversarial manner, the dilation loss attempts to expand clustering distribution by pushing away different clusters, while the shrink loss aims to compress the clustering distribution by pulling samples back to cluster centers.

With the above designs, *Dink-Net* learns the clustering-friendly representations via unifying the representation learning and clustering optimization into an end-to-end framework. In addition, the proposed loss functions effectively optimize clustering distribution on mini-batch data. Therefore, it endows the scalability of our method without performance drops. Notably, *Dink-Net* is scaled on a large graph like ogbn-papers100M with 111 million nodes and 1.6 billion edges. The main contributions are summarized as follows.

- A scalable deep graph clustering method named dilation shrink network (*Dink-Net*) is proposed to expand the deep graph clustering to large-scale graphs.
- We are the first to optimize the clustering distribution via the designed dilation and shrink loss functions in an adversarial manner. The method only relies on the mini-batch data, thus, endowing promising scalability.
- We unify the representation learning and clustering optimization procedures into an end-to-end framework for better clustering-friendly features, leading to superior clustering performance.
- Both experimental results and theoretical analyses are provided to verify the capability of *Dink-Net* from six aspects, *i.e.,* superiority, effectiveness, scalability, efficiency, sensitivity, and convergence.

2. Related Work

2.1. Deep Graph Clustering

Graph Neural Networks (Kipf & Welling, 2017; Veličković et al., 2018; Kipf & Welling, 2016; Liu et al., 2022a) become popular in different graph scenarios (Liang et al., 2022b;a; Meng et al., 2023; Liang et al., 2023b; Liu et al., 2023a; Liang et al., 2023a). Among these, attribute graph clustering is a fundamental yet challenging task to separate the nodes in the attribute graph into different clusters without human annotations.

The early methods (Tian et al., 2014; Cao et al., 2016) adopt auto-encoders to learn node embeddings and then perform K-Means on them. Subsequently, motivated by the success of graph neural networks (GNNs) (Kipf & Welling, 2016; 2017), MGAE (Wang et al., 2017) is proposed to encode nodes with graph-auto-encoders and then group the nodes into clusters with the spectral clustering algorithm. (Pan et al., 2018) propose ARGA by enforcing the latent representations to align a prior distribution. To design a clustering-directed method, they propose a unified framework termed DAEGC (Wang et al., 2019) with the attention-based graph encoder and clustering alignment loss adopted in deep clustering methods (Xie et al., 2016). SDCN (Bo et al., 2020) verifies the effectiveness of integrating structural and attribute information. Then, to avoid the expensive costs of spectral clustering, (Bianchi et al., 2020) formulate a continuous relaxation of the normalized minCUT problem and optimize the clustering objective with the GNNs. More recently, the contrastive learning (Mo et al., 2022; 2023; Zheng et al., 2022a;c) become hot research hot in deep graph clustering domain (Yang et al., 2023; 2022b;a). Concretely, AGE (Cui et al., 2020) filters the high-frequency noises in node attributes and then trains the encoder by adaptively contrasting the positive and negative samples. MVGRL (Hassani & Khasahmadi, 2020) generates an augmented structural view and contrasts node embeddings from one view with graph embeddings of another view and vice versa. Although the effectiveness of the contrastive learning paradigm is verified, there are still many open technical problems. Specifically, (Zhao et al., 2021) proposes GDCL to correct sampling bias in contrastive deep graph clustering. Moreover, (Liu et al., 2022b;e) design the dual correlation reduction strategy in the DCRN model to alleviate the representation collapse problem. Besides, HSAN (Liu et al., 2023d) mines the hard sample pairs via the dynamic weighting strategy. And SCGC (Liu et al., 2023c) simplifies the graph augmentation with parameter-unshared Siamese encoders and embedding disturbance. TGC (Liu et al., 2023b) present a general framework for deep node clustering on temporal graphs. For more details about deep graph clustering, refer to the survey paper (Liu et al., 2022c). However, most previous methods fail to scale to large graphs with millions of nodes. In order to alleviate this problem, a scalable deep graph clustering method termed S^3GC (Devvrit et al., 2022) is proposed by contrastive learning along with GNNs. Although verifying the effectiveness, they separate the optimization of representation learning and clustering, leading to sub-optimal performance. This paper presents a new scalable method that unifies embedding and clustering into an end-to-end framework. Therefore, our method not only scales to the large graphs but also learns the clustering-friendly representations.

2.2. Salable Graph Neural Network

Graph Neural Networks (GNNs) (Kipf & Welling, 2017; Veličković et al., 2018) become one of the most effective tools for learning over graph data. Many scalable GNNs have been proposed to scale to large graphs in recent years. For example, GraphSAGE (Hamilton et al., 2017) develops

a general inductive framework by sampling and aggregating features from the local neighborhood of the nodes. FastGCN (Chen et al., 2018) avoids the recursive neighborhood expansion by the layer-wise sampling to nodes in each layer independently. Additionally, SGC (Wu et al., 2019) decouples the transformation and propagation in GCN (Kipf & Welling, 2017). Besides, Graphsaint (Zeng et al., 2019) and Cluster-GCN (Chiang et al., 2019) are proposed to better maintain graph structure by sub-graph sampling. Moreover, (Li et al., 2019; 2020; 2021a; Liu et al., 2020) aim to design a sequence of works to make GCNs deeper. And various normalization and regularization techniques like DropEdge (Rong et al., 2019) and ParNorm (Zhao & Akoglu, 2019) are proposed to avoid over-fitting and over-smoothing. Furthermore, (Bojchevski et al., 2019; 2020; Rossi et al., 2020) attempt to propose more efficient propagation schemes. More recently, (Zhang et al., 2022) have designed a new efficient graph convolution via channel-wise scale transformation. (Zheng et al., 2022b) scale up the graph contrastive learning by simplifying DGI (Velickovic et al., 2019) and designing discriminate tasks. Sketch-GNN (Ding et al.) is proposed by training GNNs atop a few compact sketches of graph structure and node features. At the same time, (Wu et al., 2022; Rampášek et al., 2022) propose the scalable graph transformer model. However, the scalable GNNs for clustering tasks are few. It is challenging since the clustering task needs the model to estimate the whole sample distribution at once. Therefore, this paper aims to extend deep graph clustering methods to large-scale graphs.

3. Methodology

The methodology of *Dink-Net* is introduced in this section. We first define the problem and summary the basic notation. Then, the challenges of scaling deep graph clustering methods to large graphs are carefully illustrated. In addition, our solution to this problem is provided with the reasons.

3.1. Basic Notation

Given an attribute graph $\boldsymbol{G}$, $\boldsymbol{V} = \{v_1, v_2, ...v_N\}$ denotes a set of vertices, and $\boldsymbol{E} \subseteq \left\{(x, y)\middle|(x, y) \in \boldsymbol{V}^2\right\}$ denotes a set of edges between vertices, where each vertex attaches the corresponding D-dimension attributes. $\mathbf{X} \in \mathbb{R}^{N\times D}$ and $\mathbf{A} \in \mathbb{R}^{N\times N}$ are defined as the node attribute matrix and adjacency matrix, separately. Here, N and D denote the number of vertices and dimension number of the attributes, respectively. The basic notation table is presented in Table 1 of Appendix.

3.2. Problem Definition

For an attribute graph $\boldsymbol{G}$, the deep graph clustering algorithm aims to group the vertices into disjoint clusters. Specifically, the self-supervised neural network $\mathcal{F}$ embeds the nodes in $\boldsymbol{G}$ into the latent space as follows.

$$\mathbf{H} = \mathcal{F}(\boldsymbol{G}) = \mathcal{F}(\mathbf{X}, \mathbf{A}), \tag{1}$$

where $\mathbf{H} \in \mathbb{R}^{N\times d}$ denotes the node embeddings and d is the dimension number of latent features. Here, the self-supervised network $\mathcal{F}$ is trained with the pre-text tasks like reconstructive task, contrastive task, discriminative task, etc. In addition to encoding, the clustering method $\mathcal{C}$ is designed to group the nodes into different clusters as follows.

$$\hat{\mathbf{y}} = \mathcal{C}(\mathbf{H}, K), \tag{2}$$

where K is the number of clusters, which can be a hyper-parameter or a learnable parameter in the clustering method $\mathcal{C}$. The result $\hat{\mathbf{y}} \in \mathbb{R}^N$ is the clustering assignment vector.

Models suitable for large-scale graphs are always the goal researchers pursue. Unlike node classification and link prediction tasks, performing node clustering on a large-scale graph is more challenging. To this end, we aim to propose a method, which can empower the deep graph clustering algorithms to perform well on large-scale graphs, e.g., the graph with $\sim$111 million nodes and $\sim$10 billion edges. The detailed reasons are analyzed in the following sub-section.

3.3. Challenge Analyses

This section carefully analyzes the challenges of large-scale deep graph clustering. It begins with the differences between deep graph clustering and other tasks like node classification and link prediction. For the node classification task, instead of processing the whole graph data at once, algorithms can divide data into mini-batches (Li et al., 2014) and merely classify samples in each mini-batch. Similarly, models adopt the mini-batch technique for the link prediction task for the paired nodes and predict the probability of the links between paired nodes in mini-batches. The mini-batch technique works because the predictions of each node or link are relatively independent, and they will not influence each other at the inference stage. However, in the node clustering task, the methods need to group all nodes into disjoint clusters at once. In this process, the cluster assignment of each node will influence each other, and therefore the mini-batch technique easily fails.

Due to the above concerns, most existing deep graph clustering methods fail to use the mini-batch technique and process the whole data at once in the clustering process. Concretely, one class of methods first embeds nodes into the latent space and then directly performs the traditional clustering algorithm (Hartigan & Wong, 1979; Von Luxburg, 2007) on the learned node representations. We first analyze the complexity of traditional clustering methods. For example, the time complexity and space complexity of K-Means algorithm (Hartigan & Wong, 1979) is $\mathcal{O}(tNKD)$ and $\mathcal{O}(NK + ND + KD)$. Here, t, N, K, and D denote

the iteration times, node number, cluster number, and attribute dimension number, respectively. In addition, the time complexity and space complexity of spectral clustering algorithm (Von Luxburg, 2007) is $\mathcal{O}(N^3)$ and $\mathcal{O}(N^2)$.

Besides, the above methods separate the embedding and clustering optimization process, leading to sub-optimal performance. Differently, another class of methods (Wang et al., 2019; Bo et al., 2020; Liu et al., 2022b) unify the representation learning and clustering optimization into a joint framework by minimizing the KL divergence loss (Xie et al., 2016; Guo et al., 2017) as follows.

$$\min \mathcal{L}_{\mathrm{KL}} = \min \sum_i \sum_j \mathbf{P}_{ij} log \left(\frac{\mathbf{P}_{ij}}{\mathbf{Q}_{ij}} \right), \tag{3}$$

where $\mathbf{Q}_{ij}$ is the original clustering distribution and $\mathbf{P}_{ij}$ is the sharpened clustering distribution. This loss function optimizes the clustering distribution with the whole data. Thus, the calculation and optimization process is complex and resource-consuming, leading to $\mathcal{O}(NKd)$ time complexity and $\mathcal{O}(NK + Nd + Kd)$ space complexity.

Therefore, when the number of nodes N reaches a considerable value like $\sim$111 million on the ogbn-papers100M dataset, the previous two types of methods lead to unacceptable running time and out-of-memory problems. Besides, some methods (Hassani & Khasahmadi, 2020; Zhao et al., 2021; Liu et al., 2022b; Gong et al., 2022) need to process the $N \times N$ dense graph diffusion matrix (Klicpera et al., 2019), which also hinders the efficiency. To this end, we develop a scalable end-to-end deep graph clustering method with the guidance of divide-and-rule.

Algorithm 1 Dilation Shrink Network (*Dink-Net*)

Input: Attribute graph $\boldsymbol{G}$; cluster number K; epoch number T, T'; learning rate β,β'; batch size B; trade-off parameter α.
Output: Predicted cluster-ID $\hat{\mathbf{y}}$.

Initialize model parameters Θ in encoder $\mathcal{F}$ and projection $\mathcal{P}$;
\# Model pre-training stage
for epoch $= 1, 2, ..., T$ **do**
  Obtain new graph $\boldsymbol{G}' = \{\mathbf{X}', \mathbf{A}'\}$ via data augmentations;
  Node encoding: $\mathbf{H} = \mathcal{F}(\boldsymbol{G})$, $\mathbf{H}' = \mathcal{F}(\boldsymbol{G}')$;
  Representation projection: $\mathbf{Z} = \mathcal{P}(\mathbf{H})$, $\mathbf{H}' = \mathcal{P}(\mathbf{H}')$;
  Node summary: $\mathbf{g} = \mathbf{Z}.\mathrm{sum}(1)$, $\mathbf{g}' = \mathbf{Z}'.\mathrm{sum}(1)$;
  Calculate discrimination loss $\mathcal{L}_{\text{discri.}}$ in Eq. (4);
  Adam optimizer with learning rate β updates parameters Θ by minimizing $\mathcal{L}_{\text{discri.}}$;
end for
\# Model fine-tuning stage
Initialize the cluster center embeddings $\mathbf{C}$ in the K-means++ manner based on the learned node embeddings $\mathbf{H}$;
for epoch $= 1, 2, ..., T'$ **do**
  Generate batched graph data $\boldsymbol{B}$ with shuffle;
  for $\mathbf{G}_{\boldsymbol{B}}$ in $\boldsymbol{B}$ **do**
    Node encoding: $\mathbf{H} = \mathcal{F}(\mathbf{G}_{\boldsymbol{B}})$;
    Calculate discrimination loss $\mathcal{L}_{\text{discri.}}$ in Eq. (4);
    Calculate dilation loss $\mathcal{L}_{\text{dilation}}$ in Eq. (5);
    Calculate shrink loss $\mathcal{L}_{\text{shrink}}$ in Eq. (6);
    Calculate the total loss $\mathcal{L}$ in Eq. (7);
    Adam optimizer with learning rate β' updates parameters Θ and cluster centers $\mathbf{C}$ by minimizing $\mathcal{L}$;
  end for
end for
\# Model inference stage
Generate batched graph data $\boldsymbol{B}$ without shuffle;
for $\boldsymbol{G}_{\boldsymbol{B}}$ in $\boldsymbol{B}$ **do**
  Node encoding: $\mathbf{H} = \mathcal{F}(\boldsymbol{G}_{\boldsymbol{B}})$;
  Predict cluster-ID of batch data $\hat{\mathbf{y}}_{\boldsymbol{B}}$ by Eq. (8);
end for
Concatenate batched cluster-ID and obtain $\hat{\mathbf{y}}$;
Return $\hat{\mathbf{y}}$

3.4. Proposed Solution

Through careful analyses in the previous section, we conclude that it is challenging to expand existing deep graph clustering methods to large-scale graphs. To solve this problem, we propose a scalable end-to-end method named dilation shrink network (*Dink-Net*) with the idea of dilation and shrink of galaxies in the universe. Intuitively, our method optimizes the clustering distribution with the mini-batch data in an adversarial manner, therefore scaling to the large graph. *Dink-Net* mainly comprises the following node discriminate module and neural clustering module.

Node Discriminate Module. Given an attribute graph $\boldsymbol{G}$, we first apply the graph data augmentations τ, like attribute disturbance and edge dropout, on the node attributions $\mathbf{X} \in \mathbb{R}^{N \times D}$ and graph structure $\mathbf{A} \in \mathbb{R}^{N \times N}$. As the result, a augmented view graph view $\boldsymbol{G}'$ is constructed with $\mathbf{X}'$ and $\mathbf{A}'$. Subsequently, the parameter-share graph neural network encoder $\mathcal{F}(\cdot)$ embeds the nodes of $\boldsymbol{G}, \boldsymbol{G}'$ to the latent embeddings $\mathbf{H}, \mathbf{H}' \in \mathbb{R}^{N \times d}$. Then, a small parameter-shared neural network projection head $\mathcal{P}(\cdot)$ maps the nodes embeddings into a new latent space, where the self-supervised learning loss will be applied. It outputs $\mathbf{Z}, \mathbf{Z}' \in \mathbb{R}^{N \times d}$. After that, our method pools the new node embeddings into the node summaries $\mathbf{g}, \mathbf{g}' \in \mathbb{R}^{N \times 1}$ by the feature aggregation operation. To train the encoder $\mathcal{F}(\cdot)$ and projection head $\mathcal{P}(\cdot)$, a binary cross entropy loss function is minimized to tell apart the original node and the augmented node summaries below.

$$\begin{aligned} \min \mathcal{L}_{\text{discri.}} = \min \Bigg[& \frac{1}{N} \sum_{i=1}^{N} \left(1 \cdot log \frac{1}{\mathbf{g}_i} + 0 \cdot log \frac{1}{1 - \mathbf{g}_i} \right) + \\ & \frac{1}{N} \sum_{i=1}^{N} \left(0 \cdot log \frac{1}{\mathbf{g}'_i} + 1 \cdot log \frac{1}{1 - \mathbf{g}'_i} \right) \Bigg] = \\ & \min \frac{1}{N} \sum_{i=1}^{N} \left(log \frac{1}{\mathbf{g}_i} + log \frac{1}{1 - \mathbf{g}'} \right). \end{aligned} \tag{4}$$

where the first term aims to classify the original node summary embeddings to class 1 and the second term attempt to

classify the augmented node summary embeddings to another class 0. With this discriminative pre-text task, encoder $\mathcal{F}$ and projection head $\mathcal{P}$ are trained to extract discriminative features. Besides, this discriminate loss is compatible to batch training techniques on large graphs.

Neural Clustering Module. This module aims to guide our network to learn clustering-friendly representations. Concretely, based on the learned node representations $\mathbf{H}$, cluster center embeddings $\mathbf{C} \in \mathbb{R}^{K\times d}$ are initialized in the K-Means++ manner (Hartigan & Wong, 1979), where K denotes the cluster number. It is worth mentioning that the cluster center embeddings are assigned as the optimizable neural parameters with the gradients. Motivated by the dilation and shrink of galaxies in the universe, we design two loss functions to optimize the clustering distribution jointly.

Firstly, since the universe is expanding, the centers of different galaxies are pushed away from each other. Similarly, we attempt to push away different cluster centers by minimizing the proposed cluster dilation loss as follows.

$$\min \mathcal{L}_{\text{dilation}} = \min \frac{-1}{(K-1)K} \sum_{i=0}^{K-1} \sum_{j=0, j\neq i}^{K-1} \|\mathbf{C}_i - \mathbf{C}_j\|_2^2 , \tag{5}$$

where K denotes the cluster number. This cluster dilation loss will not bring high time or memory costs even when the sample number N is large. The idea of dilation loss comes from the universe expansion theory (Linder, 2003). The cluster centers are like stars with huge masses, and the samples are like planets around the cluster centers. The universe is expanding, and stars are moving apart from each other. Similarly, our cluster dilation loss pushes the cluster centers away from each other.

In addition to universe dilation, the galaxy's center will pull together the planets with gravity. From this observation, a cluster shrink loss is designed to optimize clustering distribution by pulling together samples to cluster centers. Considering the considerable sample number, our shrink loss is compatible with using the mini-batch samples. It is formulated as follows.

$$\min \mathcal{L}_{\text{shrink}} = \min \frac{1}{BK} \sum_{i=0}^{B-1} \sum_{j=0}^{K-1} \|\mathbf{H}_i - \mathbf{C}_j\|_2^2 , \tag{6}$$

where B denotes the batch size. Also, this objective will not bring large time or memory cost since it is linear to batch size B rather than sample number N. For this cluster shrink loss, if the clustering algorithm was perfect, we should force the samples close to the nearest cluster center since the perfect clustering algorithm can group the samples with the same ground truth into one cluster. However, for the practical clustering method, this operation easily leads to the confirmation bias problem (Nickerson, 1998). To alleviate this problem, a compromise cluster shrink loss is proposed in Eq. (6). to guide the samples to be close to all cluster centers. These two intuitive clustering losses optimize the clustering distribution with mini-batch data in shrink and dilation manners, thus endowing scalability and sample discriminative capability of *Dink-Net*.

The overall workflow of our proposed *Dink-Net* is demonstrated in Algorithm 1 and the PyTorch-style pseudo-code is given in Appendix.E. Detailed implement about *Dink-Net* can be found in Appendix.C. Next sub-section explores why our method works well on large-scale graphs.

3.5. Why *Dink-Net* Works Well on Large Graph?

By comparing with the existing methods, this section highlights the advantages of our proposed method from two aspects, including model training and model inference.

Model Training. As illustrated in Algorithm 1, the training process of *Dink-Net* contains the pre-training and fine-tuning stages. At the pre-train stage, the encoder $\mathcal{F}$ and the projection head $\mathcal{P}$ are optimized by minimizing the discriminate loss $\mathcal{L}_{\text{discri.}}$ in Eq. (4). For this loss function, the mini-batch technique can be applied since the discrimination process of each sample is independent. Therefore, given embeddings $\mathbf{Z}, \mathbf{Z}'$, the time complexity and space complexity of calculating $\mathcal{L}_{\text{discri.}}$ is $\mathcal{O}(Bd)$ and $\mathcal{O}(Bd)$, where B, d denote batch size and dimensions of latent features.

In the fine-tuning procedure, the total loss is formulated below.

$$\min \mathcal{L} = \min \left(\mathcal{L}_{\text{dilation}} + \mathcal{L}_{\text{shrink}} + \alpha \mathcal{L}_{\text{discri.}}\right), \tag{7}$$

where α is the trade-off hyper-parameter. We analyze the time and space complexity at this stage as follows. Firstly, given embeddings, the time and space complexity of calculating clustering dilation loss $\mathcal{L}_{\text{dilation}}$ in Eq. (5) is $\mathcal{O}(K^2 d)$ and $\mathcal{O}(Kd)$, where K denotes the cluster number. Since the $K \ll N$, $\mathcal{L}_{\text{dilation}}$ do not expend too much time and space resource even when the sample number grows to a large value. Secondly, the clustering shrink module pulls together the samples to the cluster centers. Considering the large sample space, it optimizes the clustering distribution with mini-batch data rather than operating on all samples. Therefore, in Eq. (6), it only brings $\mathcal{O}(BKd)$ time complexity and $\mathcal{O}(BK + Bd + Kd)$ space complexity when given the embeddings. Thirdly, calculating $\mathcal{L}_{\text{discri.}}$ takes $\mathcal{O}(Kd)$ time complexity and $\mathcal{O}(Kd)$ space complexity. To summarize, at the fine-tune stage, $\mathcal{L}_{\mathcal{C}}$ takes $\mathcal{O}(BKd + K^2 d + Kd) \rightarrow \mathcal{O}(BKd + K^2 d)$ time and $\mathcal{O}(BK + Bd + Kd)$ space costs given the embeddings. Referring to Section 3.3, it is obvious that our method's time and space costs are much less than that of the existing methods. We attribute this advantage to our proposed

cluster dilation and shrink loss functions since they allow our method to optimizing the clustering distribution with mini-batch samples even without performance drops. The experimental evidence can be found in Appendix D.2.

Model Inference. In the model inference process, with the well-learned cluster center embeddings $\mathbf{C} \in \mathbb{R}^{K\times d}$, the assignment of i-th sample can be calculated as follows.

$$\hat{\mathbf{y}}_i = \arg\min_j \|\mathbf{H}_i - \mathbf{C}_j\|_2, \tag{8}$$

where $\hat{\mathbf{y}} \in \mathbb{R}^N$ denotes the clustering assignment vector. Note that the inference of our method also can be carried out in a mini-batch manner. Therefore, when given embeddings, the time and space complexity of model inference is $\mathcal{O}(BKd)$ and $\mathcal{O}(BK + Bd + Kd)$, where B is batch size.

The above complexity analyses demonstrate time and space efficiency in theory. Compared with the existing state-of-the-art methods, the main advantages of our method are summarized as follows. 1) *Dink-Net* gets rid of from processing $N \times N$ graph diffusion matrix. 2) Our proposed loss functions allow *Dink-Net* to optimize the clustering distribution with mini-batch data even without performance drops. 3) *Dink-Net* unifies embedding learning and clustering optimization, resulting in clustering-friendly representations. Therefore, this sub-section illustrates that *Dink-Net* can scale well to large-scale graphs in theory. The next section aims to verify the superiority, effectiveness, scalability, and efficiency of *Dink-Net* by extensive experiments.

4. Experiment

In this section, we comprehensively evaluate our proposed *Dink-Net* by answering the main questions as follows.

- **Q1: Superiority.** Does *Dink-Net* outperforms the existing state-of-the-art deep graph clustering methods?
- **Q2: Effectiveness.** Are the proposed node discriminate and neural clustering modules effective?
- **Q3: Scalability.** Can the proposed method endow the deep graph clustering method scale to large graphs?
- **Q4: Efficiency.** How about the time and memory efficiency of the proposed method?
- **Q5: Sensitivity.** What is the performance of the proposed method with different hyper-parameters?
- **Q6: Convergence.** Will the proposed loss function, as well as the clustering performance, converge well?

The answers of **Q1**-**Q4** are illustrated in Section 4.2-4.5. In addition, sensitivity analyses (**Q5**) and convergence analyses (**Q6**) of *Dink-Net* can be found in Appendix.D.1 and D.2.

4.1. Experimental Setup

4.1.1. Environment

Experimental results are obtained from the server with four core Intel(R) Xeon(R) Platinum 8358 CPUs @ 2.60GHZ, one NVIDIA A100 GPU (40G), and the PyTorch platform.

4.1.2. Dataset

To evaluate the node clustering performance, we use seven attribute graph datasets, including Cora, CiteSeer, Amazon-Photo, ogbn-arxiv, Reddit, ogbn-products, ogbn-papers100M (Hu et al., 2020). The node numbers of graphs range from $\sim$3 kilo to $\sim$100 million, and the edge numbers of graphs range from $\sim$5 kilo to $\sim$1 billion. The statistical information is summarized in Table 2 of Appendix.

4.1.3. Evaluation Protocol

To evaluate the clustering methods, the predicted clustering assignment vector is firstly mapped to the ground truth by the Kuhn-Munkres algorithm (Plummer & Lovász, 1986). Then, the clustering performance is evaluated by four widely-used metrics (Liu et al., 2022b), including accuracy (ACC), normalized mutual information (NMI), adjusted rand index (ARI), and F1-score (F1). All results are obtained under three runs with different random seeds.

4.1.4. Compared Baseline

To demonstrate the superiority of the proposed method, we conduct comprehensive experiments to compare our *Dink-Net* with a variety of baseline methods. Concretely, the classical clustering method K-Means (Hartigan & Wong, 1979) uses the idea of exception maximum to separate samples. Additionally, the deep clustering methods (Xie et al., 2016; Guo et al., 2017; Yang et al., 2017; Li et al., 2021b) apply the deep neural networks to assist clustering. Moreover, the deep graph clustering methods (Grover & Leskovec, 2016; Velickovic et al., 2019; Cui et al., 2020; Bianchi et al., 2020; Hassani & Khasahmadi, 2020; Thakoor et al., 2021; Zhu et al., 2020; Gong et al., 2022; Liu et al., 2022b; Devvrit et al., 2022; Wang et al., 2017; 2019; Pan et al., 2019; Bo et al., 2020; Zhao et al., 2021; Tu et al., 2020; Lee et al., 2021) utilize graph neural networks to reveal graph structure and then group nodes into different clusters.

4.2. Superiority

This section answers the question **Q1**. To illustrate the superiority of the proposed method, extensive experiments are carried out to compare *Dink-Net* with the existing state-of-the-art methods. Four conclusions are demonstrated by carefully analyzing the results in Table 1.

1) The performance of the traditional method K-Means

Dataset	Metric	K-Means	DEC	DCN	node2vec	DGI	AGE	MinCutPool	MVGRL	BGRL	GRACE	ProGCL	AGC-DRR	DCRN	S^3GC	Ours
Cora	**ACC**	33.80	46.50	49.38	61.20	72.60	73.50	49.00	76.30	74.20	73.90	57.13	40.62	61.93	74.20	**78.10**
	NMI	14.98	23.54	25.65	44.40	57.10	57.58	41.00	60.80	58.40	57.00	41.02	18.74	45.13	58.80	**62.28**
	ARI	8.60	15.13	21.63	32.90	51.10	50.10	30.80	56.60	53.40	52.70	30.71	14.80	33.15	54.40	**61.61**
	F1	30.26	39.23	43.71	62.10	69.20	69.28	51.80	71.60	69.10	72.50	45.68	31.23	49.50	72.10	**72.66**
CiteSeer	**ACC**	39.32	55.89	57.08	42.10	68.60	69.73	53.70	62.83	67.50	63.10	65.92	68.32	69.86	68.80	**70.36**
	NMI	16.94	28.34	27.64	24.00	43.50	44.93	29.50	40.69	42.20	39.90	39.59	43.28	44.86	44.10	**45.87**
	ARI	13.43	28.12	29.31	11.60	44.50	45.31	26.20	34.18	42.80	37.70	36.16	45.34	45.64	44.80	**46.96**
	F1	36.08	52.62	53.80	40.10	64.30	64.45	51.60	59.54	63.10	60.30	57.89	64.82	64.83	64.30	**65.96**
Amazon-Photo	**ACC**	27.22	47.22	48.25	27.58	43.03	75.98	54.67	41.07	66.54	67.66	51.53	76.81	79.94	75.15	**81.71**
	NMI	13.23	37.35	38.76	11.53	33.67	65.38	50.02	30.28	60.11	53.46	39.56	66.54	73.70	59.78	**74.36**
	ARI	5.50	18.59	20.80	4.92	22.15	55.89	34.43	18.77	44.14	42.74	34.18	60.15	63.69	56.13	**68.40**
	F1	23.96	46.71	47.87	21.52	35.17	71.74	53.02	32.88	63.08	60.30	31.97	71.03	73.82	72.85	**73.92**
ogbn-arXiv	**ACC**	18.11	21.25	19.91	29.00	31.40	OOM	24.20	OOM	22.70	OOM	29.86	OOM	OOM	35.00	**43.68**
	NMI	22.13	25.14	23.81	40.60	41.20		38.00		32.10		37.51			**46.30**	43.73
	ARI	7.43	10.28	8.25	19.00	22.30		13.90		13.00		25.74			27.00	**35.22**
	F1	12.94	15.57	13.06	22.00	23.00		19.80		16.60		21.79			23.00	**26.92**
ogbn-products	**ACC**	18.11	23.79	24.50	35.70	32.00	OOM	25.70	OOM	OOM	OOM	35.21	OOM	OOM	40.20	**41.09**
	NMI	22.13	24.47	21.92	48.90	46.70		43.00				46.59			**53.60**	50.78
	ARI	7.43	9.05	10.96	17.00	17.40		13.00				19.87			**23.00**	21.08
	F1	12.94	13.54	13.95	24.70	19.20		18.00				21.55			25.00	**25.15**
Reddit	**ACC**	8.90	OOM	OOM	70.90	22.40	OOM	-	OOM	OOM	OOM	65.41	OOM	OOM	73.60	**76.03**
	NMI	11.40			79.20	30.60						70.48			**80.70**	78.91
	ARI	2.90			64.00	17.00						63.42			**74.50**	71.34
	F1	6.80			55.10	18.30						51.45			56.00	**67.95**
ogbn-papers100M	**ACC**	14.60	OOM	OOM	17.50	15.10	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	17.30	**26.67**
	NMI	37.33			38.00	41.60									45.30	**54.92**
	ARI	7.54			11.20	9.60									11.00	**18.01**
	F1	10.45			11.10	11.10									11.80	**19.48**

Table 1. Clustering performance (%) of our method and fourteen state-of-the-art baselines. The bold and underlined values are the best and the runner-up results. "OOM" indicates that the method raises the out-of-memory failure. "-" denotes that the methods do not converge.

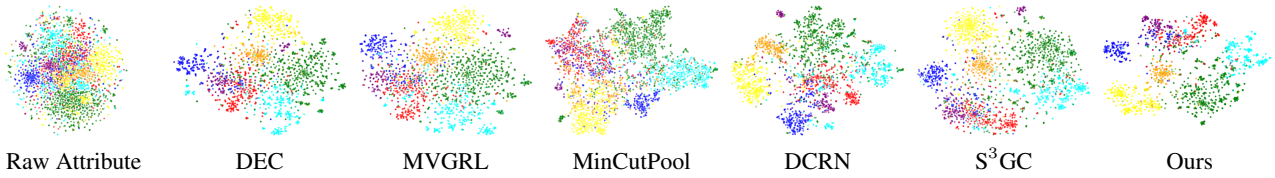


Figure 1. *t*-SNE visualization of seven methods on the Cora dataset.

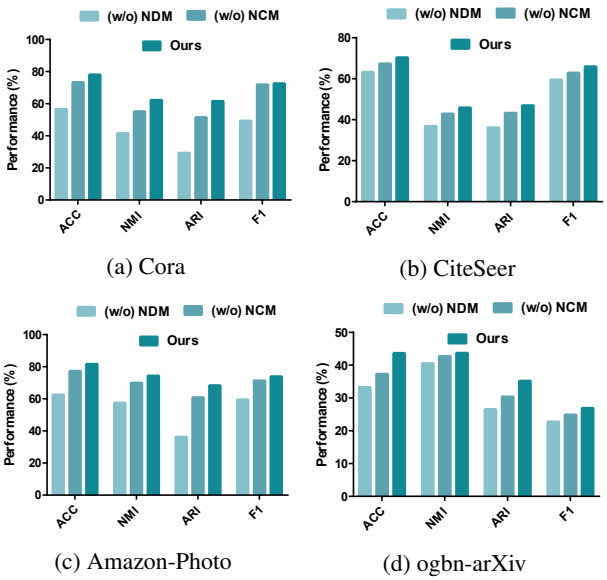
is limited. For example, on the Reddit dataset, K-Means merely achieves 8.90% ACC. The main reason is that it lacks representation learning, leading to indiscriminate samples.

2) The deep clustering methods DEC (Xie et al., 2016) and DCN (Yang et al., 2017) achieve an un-promising performance because they merely extract features from node attributes but ignore the structural information. For example, on Cora dataset, our method *Dink-Net* outperforms DEC by about 38.74% NMI.

3) For the deep graph clustering method, node2vec (Grover & Leskovec, 2016) merely takes care of the graph structure and overlooks the node attributes. Besides, the graph representation learning methods (Velickovic et al., 2019; Cui et al., 2020; Hassani & Khasahmadi, 2020; Thakoor et al., 2021; Zhu et al., 2020; Xia et al., 2022) can not optimize embedding and clustering in a unified framework. Therefore, these methods gain the sub-optimal clustering performance compared to our proposed method. For example, on the CiteSeer dataset, compared with BGRL, our method achieves about 2.86% ACC improvement.

4) Our proposed method outperforms the recent state-of-the-art deep graph clustering methods. For example, on the ogbn-papers100M dataset, *Dink-Net* achieves 9.62% NMI increment compared to the runner-up method S^3GC (Devvrit et al., 2022). The reason contains two aspects as follows. Firstly, the discriminate pre-text task in the representation learning process enhances the discriminative capability of samples. Secondly, the clustering modules unify the representation learning and the clustering process, guiding models to learn clustering-friendly representations.

Moreover, in order to intuitively demonstrate the superiority of our proposed *Dink-Net*, we visualize the learned node representations via the *t*-SNE algorithm (Van der Maaten & Hinton, 2008). As shown in Figure 1, we find that our



(a) Cora (b) CiteSeer

(c) Amazon-Photo (d) ogbn-arXiv

Figure 2. Ablation studies of the proposed modules on four datasets. "(w/o) NDM" denotes *Dink-Net* without the node discriminate module. "(w/o) NDM" denotes *Dink-Net* without the neural clustering module. "Ours" denotes our proposed *Dink-Net*.

proposed method better reveals cluster structure in the latent space. Due to the page limitation, the additional experimental results and analyses are presented in Appendix D.4.

4.3. Effectiveness

The question **Q2** is answered in this section. To verify the effectiveness of the proposed modules, we carefully conduct ablation studies on four datasets. Specifically, as shown in Figure 2, our method is denoted as "Ours". Additionally, our method without the node discriminate module and without the neural clustering module is denoted as "(w/o) NDM" and "(w/o) NCM", respectively. From the experimental results in Figure 2, three observations are presented as follows.

1) "(w/o) NDM" can not achieve expected clustering performance since it lacks the strong representation learning capability of the node discriminate module.

2) The results indicate that "(w/o) NCM" becomes the runner-up. Although the node discriminate module endows strong representation capability, it can not learn clustering-friendly features since the process of embedding, and clustering is detached.

3) Our *Dink-Net* achieves the best clustering performance because it unifies representation learning and clustering optimization to extract clustering-friendly features.

The above results and observations prove the effectiveness of the node discriminate and neural clustering module in *Dink-Net*. Both of them can boost performance.

4.4. Scalability

This section attempts to answer the question **Q3**. To verify the scalability of the proposed method, we conduct experiments on seven graph datasets with different scales. For instance, the Cora dataset contains 2708 nodes, and ogbn-papers100M contains $\sim$111 million nodes. The statistics of these datasets can be found in Table 2 in Appendix. Table 1 provides the clustering performance on these datasets. From these results, we have four observations as follows.

1) The traditional clustering method K-Means can complete the clustering process on seven datasets. However, there are two drawbacks as follows. Firstly, it fails to achieve promising performance since it lacks the representation learning process. For example, on the Reddit dataset our method outperforms K-Means 67.13% from the aspect of ACC. Secondly, it takes a long running time on large-scale graph datasets, e.g., $\sim$5 days for papers100M dataset on CPU.

2) The deep clustering methods raise the out-of-memory failure on the Reddit and ogbn-papers100M datasets. The main reason for enormous memory costs is that the KL divergence loss (Xie et al., 2016; Guo et al., 2017) estimates and sharpens the cluster distribution with all samples, leading to enormous memory costs. Also, they neglect the graph structure leading to worse performance.

3) The most of deep graph clustering methods easily lead to the out-of-memory problem because some methods (Hassani & Khasahmadi, 2020; Liu et al., 2022b; Gong et al., 2022) need to process $N \times N$ dense graph diffusion matrix, which is inefficient on time and memory. For the scalable methods like node2vec (Grover & Leskovec, 2016), DGI (Velickovic et al., 2019), and S^3GC (Devvrit et al., 2022), they separate the embedding and clustering, leading to sub-optimal performance.

4) Our proposed *Dink-Net* scales well to all seven datasets and achieves promising performances. The reasons and analyses are demonstrated in Section 3.5.

Through the above observations, we conclude that the existing methods easily lead to the out-of-memory problem or the long ruining time problem. But our method can endow the models with excellent scalability to large-scale graphs.

4.5. Efficiency

This section attempts to answer the question **Q4**. To verify the efficiency of *Dink-Net*, we test the time and memory costs of various methods on the Cora and ogbn-papers100M datasets. The main observations (See Table 1 and Table 2) and analyses are illustrated.

1) For the time costs, on ogbn-papers100M dataset, the scalable baselines node2vec(Grover & Leskovec, 2016), DGI(Velickovic et al., 2019), and S^3GC(Devvrit et al., 2022)

Method	Time Complexity (per iteration)	Space Complexity	Time Cost (s)	Memory Cost (MB)
DGI	$\mathcal{O}(ED + Nd^2)$	$\mathcal{O}(E + Nd + d^2)$	19.03	3798
MVGRL	$\mathcal{O}(N^2d + Nd^2)$	$\mathcal{O}(N^2 + Nd + d^2)$	168.20	9466
S^{3}GC	$\mathcal{O}(NSd^2)$	$\mathcal{O}(Nd + BSd + d^2)$	508.21	1474
Dink-Net (Ours)	$\mathcal{O}(BKd + K^2d)$	$\mathcal{O}(BK + Bd + Kd)$	35.09	1248

Table 2. Time and space analyses of various methods. The experimental GPU memory costs and time costs are obtained on Cora dataset.

run in ∼24 hours. Differently, our proposed method takes ∼9 hours for model pre-training and ∼3 hours for model fine-tuning. From the theoretical view, at the pre-training stage, the time complexity of calculating discriminative loss is $\mathcal{O}(Bd)$, which is linear to batch size. Moreover, at fine-tuning state, calculating the clustering loss takes $\mathcal{O}(BKd + K^2d + Kd)$ time complexity, which is also linear to batch size.

2) For the memory costs on ogbn-papers100M dataset, most baseline methods raise the out-of-memory problem on 40GB GPU. But our method takes ∼20GB GPU memory during training. In addition, from a theoretical perspective, calculating discriminative loss and clustering loss take $\mathcal{O}(Bd)$ and $\mathcal{O}(BK + Bd + Kd)$ space complexity, which are both linear to node number in a mini-batch.

These results and analyses demonstrate the efficiency of our proposed method in both the time and memory aspects. Due to page limitation, the additional experiments and complexity analyses can be found in Appendix.B.

5. Conclusion

This work aims to scale deep graph clustering to large graphs. It begins with analyzing the drawbacks of existing methods. Firstly, part of the method must process a space-consuming graph diffusion matrix. Secondly, some algorithms must optimize clustering distribution with all nodes, easily resulting in the out-of-memory problem. Thirdly, the scalable S^{3}GC achieves sub-optimal performance since it separates representation and clustering. To solve this problem, a novel scalable deep graph clustering termed *Dink-Net* is proposed under the guidance idea of dilation and shrink. Our method contains the node discriminate and neural clustering module. With these designs, we unify representation learning and clustering optimization into an end-to-end framework, guiding network to learn clustering-friendly features. Also, cluster dilation and shrink loss functions allow our method to optimize clustering distribution with mini-batch data. Extensive experiments and theoretical analyses verify the superiority. In the future, it is worth to extending *Dink-Net* to heterogeneous graphs (Zheng et al., 2021), heterophily graphs(Liu et al., 2022d), knowledge graphs (Liang et al., 2022a), and molecular graphs (Xia et al., 2023).

6. Acknowledgments

We thank all anonymous reviewers and program chairs for their constructive and helpful reviews. This work was supported by the National Key R&D Program of China (project no. 2020AAA0107100) and the National Natural Science Foundation of China (project no. 62276271). Besides, this work was also supported by the National Key R&D Program of China (Project 2022ZD0115100), the National Natural Science Foundation of China (Project U21A20427), the Research Center for Industries of the Future (Project WU2022C043), and the Competitive Research Fund (Project WU2022A009) from the Westlake Center for Synthetic Biology and Integrated Bioengineering.

References

Bianchi, F. M., Grattarola, D., and Alippi, C. Spectral clustering with graph neural networks for graph pooling. In *Proc. of ICML*, 2020.

Bo, D., Wang, X., Shi, C., Zhu, M., Lu, E., and Cui, P. Structural deep clustering network. In *Proc. of WWW*, 2020.

Bojchevski, A., Klicpera, J., Perozzi, B., Blais, M., Kapoor, A., Lukasik, M., and Günnemann, S. Is pagerank all you need for scalable graph neural networks? In *ACM KDD, MLG Workshop*, 2019.

Bojchevski, A., Klicpera, J., Perozzi, B., Kapoor, A., Blais, M., Rózemberczki, B., Lukasik, M., and Günnemann, S. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2464–2473, 2020.

Cao, S., Lu, W., and Xu, Q. Deep neural networks for learning graph representations. In *Proc. of AAAI*, 2016.

Chen, J., Ma, T., and Xiao, C. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings*

of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 257–266, 2019.

Cui, G., Zhou, J., Yang, C., and Liu, Z. Adaptive graph encoder for attributed graph embedding. In *Proc. of KDD*, 2020.

Devvrit, F., Sinha, A., Dhillon, I., and Jain, P. S3gc: Scalable self-supervised graph clustering. 2022.

Ding, M., Rabbani, T., An, B., Wang, E. Z., and Huang, F. Sketch-gnn: Scalable graph neural networks with sublinear training complexity. In *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*.

Gong, L., Zhou, S., Liu, X., and Tu, W. Attributed graph clustering with dual redundancy reduction. In *Proc. of IJCAI*, 2022.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Guo, X., Gao, L., Liu, X., and Yin, J. Improved deep embedded clustering with local structure preservation. In *Proc. of IJCAI*, 2017.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Hartigan, J. A. and Wong, M. A. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 1979.

Hassani, K. and Khasahmadi, A. H. Contrastive multi-view representation learning on graphs. In *Proc. of ICML*, 2020.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *Proc. of ICLR*, 2017.

Klicpera, J., Weißenberger, S., and Günnemann, S. Diffusion improves graph learning. *arXiv preprint arXiv:1911.05485*, 2019.

Lee, N., Lee, J., and Park, C. Augmentation-free self-supervised learning on graphs. *arXiv preprint arXiv:2112.02472*, 2021.

Li, G., Muller, M., Thabet, A., and Ghanem, B. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9267–9276, 2019.

Li, G., Xiong, C., Thabet, A., and Ghanem, B. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

Li, G., Müller, M., Ghanem, B., and Koltun, V. Training graph neural networks with 1000 layers. In *International conference on machine learning*, pp. 6437–6449. PMLR, 2021a.

Li, M., Zhang, T., Chen, Y., and Smola, A. J. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 661–670, 2014.

Li, X., Zhang, H., and Zhang, R. Adaptive graph auto-encoder for general data clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021b.

Liang, K., Liu, Y., Zhou, S., Liu, X., and Tu, W. Relational symmetry based knowledge graph contrastive learning. *arXiv preprint arXiv:2211.10738*, 2022a.

Liang, K., Meng, L., Liu, M., Liu, Y., Tu, W., Wang, S., Zhou, S., Liu, X., and Sun, F. Reasoning over different types of knowledge graphs: Static, temporal and multi-modal. *arXiv preprint arXiv:2212.05767*, 2022b.

Liang, K., Meng, L., Zhou, S., Wang, S., Tu, W., Liu, Y., Liu, M., and Liu, X. Message intercommunication for inductive relation reasoning. *arXiv preprint arXiv:2305.14074*, 2023a.

Liang, K., Tan, J., Zeng, D., Huang, Y., Huang, X., and Tan, G. Abslearn: a gnn-based framework for aliasing and buffer-size information retrieval. *Pattern Analysis and Applications*, pp. 1–19, 2023b.

Linder, E. V. Exploring the expansion history of the universe. *Physical review letters*, 90(9):091301, 2003.

Liu, M., Gao, H., and Ji, S. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 338–348, 2020.

Liu, M., Liang, K., Xiao, B., Zhou, S., Tu, W., Liu, Y., Yang, X., and Liu, X. Self-supervised temporal graph learning with temporal and structural intensity alignment. *arXiv preprint arXiv:2302.07491*, 2023a.

Liu, M., Liu, Y., Liang, K., Wang, S., Zhou, S., and Liu, X. Deep temporal graph clustering. *arXiv preprint arXiv:2305.10738*, 2023b.

Liu, Y., Jin, M., Pan, S., Zhou, C., Zheng, Y., Xia, F., and Philip, S. Y. Graph self-supervised learning: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 35(6):5879–5900, 2022a.

Liu, Y., Tu, W., Zhou, S., Liu, X., Song, L., Yang, X., and Zhu, E. Deep graph clustering via dual correlation reduction. In *Proc. of AAAI*, 2022b.

Liu, Y., Xia, J., Zhou, S., Wang, S., Guo, X., Yang, X., Liang, K., Tu, W., Li, Z. S., and Liu, X. A survey of deep graph clustering: Taxonomy, challenge, and application. *arXiv preprint arXiv:2211.12875*, 2022c.

Liu, Y., Zheng, Y., Zhang, D., Lee, V., and Pan, S. Beyond smoothing: Unsupervised graph representation learning with edge heterophily discriminating. *arXiv preprint arXiv:2211.14065*, 2022d.

Liu, Y., Zhou, S., Liu, X., Tu, W., and Yang, X. Improved dual correlation reduction network. *arXiv preprint arXiv:2202.12533*, 2022e.

Liu, Y., Yang, X., Zhou, S., and Liu, X. Simple contrastive graph clustering. *IEEE Transactions on Neural Networks and Learning Systems*, 2023c.

Liu, Y., Yang, X., Zhou, S., Liu, X., Wang, Z., Liang, K., Tu, W., Li, L., Duan, J., and Chen, C. Hard sample aware network for contrastive deep graph clustering. In *Proc. of AAAI*, 2023d.

Meng, L., Liang, K., Xiao, B., Zhou, S., Liu, Y., Liu, M., Yang, X., and Liu, X. Sarf: Aliasing relation assisted self-supervised learning for few-shot relation reasoning. *arXiv preprint arXiv:2304.10297*, 2023.

Mo, Y., Peng, L., Xu, J., Shi, X., and Zhu, X. Simple unsupervised graph representation learning. In *AAAI*, pp. 7797–7805, 2022.

Mo, Y., Chen, Y., Lei, Y., Peng, L., Shi, X., Yuan, C., and Zhu, X. Multiplex graph representation learning via dual correlation reduction. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

Nickerson, R. S. Confirmation bias: A ubiquitous phenomenon in many guises. *Review of general psychology*, 2(2):175–220, 1998.

Pan, S., Hu, R., Long, G., Jiang, J., Yao, L., and Zhang, C. Adversarially regularized graph autoencoder for graph embedding. In *Proc. of IJCAI*, 2018.

Pan, S., Hu, R., Fung, S.-f., Long, G., Jiang, J., and Zhang, C. Learning graph embedding with adversarial training methods. *IEEE transactions on cybernetics*, 2019.

Peng, Z., Liu, H., Jia, Y., and Hou, J. Attention-driven graph clustering network. In *Proc. of ACM MM*, 2021.

Plummer, M. D. and Lovász, L. *Matching theory*. Elsevier, 1986.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. Recipe for a general, powerful, scalable graph transformer. *arXiv preprint arXiv:2205.12454*, 2022.

Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

Rossi, E., Frasca, F., Chamberlain, B., Eynard, D., Bronstein, M., and Monti, F. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 7:15, 2020.

Thakoor, S., Tallec, C., Azar, M. G., Munos, R., Veličković, P., and Valko, M. Bootstrapped representation learning on graphs. In *ICLR 2021 Workshop on Geometrical and Topological Representation Learning*, 2021.

Tian, F., Gao, B., Cui, Q., Chen, E., and Liu, T.-Y. Learning deep representations for graph clustering. In *Proc. of AAAI*, 2014.

Tu, W., Zhou, S., Liu, X., Guo, X., Cai, Z., Cheng, J., et al. Deep fusion clustering network. *arXiv preprint arXiv:2012.09600*, 2020.

Van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research*, 2008.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *Proc. of ICLR*, 2018.

Velickovic, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., and Hjelm, R. D. Deep graph infomax. *ICLR (Poster)*, 2019.

Von Luxburg, U. A tutorial on spectral clustering. *Statistics and computing*, 2007.

Wang, C., Pan, S., Long, G., Zhu, X., and Jiang, J. Mgae: Marginalized graph autoencoder for graph clustering. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017.

Wang, C., Pan, S., Hu, R., Long, G., Jiang, J., and Zhang, C. Attributed graph clustering: A deep attentional embedding approach. *arXiv preprint arXiv:1906.06532*, 2019.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

Wu, Q., Zhao, W., Li, Z., Wipf, D., and Yan, J. Nodeformer: A scalable graph structure learning transformer for node classification. In *Advances in Neural Information Processing Systems*, 2022.

Xia, J., Wu, L., Wang, G., Chen, J., and Li, S. Z. Progcl: Rethinking hard negative mining in graph contrastive learning. In *Proc. of ICML*, 2022.

Xia, J., Zhao, C., Hu, B., Gao, Z., Tan, C., Liu, Y., Li, S., and Li, S. Z. Mole-bert: Rethinking pre-training graph neural networks for molecules. In *The Eleventh International Conference on Learning Representations*, 2023.

Xie, J., Girshick, R., and Farhadi, A. Unsupervised deep embedding for clustering analysis. In *Proc. of ICML*, 2016.

Yang, B., Fu, X., Sidiropoulos, N. D., and Hong, M. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *Proc. of ICML*, 2017.

Yang, X., Liu, Y., Zhou, S., Liu, X., and Zhu, E. Interpolation-based correlation reduction network for semi-supervised graph learning. *arXiv preprint arXiv:2206.02796*, 2022a.

Yang, X., Liu, Y., Zhou, S., Wang, S., Liu, X., and Zhu, E. Contrastive deep graph clustering with learnable augmentation. *arXiv preprint arXiv:2212.03559*, 2022b.

Yang, X., Liu, Y., Zhou, S., Wang, S., Tu, W., Zheng, Q., Liu, X., Fang, L., and Zhu, E. Cluster-guided contrastive graph clustering network. In *Proc. of AAAI*, 2023.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

Zhang, T., Wu, Q., Yan, J., Zhao, Y., and Han, B. Scalegcn: Efficient and effective graph convolution via channel-wise scale transformation. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Zhao, H., Yang, X., Wang, Z., Yang, E., and Deng, C. Graph debiased contrastive learning with joint representation clustering. In *Proc. of IJCAI*, 2021.

Zhao, L. and Akoglu, L. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

Zheng, Y., Lee, V. C., Wu, Z., and Pan, S. Heterogeneous graph attention network for small and medium-sized enterprises bankruptcy prediction. In *Advances in Knowledge Discovery and Data Mining: 25th Pacific-Asia Conference, PAKDD 2021, Virtual Event, May 11–14, 2021, Proceedings, Part I*, pp. 140–151. Springer, 2021.

Zheng, Y., Jin, M., Pan, S., Li, Y.-F., Peng, H., Li, M., and Li, Z. Toward graph self-supervised learning with contrastive adjusted zooming. *IEEE Transactions on Neural Networks and Learning Systems*, 2022a.

Zheng, Y., Pan, S., Lee, V. C., Zheng, Y., and Yu, P. S. Rethinking and scaling up graph contrastive learning: An extremely efficient approach with group discrimination. *arXiv preprint arXiv:2206.01535*, 2022b.

Zheng, Y., Zheng, Y., Zhou, X., Gong, C., Lee, V. C., and Pan, S. Unifying graph contrastive learning with flexible contextual scopes. In *2022 IEEE International Conference on Data Mining (ICDM)*, pp. 793–802. IEEE, 2022c.

Zhu, Y., Xu, Y., Yu, F., Liu, Q., Wu, S., and Wang, L. Deep Graph Contrastive Representation Learning. In *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

Appendix of "Dink-Net: Neural Clustering on Large Graph"

A. Notations & Datasets

The basic notations are summarized in Table 1. Table 2 lists the statistical information about seven datasets. These datasets have various scales. For example, CiteSeer has about 3.3K nodes and 4.6K edges, while ogbn-papers100M has about 111M nodes and 1.6B edges. In addition, the network densities of these graphs are various. Concretely, the density of Cora is 0.07% while the density of Amazon-Photo is 0.25%.

Notation	Meaning
$\boldsymbol{G}$	Attribute Graph
N	Sample Number
D	Attribute Dimension Number
d	Latent Feature Dimension Number
$\mathcal{F}$	Encoding Network
$\mathcal{C}$	Clustering Method
$\mathbf{X} \in \mathbb{R}^{N \times D}$	Attribute Matrix
$\mathbf{A} \in \mathbb{R}^{N \times N}$	Adjacency Matrix
$\mathbf{H} \in \mathbb{R}^{N \times d}$	Node Embedding Matrix
$\mathbf{g} \in \mathbb{R}^{N \times 1}$	Node Summary Vector
$\mathbf{C} \in \mathbb{R}^{K \times d}$	Cluster Center Embedding Matrix
$\hat{\mathbf{y}} \in \mathbb{R}^{N \times 1}$	Clustering Assignment Vector
$\mathbf{y} \in \mathbb{R}^{N \times 1}$	Sample Label Vector

Table 1. Basic Notations

Dataset	Type	# Nodes	# Edges	# Feature Dims	# Classes
Cora	Attributed Graph	2,708	5,278	1,433	7
CiteSeer	Attributed Graph	3,327	4,614	3,703	6
Amazon-Photo	Attributed Graph	7,650	119,081	745	8
ogbn-arxiv	Attributed Graph	169,343	1,166,243	128	40
Reddit	Attributed Graph	232,965	23,213,838	602	41
ogbn-products	Attributed Graph	2,449,029	61,859,140	100	47
ogbn-papers100M	Attributed Graph	111,059,956	1,615,685,872	128	172

Table 2. The statistical information of seven datasets.

B. Time and Space Analyses

In this section, we analyze and summarize the time and space complexity of the various baseline methods, including spectral clustering (Von Luxburg, 2007), K-Means (Hartigan & Wong, 1979), DEC (Xie et al., 2016), node2vec (Grover & Leskovec, 2016), DGI (Velickovic et al., 2019), MVGRL (Hassani & Khasahmadi, 2020), GRACE (Zhu et al., 2020), BGRL (Thakoor et al., 2021), S^3GC (Devvrit et al., 2022), and our proposed *Dink-Net* in Table 3. Here, N denotes the node number in the graph, B denotes the batch size, K denotes the cluster number, E denotes the edges number of the graph, S denotes the average degree of the graph, D denotes the dimensions of node attributes, and d denotes the dimensions of latent features. From these analyses, we find that the complexity of most methods will become unacceptable when the sample number reaches a tremendous value. Different, our method's time and memory complexity is linear to the batch size, alleviating the out-of-memory and long-running time problems. In addition, we also test the memory and time costs of these methods via

experiments on the Cora dataset. From these experimental results, we find that our proposed *Dink-Net* also achieves efficient results even without batch-training techniques on a small dataset. More importantly, *Dink-Net* is compatible with the mini-batch training technique even without performance drops. Therefore it scales well with the large graphs. Experimental evidence can be found in Figure 2.

Method	Time Complexity (per iteration)	Space Complexity	GPU Memory Cost (MB)	Time Cost (s)
Spectral Clustering	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2)$	-	29.31
K-Means	$\mathcal{O}(NKD)$	$\mathcal{O}(NK+ND+KD)$	-	6.01
DEC	$\mathcal{O}(NKd)$	$\mathcal{O}(NK+Nd+Kd)$	1294	14.59
node2vec	$\mathcal{O}(Bd)$	$\mathcal{O}(Nd)$	-	111.03
DGI	$\mathcal{O}(ED+Nd^2)$	$\mathcal{O}(E+Nd+d^2)$	3798	19.03
MVGRL	$\mathcal{O}(N^2d+Nd^2)$	$\mathcal{O}(N^2+Nd+d^2)$	9466	168.20
GRACE	$\mathcal{O}(N^2d+Ed+d^2)$	$\mathcal{O}(E+Nd)$	1292	44.77
BGRL	$\mathcal{O}(Ed+Nd^2)$	$\mathcal{O}(E+Nd+d^2)$	1258	44.18
S^{3}GC	$\mathcal{O}(NSd^2)$	$\mathcal{O}(Nd+BSd+d^2)$	1474	508.21
Dink-Net	$\mathcal{O}(BKd+K^2d)$	$\mathcal{O}(BK+Bd+Kd)$	1248	35.09

Table 3. Time and space analyses of various methods. The experimental costs are obtained on the Cora dataset. "-" means ruining on CPU.

C. Design Details & Hyper-parameter Settings

In this section, we introduce the design details of our proposed method and summarize the hyper-parameter settings. Following the existing works (Zheng et al., 2022; Velickovic et al., 2019), for the encoder $\mathcal{F}$, we adopt the graph convolutional network (GCN) (Kipf & Welling, 2017). Besides, we use multilayer perceptron (MLP) (Pal & Mitra, 1992) as the projector in *Dink-Net*. Next, we report the hyper-parameter settings of our method in Table 4. Here, T is the epoch number of pre-training, T' is the epoch number of fine-tuning, β is the learning rate of pre-training, β' is the learning rate of fine-tuning, α is the trade-off parameter, B is the batch size, and d is the dimension number of latent features.

	T	β	T'	β'	α	B	d
Cora	200	1e-3	200	1e-2	1e-10	-	512
CiteSeer	100	5e-4	200	1e-2	1e-10	-	1536
Amazon-Photo	2000	5e-4	100	1e-2	1e-10	-	512
ogbn-arXiv	1	1e-4	100	1e-4	1e-10	8192	1500
ogbn-products	10	1e-3	10	1e-2	1e-10	8192	1024
Reddit	10	1e-4	1	1e-5	1e-10	10240	512
ogbn-papers100M	1	1e-4	1	1e-5	1e-10	10240	256

Table 4. Hyper-parameter settings of our proposed method. "-" denotes that it does not use mini-batch training.

D. Additional Experimental Result

D.1. Compare Experiment

Due to the limited regular paper pages, the additional compare experimental results are demonstrated in Table 5. We further compare our proposed *Dink-Net* with the nine baselines, including IDEC (Guo et al., 2017), AdaGAE(Li et al., 2021), MGAE (Wang et al., 2017), DAEGC (Wang et al., 2019), ARGA (Pan et al., 2019), DMoN (Tsitsulin et al., 2020), SDCN (Bo et al., 2020), GDCL (Zhao et al., 2021), and DFCN (Tu et al., 2020). These additional experimental results further verify the superiority and scalability of our proposed *Dink-Net*.

D.2. Sensitivity Analyses

This section aims to answer **Q**5: Is the performance of the proposed method sensitive to hyper-parameters? This section analyzes the sensitivity of our proposed *Dink-Net*. Firstly, we analyze the trade-off parameter α on Cora and CiteSeer datasets. As shown in Figure 2 (a) and Figure 2(b), we find that our *Dink-Net* can achieve good performance with different

Dataset	Metric	IDEC	AdaGAE	MGAE	DAEGC	ARGA	DMoN	SDCN	GDCL	DFCN	Ours
Cora	**ACC**	51.61	50.06	43.38	70.43	71.04	51.70	35.60	70.83	36.33	**78.10**
	NMI	26.31	32.19	28.78	52.89	51.06	47.30	14.28	56.60	19.36	**62.28**
	ARI	22.07	28.25	16.43	49.63	47.71	30.10	7.78	48.05	4.67	**61.61**
	F1	47.17	53.53	33.48	68.27	69.27	57.40	24.37	52.88	26.16	**72.66**
CiteSeer	**ACC**	60.49	54.01	61.35	64.54	61.07	38.50	65.96	66.39	69.50	**70.36**
	NMI	27.17	27.79	34.63	36.41	34.40	30.30	38.71	39.52	43.90	**45.87**
	ARI	25.70	24.19	33.55	37.78	34.32	20.00	40.17	41.07	45.50	**46.96**
	F1	61.62	51.11	57.36	62.20	58.23	43.70	63.62	61.12	64.30	**65.96**
Amazon-Photo	**ACC**	47.62	67.70	71.57	75.96	69.28	24.77	53.44	43.75	76.82	**81.71**
	NMI	37.83	55.96	62.13	65.25	58.36	7.69	44.85	37.32	66.23	**74.36**
	ARI	19.24	46.20	48.82	58.12	44.18	3.81	31.21	21.57	58.28	**68.40**
	F1	47.20	62.95	68.08	69.87	64.30	17.98	50.66	38.37	71.25	**73.51**
ogbn-arXiv	**ACC**	22.67	OOM	OOM	OOM	OOM	25.00	OOM	OOM	OOM	**43.68**
	NMI	27.54					35.60				**43.73**
	ARI	12.15					12.70				**35.22**
	F1	17.58					19.00				**26.92**
ogbn-products	**ACC**	20.53	OOM	OOM	OOM	OOM	30.40	OOM	OOM	OOM	**41.09**
	NMI	22.15					42.80				**50.78**
	ARI	9.87					13.90				**21.08**
	F1	12.48					21.00				**25.15**
Reddit	**ACC**	OOM	OOM	OOM	OOM	OOM	52.90	OOM	OOM	OOM	**76.03**
	NMI						62.80				**78.91**
	ARI						50.20				**71.34**
	F1						26.00				**67.95**
ogbn-papers100M	**ACC**	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	**26.67**
	NMI										**54.92**
	ARI										**18.01**
	F1										**19.48**

Table 5. Clustering performance (%) of our method and nine state-of-the-art baselines. The bold values are the best results. "OOM" indicates that the method raise the out-of-memory failure.

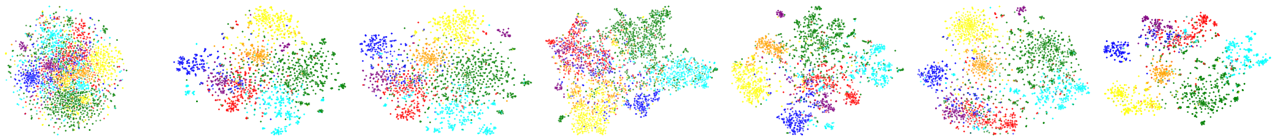
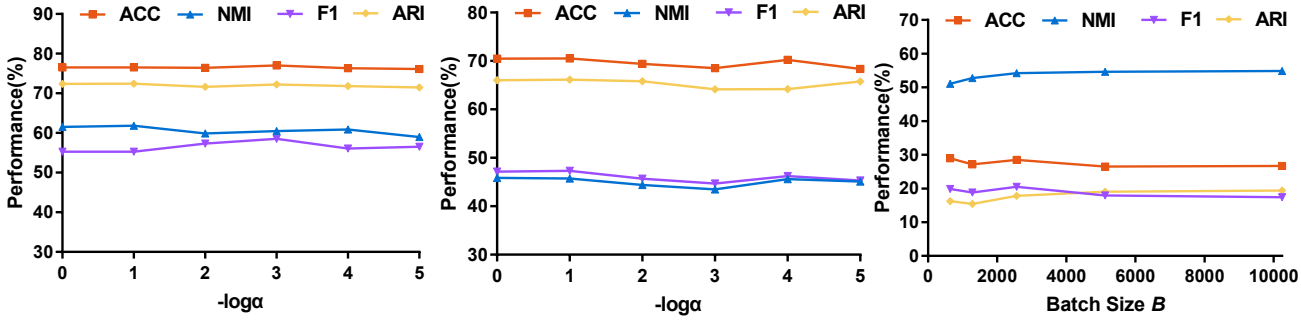


Figure 1. t-SNE visualization of seven methods on Cora dataset.

values of α. Therefore it is not sensitive to α. Secondly, we analyze another important hyper-parameter batch size B on the ogbn-papers100M dataset. As shown in Figure 2 (c), we find that the performance of *Dink-Net* is not sensitive to batch size B, therefore our proposed loss functions allow our method to optimizing clustering distribution with mini-batch data even without performance drops. Besides, training model with a larger batch size will bring some extent of ACC improvement.

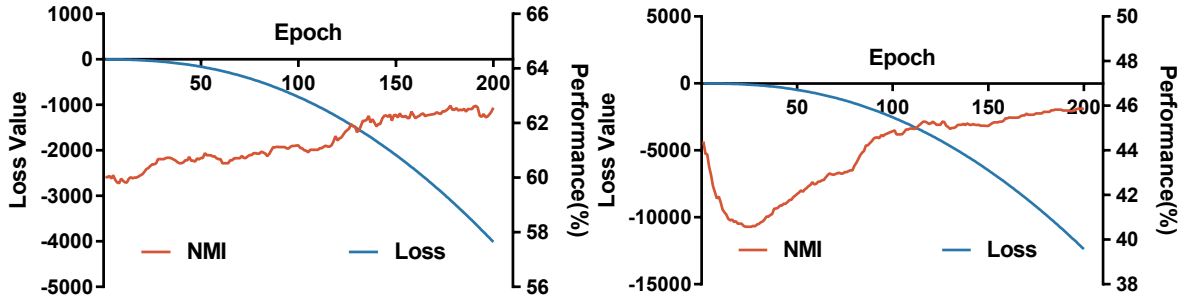
D.3. Convergence Analyses

This section aims to answer **Q**6: Can the proposed loss functions and the clustering performance converge well? To verify the convergence of our proposed *Dink-Net*, we conduct experiments on two datasets, including Cora and CiteSeer. Specifically, as shown in Figure 3, the NMI and loss values are recorded per epoch in the training process. From these experimental results, we observe that the loss values gradually decrease and tend to converge. Meanwhile, the clustering performance NMI increases. Therefore, our proposed method converges well.



(a) trade-off parameter α on Cora (b) trade-off parameter α on CiteSeer (c) batch size on ogbn-papers100M

Figure 2. Sensitivity analyses of hyper-parameters.



(a) Cora (b) CiteSeer

Figure 3. Convergence analyses on the Cora and CiteSeer datasets.

E. PyTorch-style Pseudo Code

We give the PyTorch-style pseudo code of our proposed *Dink-Net* in Code 1. The source codes are released on GitHub platform: https://github.com/yueliu1999/Dink-Net.

F. Open Resource Supports

F.1. Awesome Deep Graph Clustering

This paper is supported by the Awesome Deep Graph Clustering [1] project at GitHub. Awesome Deep Graph Clustering project summarize a comprehensive collection of the state-of-the art deep graph clustering methods, including papers, codes, and datasets. In addition, based on this GitHub project, we make a comprehensive survey about deep graph clustering (Liu et al., 2022). Firstly, we give the formulaic definition of deep graph clustering and introduce the milestone baselines in this field. Secondly, the taxonomy of deep graph clustering methods is presented based on four different criteria, including graph type, network architecture, learning paradigm, and clustering method. Thirdly, we carefully analyze the existing methods via extensive experiments and summarize the challenges and opportunities from five perspectives. Besides, the applications of deep graph clustering methods in four domains are presented. We hope this work can serve as a quick guide and help researchers to overcome challenges in this vibrant field.

F.2. A Unified Framework of Deep Graph Clustering

In addition, this paper is supported by a unified framework of deep graph clustering[2] on GitHub. This GitHub project provides a practical unified framework of deep graph clustering methods. Concretely, it refactored the codes of recent state-of-the-art deep graph clustering methods to make them achieve a higher level of unification. The architecture of these codes is redesigned so that the researchers can run the open-source code efficiently. In addition, the defined tool classes and

[1] https://github.com/yueliu1999/Awesome-Deep-Graph-Clustering
[2] https://github.com/Marigoldwu/A-Unified-Framework-for-Deep-Attribute-Graph-Clustering

functions simplify the code and clarify the settings' configuration.

G. URLs of Used Datasets

This section gives the URLs of the used benchmark datasets in Table 2.

- Cora: https://docs.dgl.ai/#CoraGraphDataset
- CiteSeer: https://docs.dgl.ai/#dgl.data.CiteseerGraphDataset
- Amazon-Photo: https://docs.dgl.ai/#dgl.data.AmazonCoBuyPhotoDataset
- ogbn-arxiv: https://ogb.stanford.edu/docs/nodeprop/#ogbn-arxiv
- Reddit: https://docs.dgl.ai/#dgl.data.RedditDataset
- ogbn-products: https://ogb.stanford.edu/docs/nodeprop/#ogbn-products
- ogbn-papers100M: https://ogb.stanford.edu/docs/nodeprop/#ogbn-papers100M

References

Bo, D., Wang, X., Shi, C., Zhu, M., Lu, E., and Cui, P. Structural deep clustering network. In *Proc. of WWW*, 2020.

Devvrit, F., Sinha, A., Dhillon, I., and Jain, P. S3gc: Scalable self-supervised graph clustering. 2022.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Guo, X., Gao, L., Liu, X., and Yin, J. Improved deep embedded clustering with local structure preservation. In *Proc. of IJCAI*, 2017.

Hartigan, J. A. and Wong, M. A. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 1979.

Hassani, K. and Khasahmadi, A. H. Contrastive multi-view representation learning on graphs. In *Proc. of ICML*, 2020.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *Proc. of ICLR*, 2017.

Li, X., Zhang, H., and Zhang, R. Adaptive graph auto-encoder for general data clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Liu, Y., Xia, J., Zhou, S., Wang, S., Guo, X., Yang, X., Liang, K., Tu, W., Li, Z. S., and Liu, X. A survey of deep graph clustering: Taxonomy, challenge, and application. *arXiv preprint arXiv:2211.12875*, 2022.

Pal, S. K. and Mitra, S. Multilayer perceptron, fuzzy sets, classifiaction. 1992.

Pan, S., Hu, R., Fung, S.-f., Long, G., Jiang, J., and Zhang, C. Learning graph embedding with adversarial training methods. *IEEE transactions on cybernetics*, 2019.

Thakoor, S., Tallec, C., Azar, M. G., Munos, R., Veličković, P., and Valko, M. Bootstrapped representation learning on graphs. In *ICLR 2021 Workshop on Geometrical and Topological Representation Learning*, 2021.

Tsitsulin, A., Palowitch, J., Perozzi, B., and Müller, E. Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904*, 2020.

Tu, W., Zhou, S., Liu, X., Guo, X., Cai, Z., Cheng, J., et al. Deep fusion clustering network. *arXiv preprint arXiv:2012.09600*, 2020.

Velickovic, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., and Hjelm, R. D. Deep graph infomax. *ICLR (Poster)*, 2019.

Von Luxburg, U. A tutorial on spectral clustering. *Statistics and computing*, 2007.

Wang, C., Pan, S., Long, G., Zhu, X., and Jiang, J. Mgae: Marginalized graph autoencoder for graph clustering. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017.

Wang, C., Pan, S., Hu, R., Long, G., Jiang, J., and Zhang, C. Attributed graph clustering: A deep attentional embedding approach. *arXiv preprint arXiv:1906.06532*, 2019.

Xie, J., Girshick, R., and Farhadi, A. Unsupervised deep embedding for clustering analysis. In *Proc. of ICML*, 2016.

Zhao, H., Yang, X., Wang, Z., Yang, E., and Deng, C. Graph debiased contrastive learning with joint representation clustering. In *Proc. of IJCAI*, 2021.

Zheng, Y., Pan, S., Lee, V. C., Zheng, Y., and Yu, P. S. Rethinking and scaling up graph contrastive learning: An extremely efficient approach with group discrimination. *arXiv preprint arXiv:2206.01535*, 2022.

Zhu, Y., Xu, Y., Yu, F., Liu, Q., Wu, S., and Wang, L. Deep Graph Contrastive Representation Learning. In *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

Algorithm 1 PyTorch-style Pseudo Code of *Dink-Net*.

```
# G: attribute graph
# F: GNN encoder
# P: MLP projector
# discriminate_loss: loss in Eq. (4)
# dilation_loss: loss in Eq. (5)
# shrink_loss: loss in Eq. (6)
# K_Means_plus_plus: initialization of K-Means++
# alpha: trade-off parameter

# Model Pre-training Stage
pre_optimizer = Adam(lr=pretrain_lr, parameter=Dink_net.parameters())
for epoch in range(pretrain_epochs):
   sub_G_list = sub_graph_sampling(G)
   # batch training
   for sub_G in sub_G_list:
      sub_G_augmented = data_augmentation(sub_G)
      H = F(sub_G)
      H_ = F(sub_G_augmented)
      Z = P(H)
      Z_ = P(H_)
      dis_loss = discriminate_loss(Z.sum(-1), Z_.sum(-1))
      pre_optimizer.zero_grad()
      dis_loss.backward()
      pre_optimizer.step()
H_all = F(G)
C = K_Means_plus_plus(H_all)

# Model Fine-tuning Stage
fine_optimizer = Adam(lr=finetune_lr, parameter=Dink_net.parameters())
for epoch in range(finetune_epochs):
   sub_G_list = sub_graph_sampling(G)
   # batch training
   for sub_G in sub_G_list:
      sub_G_augmented = data_augmentation(sub_G)
      H = F(sub_G)
      H_ = F(sub_G_augmented)
      Z = P(H)
      Z_ = P(H_)
      dis_loss = discriminate_loss(Z.sum(-1), Z_.sum(-1))
      dil_loss = dilation_loss(H, C)
      shr_loss = shrink_loss(H, C)
      total_loss = dil_loss + shr_loss + alpha * dis_loss
      fine_optimizer.zero_grad()
      total_loss.backward()
      fine_optimizer.step()

# Model Inference Stage
y_hat = []
sub_G_list = sub_graph_sampling(G)
for sub_G in sub_G_list:
   H = F(sub_G)
   y_hat_batch = argmin(distance(H, C))
   y_hat.append(y_hat_batch)
return y_hat
```