Differentiable Clustering for Graph Attention

Haicang Zhou, Tiantian He, *Member, IEEE*, Yew-Soon Ong, *Fellow, IEEE*, Gao Cong, and Quan Chen

Abstract—Graph clusters (or communities) represent important graph structural information. In this paper, we present Differentiable Clustering for graph ATtention (DCAT). To the best of our knowledge, DCAT is the first solution that incorporates graph clustering into graph attention networks (GAT) to learn cluster-aware attention scores for semi-supervised learning tasks. In DCAT, we propose a novel approach to formulating graph clustering as an auxiliary differentiable objective based on modularity maximization, which can be optimized together with the learning objective of GAT for a semi-supervised task. Specifically, we propose a solution to relaxing modularity maximization from a discrete optimization problem to a differentiable objective with theoretical guarantee so that we can learn cluster-aware attention scores by jointly learning from graph clustering and a semi-supervised learning task. To address the computational challenge, we further propose to reformulate the constraint introduced by the clustering objective into a new form. Our analysis shows that DCAT allocates higher attention scores to nodes within the same cluster, allowing them to have a higher influence in node representation learning, and thus DCAT will generate better node representations for downstream applications. The experimental results on commonly used datasets show that DCAT outperforms popular and state-of-the-art graph neural networks.

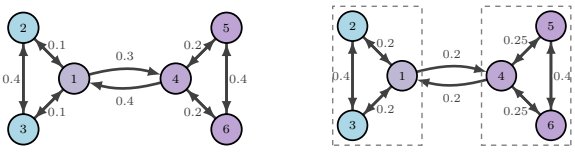
Index Terms—Graph neural networks, graph attention networks, attention mechanism, semi-supervised learning, graph clustering

1 Introduction

Graph learning [1], [2], [3] has been a hot field in both machine learning and data mining communities. Semi-supervised representation learning is a classical problem of graph learning and has been successfully adapted to address many real-world problems, such as document classification [4], [5], chemical compounds prediction [6], drug discovery [7], machine language understanding [5], [8], and recommender system [9], [10]. Attention-based graph neural networks (GNNs) [8], [11], [12], [13] often achieve state-of-the-art results for semi-supervised representation learning tasks on graphs. Graph attention networks (GATs) use node features to compute the attention coefficient for each neighbor of a node and learn representations by aggregating the features of the neighboring nodes. The learned representations can be used for various real-world applications.

Research works on using node features to compute attention coefficients in attention-based GNNs are common. Activities on using graph structures for computing attention coefficients in GNNs have on the other hand been observed to increase recently. Node degrees [14], node-node distance [14], [15], and structural similarity between nodes [12] are among some of the single-node and pairwise structures that have been considered lately. Nonetheless, the study on graph clusters or communities (i.e., groups of nodes in a graph), which serve as valuable information about the structure of a graph, has remained unexplored in attention-based GNNs for semi-supervised learning tasks.

Graph clusters represent very important graph structures and widely exist in various types of graphs [16], [17]. We illustrate the potential benefit of considering graph clusters in attention-based GNNs with a simple example in Fig. 1. The graph contains two cluster structures and the colors represent the features of the nodes. According to clustering properties, node 1 should have high similarity to nodes 2 and 3, as they are in the same cluster. However, without considering cluster structures (Fig. 1a), the attention score between nodes 1 and 4 would be much higher than that between nodes 1 and 2 (or 3), as nodes 1 and 4 are more similar in terms of color, i.e., node features. In contrast, by incorporating graph clusters into graph attention networks (Fig. 1b), we will be able to attend to nodes within the same cluster with a higher attention score so that they will have a higher influence on each other in learning node representations.



(a) Attention scores based on node features only

(b) Attention scores based on node features and graph clusters

Fig. 1. An example of a graph, its clusters, and the attention scores, with self-loops omitted for simplicity. The dashed rectangles indicate two clusters in the graph. The numbers in the nodes are their indices, while the colors represent the feature of the nodes. The numbers on the edges are the learned attention scores. The attention scores in Fig. 1 (a) do not learn from clusters. The attention scores in Fig. 1 (b) learn from both node features and graph clusters, so the attention scores for the nodes in the same cluster are higher than those in Fig. 1 (a).

Although utilizing cluster structures in attention-based GNNs appears to be a natural and promising idea as il-

- *H. Zhou, Y. S. Ong and G. Cong are with School of Computer Science and Engineering, Nanyang Technological University, Singapore, 639798. E-mail: {haicang001@e.,asysong@, gaocong@}ntu.edu.sg*
- *T. He is with Center for Frontier AI Research, Institute of High Performance Computing, Singapore Institute of Manufacturing Technology, Agency for Science, Technology and Research (A*STAR), Singapore, 138632. E-mail: he_tiantian@cfar.a-star.edu.sg*
- *Q. Chen is with Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU), Singapore, 639798.*
- *Y. S. Ong is also with Agency for Science, Technology and Research (A*STAR), Singapore, 138632. E-mail: ong_yew_soon@hq.a-star.edu.sg*

Manuscript received April 19, 2005; revised August 26, 2015.

lustrated in Fig 1, to the best of our knowledge, clusters have not been explored in existing attention-based GNNs designed for semi-supervised learning tasks. However, We note that clusters have been utilized by other types of work, such as learning graph embedding under the framework of matrix factorization for unsupervised learning tasks [18]. Such work further motivates us to explore the idea of utilizing clusters in attention-based GNNs to compute cluster-aware attention scores, although the setting of such work is essentially different from that of our work. One straightforward idea is to generate clusters of a graph and then concatenate the vectorized cluster information to the input feature of nodes for graph attention networks. However, this simple idea does not work well in attention-based GNNs in our preliminary experiments, since the impact of additional structural information such as graph clusters in the input will decay layer by layer [19], and cannot be well reflected in the learned attention scores.

To better utilize clustering information in attention-based GNNs for semi-supervised learning tasks, in this paper, we propose a novel attention-based GNN to learn cluster-aware attention scores. For the first time, we incorporate graph clustering into attention-based GNNs as part of the objective function for optimization to enhance the predictive performance of semi-supervised learning tasks. To make the idea feasible, two challenges need to be addressed. Firstly, most clustering algorithms neither have learnable parameters nor support automatic differentiation [20], [21], and thus cannot be formulated as part of the objective of neural networks so that we can optimize them together. For example, modularity-based methods [16], [17] and graph cuts [22] are discrete optimization problems, while spectral clustering [23] requires eigen-decomposition. Secondly, graph clustering algorithms usually have strict constraints (e.g., the constraints required by modularity optimization [17], [24] and graph cuts [25], [26]). Even though we could find a way to reformulate the clustering objective of a clustering algorithm as an auxiliary objective to the learning objective of neural networks for a semi-supervised task, the constraints from the clustering objective will make the training very challenging in terms of both efficiency and effectiveness [27], [28].

To overcome the aforementioned challenges, in this paper, we propose a novel Differentiable Clustering for graph ATtention (named DCAT) for semi-supervised learning on graphs. In DCAT, we propose a novel approach to formulating graph clustering as an auxiliary differentiable objective, which can be optimized together with the learning objective of graph attention networks (for a semi-supervised task). Specifically, we propose to relax the original discrete modularity maximization to an objective of continuous optimization by means of spectral relaxation. We then use graph attention layers to parameterize the converted objective. We prove that our relaxation method has a theoretical guarantee that the solution of the relaxed objective is an effective approximation that is very close to the solution of the original modularity maximization clustering objective. However, the constraint of the clustering algorithm on the output of graph attention layers severely reduces the efficiency and effectiveness in the training of neural networks [27], [28]. To resolve this, we reformulate the constrained optimization objective of graph clustering into an unconstrained and easy-to-optimize one [28]. Through the joint learning of the reformulated objective and semi-supervised loss of the neural networks, the parameter learning of graph attention in DCAT can benefit from the inductive bias from graph clustering. We analyze the computational cost of the proposed DCAT model, and we illustrate that through learning from the clustering objective, DCAT will allocate smaller attention scores to nodes in different clusters and larger attention scores to nodes in the same cluster.

The contributions of the proposed approach are summarized as follows:

1) We propose DCAT. To the best of our knowledge, it is the first solution to incorporate graph clustering as an auxiliary objective into graph attention networks to learn cluster-aware attention scores for semi-supervised learning tasks. In DCAT, we parameterize the objective of clustering with graph attention layers. By jointly learning with a semi-supervised loss and the clustering objective, DCAT will learn graph clusters and utilize clusters to learn cluster-aware attention scores.
2) To design a differentiable clustering objective, which is used as an auxiliary learning objective together with the learning objective of a semi-supervised learning task, we propose an approach to deriving a continuous relaxed form of modularity maximization. We demonstrate that our relaxation method has a theoretical guarantee that it well-preserves clustering information. We further propose a solution to reformulating the constraint introduced by the relaxation to a new form that can be solved with better computational cost.
3) We conduct extensive experiments on two semi-supervised learning tasks to evaluate the effectiveness of DCAT. The empirical results show that DCAT outperforms popular and state-of-the-art GNNs.

2 Related Work

In this section, we investigate the work related to our proposed DCAT, including *attention on graphs* and *GNNs for graph clustering*.

2.1 Attention on Graphs

Attention-based GNNs [11], [12], [13], [29], [30], [31], [32] follow the message passing paradigm [1], [6] to learn representations of nodes. These methods adopt the attention mechanism [33], [34] to generate learnable attention scores instead of static weights for aggregation. However, the graph attention mechanism from [11] computes attention scores only according to node features, and ignores important inductive bias from graph structure. Recently, several efforts target at injecting graph structure into graph attention [19], [35]. For example, some approaches [14], [31], [36] compress various graph structures to vectors and add (or concatenate) them to the inputs. Others carefully revise the computation formulation of graph attention [14], [15] to incorporate graph structure, such as adding structure

bias [14]. Our DCAT is fundamentally different from these approaches. Rather than changing the input of neural networks, or the formulation of graph attention, DCAT adopts a novel clustering objective that is parameterized by graph attention layers. In this way, graph attention can seamlessly learn the inductive bias from graph clusters.

2.2 GNNs for Graph Clustering

In recent years, there is a trend to solve traditional clustering algorithms with neural networks. Those methods are called deep clustering or differentiable clustering [37]. Among these work, some design graph neural networks for graph clustering [24], [25], [26], [38]. Our DCAT is fundamentally different from these GNNs for graph clustering in the following perspectives. First, those clustering approaches are designed for unsupervised tasks on graphs while our method targets semi-supervised learning tasks. Second, we provide novel solutions to handle the orthonormal constraints [26], [39] of the clustering objective, which have not been thoroughly investigated in previous works. Specifically, [24], [38], [39] do not consider the orthogonal constraint, while the constraint in [26] is highly non-convex.

3 Notations

Before introducing the proposed DCAT, we list the essential notations and preliminaries used in this paper.

Vectors and Matrices. We use column vectors by default unless stated otherwise. A matrix is denoted as $\boldsymbol{A} \in \mathbb{R}^{n\times m}$. $\boldsymbol{A}_{i,j}$ denotes the element of $\boldsymbol{A}$ in the i-th row and j-th column. $\boldsymbol{A}_i$ denotes the i-th row and $\boldsymbol{A}_{:,j}$ denotes the j-th column of matrix $\boldsymbol{A}$. A matrix can also be represented by columns of vectors: $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m] \in \mathbb{R}^{n\times m}$, with $\boldsymbol{v}_i = \boldsymbol{V}_{:,i}$. We use $\|\cdot\|$ to denote the norm of a vector or a matrix, and $\mathrm{Tr}(\cdot)$ to denote the trace of a matrix, i.e., $\mathrm{Tr}(\boldsymbol{A}) = \sum_i \boldsymbol{A}_{i,i}$.

Graph. In this paper, we consider undirected graphs. A graph is denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{X})$, with $\mathcal{V}$ and $\mathcal{E}$ being the set of nodes and edges respectively. Let $n = |\mathcal{V}|$ denote the number of nodes and $m = |\mathcal{E}|$ denote the number of edges. $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ is the feature matrix of nodes, with d being the dimension of feature vectors. The adjacency matrix is denoted as $\boldsymbol{A} \in \{0,1\}^{n\times n}$. If node i and node j are connected, $\boldsymbol{A}_{i,j} = 1$; otherwise $\boldsymbol{A}_{i,j} = 0$. As we consider undirected graphs, $\boldsymbol{A}$ is a real symmetric matrix. The neighborhood of node i is denoted as $\mathcal{N}_i := \{j | \{i,j\} \in \mathcal{E}\} \cup \{i\}$. Let d_i denote the degree of node i and let $\boldsymbol{D}$ denote the degree matrix of $\mathcal{G}$, i.e., a diagonal matrix with the i-th diagonal element $\boldsymbol{D}_{i,i} = d_i$. In a GNN layer, we denote the input by $\boldsymbol{H}$ and the output by $\boldsymbol{H}'$.

Graph Laplacian Matrix. The Laplacian matrix of a graph is denoted as $\boldsymbol{L} := \boldsymbol{D} - \boldsymbol{A}$. It has a normalized version $\boldsymbol{L} := \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{L}\boldsymbol{D}^{-\frac{1}{2}} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$. In the following parts, we will use the normalized graph Laplacian matrix. For an undirected graph, $\boldsymbol{L}$ is a real symmetric matrix, which has n real eigenvalues $0 \le \lambda_1 \le \cdots \le \lambda_n \le 2$ and n corresponding orthonormal eigenvectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$. "Orthonormal" means, $\boldsymbol{u}_i$ is orthogonal to $\boldsymbol{u}_j, \forall i \ne j$ and $\|\boldsymbol{u}_i\|_2 = 1, \forall i$.

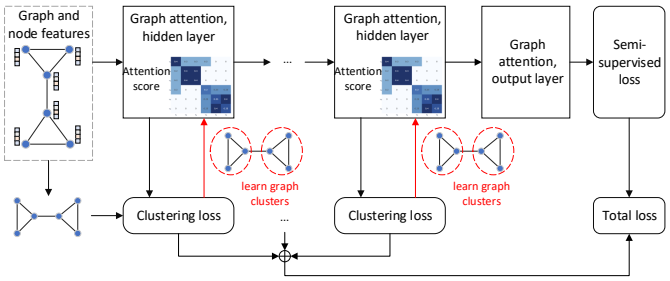


Fig. 2. Schematic view of the proposed DCAT. We design a clustering loss on the hidden layer to learn cluster-aware attention scores, as the red line shows. The clustering loss is an auxiliary objective in addition to the original semi-supervised loss.

4 Differentiable Clustering for Graph Attention

In this section, we present Differentiable Clustering for graph ATtention (DCAT) for semi-supervised learning tasks.

4.1 Overview

As discussed in Introduction, we aim to use graph clusters to improve graph attention networks so that we will be able to attend to nodes within the same cluster with higher attention scores, i.e., they will have larger influences on each other in learning node representations. To achieve this, our idea is to formulate graph clustering as an auxiliary learning objective, and combine it with the learning objective of a semi-supervised task so that graph attention will benefit from jointly learning from graph clustering and a semi-supervised task. To make the idea work, we address the two challenges mentioned in Section 1 as follows.

First, popular graph clustering algorithms usually are neither differentiable nor have learnable parameters, and thus they cannot be formulated as an auxiliary learning objective, which we expect to optimize together with the learning objective of neural networks for a semi-supervised task. To address the challenge of formulating an effective and differentiable auxiliary learning objective for clustering, we develop a method to relax the modularity maximization problem [16], [17], which is a discrete optimization problem for effective clustering, and is one of the most popular optimization objective for graph clustering, to a continuous optimization problem. We prove that our relaxation method has a theoretical guarantee that the solution of the relaxed objective is an effective approximation that is very close to the solution of the original modularity maximization clustering objective(Section 4.2). To jointly optimize the reformulated clustering objective and the neural network objective (of a semi-supervised task) to learn cluster-aware attention scores, we propose to adapt graph attention layers to parameterize the differentiable clustering objective (Section 4.3).

Second, with the idea of parameterizing the differentiable clustering objective, DCAT is able to learn cluster-aware attention scores by jointly optimizing the auxiliary clustering objective together with the semi-supervised task objective. However, the constraints introduced by the auxiliary clustering objective make it very challenging to optimize the objective in terms of both efficiency and effective-

ness [27], [28]. To address this, we propose to reformulate the constraint introduced by the clustering objective into a new form with affordable computational cost, which is able to effectively preserve the properties of the original constraint. The clustering objective with the reformulated constraint is differentiable, effective, and efficient, and can be formulated as an auxiliary learning objective in neural networks (Section 4.4).

After solving the two challenges, we can build the proposed DCAT layer and stack the layers to build DCAT network. Figure 2 shows the overview of the proposed DCAT network. In the rest of this section, we will present the main components of DCAT model. In Section 4.6 we analyze the computational cost of DCAT, and in Section 4.7 we discuss the impact of the clustering objective on representation and attention scores. Specifically, we illustrate that through learning from the clustering objective, DCAT will allocate smaller attention scores to nodes in different clusters and larger attention scores to nodes in the same cluster.

4.2 Continuous Relaxation of Modularity Maximization

Here, we consider deriving the learning objective using modularity maximization, one of the most effective algorithms for graph clustering. However, as a discrete optimization problem, modularity maximization does not support automatic differentiation required by the training of neural networks. Thus, we propose to relax modularity maximization to a continuous form through spectral approximation.

Following [40], we define the modularity for multiple groups of nodes as follows.

Definition 1 (Modularity for multiple clusters [40])**.** For a given partition of nodes (into K groups) in a graph, the modularity is defined as follows,

$$Q = \frac{1}{2m} \sum_{i,j} \boldsymbol{B}_{i,j} \boldsymbol{S}'_i \boldsymbol{S}'^T_j, \tag{1}$$

where $\boldsymbol{B}_{i,j} = \boldsymbol{A}_{i,j} - \frac{d_i d_j}{2m}$, $\boldsymbol{S}'_i$ is the i-th row of matrix $\boldsymbol{S}' \in \{0,1\}^{n\times K}$ and

$$\boldsymbol{S}'_{i,k} := \begin{cases} 1, & \text{when node } i \text{ belongs to cluster k,} \\ 0, & \text{otherwise.} \end{cases}$$

Modularity maximization aims to find an optimum $\boldsymbol{S}' \in \{0,1\}^{n\times K}$ such that Q is maximized. The optimum $\boldsymbol{S}'_i$ indicates which cluster node i belongs to, and this introduces a constraint that a node should only belong to one cluster (i.e., $\sum_k \boldsymbol{S}'_{i,k} = 1, \forall i$). However, modularity maximization is NP-complete in general [41]. Therefore, an effective approximation for the solution of modularity maximization is required for graph clustering. A lot of efforts based on heuristic [16], [42], probabilistic methods [43], or spectral methods [44], [45] have been proposed to find an approximated solution for modularity maximization. Heuristic methods generally fall under discrete optimization, and thus they do not support neural network learning through automatic differentiation. Probabilistic methods usually have strong assumptions on the graph (e.g., Stochastic Block Model [46]), which may not hold in real-world data. In contrast, spectral methods have no additional assumption and give an effective continuous approximation [45] of modularity maximization, which is necessary for automatic differentiation in neural networks. Thus, we adapt the spectral method to find the approximated solution of modularity maximization by following [45]. Formally, we have:

Lemma 1 (Spectral approximation of modularity maximization for multiple clusters)**.** *Let $\boldsymbol{L}$ denote the graph Laplacian matrix, $\boldsymbol{S} \in \mathbb{R}^{n\times K}$ denote the continuous relaxation of $\boldsymbol{S}' \in \{0,1\}^{n\times K}$ and $\boldsymbol{X} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{S}$, then*

$$\boldsymbol{L}\boldsymbol{X} = \boldsymbol{X}\boldsymbol{\Lambda} \quad \textit{subject to } \boldsymbol{X}^T\boldsymbol{X} = c\boldsymbol{I}, \tag{2}$$

where c is a constant, gives the relaxed solution of the modularity maximization defined in Eq. (1).

The proof of Lemma 1 is provided in Section 5.1. The approximated solution $\boldsymbol{S}$ (and $\boldsymbol{X} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{S}$) in Lemma 1 contains the clustering information in $\boldsymbol{S}'$. Note that we additionally include an explicit constraint $\boldsymbol{X}^T\boldsymbol{X} = c\boldsymbol{I}$ (or its equivalent form $\boldsymbol{S}^T\boldsymbol{D}\boldsymbol{S} = c\boldsymbol{I}$). According to [45], relaxing $\boldsymbol{S}'$ to $\boldsymbol{S}$ onto a hyperellipsoid (i.e., $\boldsymbol{S}^T\boldsymbol{D}\boldsymbol{S} = c\boldsymbol{I}$) is essential to produce effectively approximated results.

Our goal is to incorporate modularity maximization into the learning process of graph attention. Lemma 1 gives the continuous relaxed solution of the original discrete objective. However, the eigen-decomposition has a high computational cost ($O(n^3)$), and it still does not support automatic differentiation. Thus it cannot be directly incorporated into the learning of graph neural networks. To tackle this issue, we propose to convert the eigen-decomposition (in Lemma 1) to an equivalent optimization problem, based on a variant of Rayleigh Ritz Theorem (Section 5.2.2.(6) of [47]):

Theorem 1 (Rayleigh Ritz Theorem)**.** *Assume a $(n \times n)$ real symmetric matrix $\boldsymbol{L}$ has eigenvalues $\lambda_1 \leq \cdots \leq \lambda_n$ and associated orthonormal eigenvectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$, $K \in \{1, \ldots, n\}$. Then for the following optimization problem*

$$\min_{\boldsymbol{X}\in\mathbb{R}^{n\times K}} \operatorname{Tr}\left(\boldsymbol{X}^T\boldsymbol{L}\boldsymbol{X}\right) \quad \textit{subject to } \boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{I}, \tag{3}$$

the minimizing matrix is $\boldsymbol{X}^\star = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K]$.

The solution of the optimization problem in Theorem 1 is exactly the eigen-vectors of the eigen-decomposition. If we change the constraint to $\boldsymbol{X}^T\boldsymbol{X} = c\boldsymbol{I}$, where c is a constant, the new solution will be $\sqrt{c}\boldsymbol{X}^\star$. Its only difference from the solution of Eq. (3) is a constant factor $\sqrt{c}$. With this objective, GNNs will be able to learn the clusters by jointly optimizing it with semi-supervised loss.

4.3 Clustering for Graph Attention

Aiming at incorporating clustering information into the learning of graph attention as an auxiliary learning objective, we propose to parameterize the clustering objective shown in Theorem 1 with graph attention layers.

Specifically, a graph attention layer can be formulated as

$$\boldsymbol{H}' = \sigma\left(\mathcal{A}\boldsymbol{H}\boldsymbol{W}^T\right), \tag{4}$$

where $\sigma(\cdot)$ denotes the non-linear activation function, $\boldsymbol{W}$ denotes the learnable parameter, $\boldsymbol{H}$ denotes the input to the layer, and $\boldsymbol{H}'$ denotes the output of the layer. $\mathcal{A}$ is the matrix containing the attention scores and $\mathcal{A}_{i,j}$ represents

the attention score from node i to node j. $\mathcal{A}_{i,j}$ is computed from the feature (or representation) of nodes:

$$\mathcal{A}_{i,j} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T(\boldsymbol{W}\boldsymbol{H}_i^T \parallel \boldsymbol{W}\boldsymbol{H}_j^T)))}{\sum_{k\in\mathcal{N}_i}\exp(\text{LeakyReLU}(\mathbf{a}^T(\boldsymbol{W}\boldsymbol{H}_i^T \parallel \boldsymbol{W}\boldsymbol{H}_k^T)))}. \tag{5}$$

Here $\boldsymbol{W}$ is the same parameter matrix as in Eq. (4). $\parallel$ denotes the concatenation operation of two vectors, and $\mathbf{a}$ is the parameter for attention.

In order to learn from graph clusters, we use a graph attention layer (without activation) $f_{\boldsymbol{W},\mathbf{a}}(\boldsymbol{A},\boldsymbol{H}) = \mathcal{A}\boldsymbol{H}\boldsymbol{W}^T$ to parameterize $\boldsymbol{X}$ in Theorem 1. Consequently the objective turns into

$$\begin{aligned} &\min_{\boldsymbol{W},\mathbf{a}} \text{Tr}(f_{\boldsymbol{W},\mathbf{a}}(\boldsymbol{A},\boldsymbol{H})^T\boldsymbol{L}f_{\boldsymbol{W},\mathbf{a}}(\boldsymbol{A},\boldsymbol{H})),\\ &\text{subject to}\quad f_{\boldsymbol{W},\mathbf{a}}(\boldsymbol{A},\boldsymbol{H})^T f_{\boldsymbol{W},\mathbf{a}}(\boldsymbol{A},\boldsymbol{H}) = c\boldsymbol{I}. \end{aligned} \tag{6}$$

During the learning of this objective, the learnable parameters of the graph attention layer, especially the parameter for attention $\mathbf{a}$, can learn from the cluster structure in the graph. For notation simplicity in the following elaboration, let $\boldsymbol{Z} = f_{\boldsymbol{W},\mathbf{a}}(\boldsymbol{A},\boldsymbol{H}) = \mathcal{A}\boldsymbol{H}\boldsymbol{W}^T$.

4.4 Reformulating Constraints in DCAT for Automatic Differentiation

The objective in Eq. (6) has a constraint $\boldsymbol{Z}^T\boldsymbol{Z} = c\boldsymbol{I}$, which is essential to produce effectively approximated results. Directly optimizing $\text{Tr}\left(\boldsymbol{Z}^T\boldsymbol{L}\boldsymbol{Z}\right)$ under this constraint is highly inefficient [28]. To make the objective in Eq. (6) differentiable for neural networks, we reformulate the constraint as part of the objective and neural network. Specifically, we decompose the constraint to a spherical constraint (i.e., fixing the scale of columns of $\boldsymbol{Z}$: $\|\boldsymbol{Z}_{:,j}\|_2 = c, \forall j$, where c is a constant) and an orthogonal constraint (i.e., controlling the columns of $\boldsymbol{Z}$ to be pair-wise orthogonal: $\boldsymbol{Z}_{:,j}^T\boldsymbol{Z}_{:,k} = 0, \forall j,k, j \neq k$). Subsequently we propose to preserve these two constraints through normalization and orthogonal loss, which will make the objective differentiable and efficient to learn.

The normalization can strictly fix the scale of the columns of $\boldsymbol{Z}$, which is guaranteed by the following lemma.

Lemma 2. *Let $\boldsymbol{Z}_{:,j}$ denote the j-th column of matrix $\boldsymbol{Z} \in \mathbb{R}^{n\times k}$ and μ_j, σ_j denote the mean and the standard deviation of values in $\boldsymbol{Z}_{:,j}$ respectively. Normalization is applied to $\boldsymbol{Z}$ to generate $\hat{\boldsymbol{Z}} \in \mathbb{R}^{n\times k}$ where*

$$\hat{\boldsymbol{Z}}_{:,j} = \frac{g_j}{\sigma_j}(\boldsymbol{Z}_{:,j} - \mu_j) \tag{7}$$

Then $\hat{\boldsymbol{Z}}_{:,j}^T\hat{\boldsymbol{Z}}_{:,j} = ng_j^2, \forall j \in \{1,\ldots,k\}$, where n is the number of nodes in the graph.

The proof can be found in Section 5.2. Lemma 2 further gives the value of $\hat{\boldsymbol{Z}}_{:,j}^T\hat{\boldsymbol{Z}}_{:,j} = ng_j^2, \forall j$, where n is the number of nodes in the input graph. If we choose g_js to be the same fixed value, e.g., $g_j = 1, \forall j$, we will get $\|\hat{\boldsymbol{Z}}_{:,j}\|_2 = \sqrt{n}$. Given an input graph, n is a constant, thus the length of the columns is fixed. In addition, the normalization prevents gradient from vanishing or exploding during neural network training.

To achieve the orthogonal effect, we adopt an orthogonal loss [18], [48] to penalize the inner product of different columns of $\hat{\boldsymbol{Z}}$:

$$\begin{aligned} \mathcal{L}_o &= \left\|\hat{\boldsymbol{Z}}^T\hat{\boldsymbol{Z}} - n\boldsymbol{I}\right\|_F^2 \\ &= \sum_{\substack{i,j=1,\\ i=j}}^{d}\left\|\hat{\boldsymbol{Z}}_{:,i}^T\hat{\boldsymbol{Z}}_{:,j} - n\right\|_2^2 + \sum_{\substack{i,j=1,\\ i\neq j}}^{d}\left\|\hat{\boldsymbol{Z}}_{:,i}^T\hat{\boldsymbol{Z}}_{:,j} - 0\right\|_2^2, \end{aligned} \tag{8}$$

with d being the number of columns of $\hat{\boldsymbol{Z}}$. If $i = j$, the loss will penalize to push the length of each column to be the same, which has similar effect to the normalization. If $i \neq j$, as the columns of $\hat{\boldsymbol{Z}}$ have the same fixed length, penalizing the inner product of $\hat{\boldsymbol{Z}}_{:,i}$ and $\hat{\boldsymbol{Z}}_{:,j}$ is equivalent to penalizing them to be orthogonal. Thus the second term penalizes every two columns of $\hat{\boldsymbol{Z}}$ to be pair-wise orthogonal.

Equipped with the normalization and the orthogonal loss, one DCAT layer is formulated as follows:

$$\begin{aligned} \hat{\boldsymbol{Z}} &= \text{Norm}(\mathcal{A}\boldsymbol{H}\boldsymbol{W}^T),\\ \boldsymbol{H}' &= \sigma(\hat{\boldsymbol{Z}}),\\ \mathcal{L}_c &= \mathcal{L}_{tr} + \mathcal{L}_o = \frac{1}{n}\text{Tr}\left(\hat{\boldsymbol{Z}}^T\boldsymbol{L}\hat{\boldsymbol{Z}}\right) + \left\|\frac{1}{n}\hat{\boldsymbol{Z}}^T\hat{\boldsymbol{Z}} - \boldsymbol{I}\right\|_F^2, \end{aligned} \tag{9}$$

where $\mathcal{L}_c$ is the clustering loss, which is differentiable in neural networks. DCAT first generates the hidden representation by attention mechanism and normalization. The hidden representation is then used to compute the clustering loss and produce the layer output through non-linear activation. By learning the clustering loss, the information of clusters is learned in the hidden representation $\hat{Z}$. In the formula of clustering loss, with the factor $\frac{1}{n}$, $\mathcal{L}_c$ will not grow with the number of nodes in a graph. We also find that setting another hyper-parameter to balance the two terms in the clustering loss is unnecessary, as the current formulation can produce promising performance in our experiment.

4.5 The Learning Objective

With the DCAT layers, we can build DCAT Networks for the downstream graph learning tasks. Similar to [11], [34], we adopt multi-head attention to "stabilize the learning". Specifically, we concatenate outputs from different attention heads (Eq. (9)) and compute the clustering loss for multiple attention heads as follows:

$$\mathcal{L}_c = \frac{1}{n}\text{Tr}\left(\hat{\boldsymbol{Z}}^T\boldsymbol{L}\hat{\boldsymbol{Z}}\right) + \left\|\frac{1}{n}\hat{\boldsymbol{Z}}^T\hat{\boldsymbol{Z}} - \boldsymbol{I}\right\|_F^2. \tag{10}$$

The first term is the summation of the traces from multiple attention heads, while the second term adds an orthogonal constraint to every two columns of $\hat{\boldsymbol{Z}}$, i.e., penalizes $\hat{Z}_{:,i}$ and $\hat{\boldsymbol{Z}}_{:,j}$ to be orthogonal for every $i \neq j$.

To avoid the decay of the clustering information over layers, instead of computing $\mathcal{L}_c$ only from the output, we sum up the clustering loss from each hidden layer:

$$\mathcal{L}_c = \frac{1}{L-1}\sum_{l=1}^{L-1}\mathcal{L}_c^{(l)} \tag{11}$$

where L is the number of layers, and the superscript $\cdot^{(l)}$ denotes $\mathcal{L}_c$ being in the l-th layer. Our DCAT Networks

can learn from the semi-supervised loss and clustering loss jointly. The total loss is formulated as follows:

$$\mathcal{L} = (1-\beta)\mathcal{L}_{\text{semi}} + \beta\mathcal{L}_c. \tag{12}$$

Here β is a hyper-parameter in the range of $(0, 1)$ to balance the impact of the information from semi-supervised loss and graph clusters.

4.6 Computational Cost

In this subsection, the computational complexity of DCAT is analyzed and compared with classical graph attention networks [11]. Firstly, the proposed DCAT does not introduce new learnable parameters and has the same parameters as classical graph attention. Secondly, $\mathcal{L}_c$, the clustering loss, does not cause a huge computational cost. Let the dimensions of $\hat{\boldsymbol{Z}}$ and $\boldsymbol{Z}$ be $(n \times d)$, where n is the number of nodes in the graph. The trace can be efficiently computed as $\text{Tr}(\hat{\boldsymbol{Z}}^T\boldsymbol{L}\hat{\boldsymbol{Z}}) = \sum_{i,j}\boldsymbol{A}_{ij}(\hat{\boldsymbol{Z}}_i - \hat{\boldsymbol{Z}}_j)^T(\hat{\boldsymbol{Z}}_i - \hat{\boldsymbol{Z}}_j)$. Since $\boldsymbol{A}$ is the sparse adjacent matrix, we only need to compute the term in the summation for connected node pairs, and thus the complexity is $O(|\mathcal{E}|d)$, where $|\mathcal{E}|$ denotes the number of edges in a graph. The complexity of orthogonal loss is $O(nd^2)$. Both are affordable for GNNs (with the complexity $O((|\mathcal{E}|+n)d^2)$ for each layer) even on large graphs.

4.7 Remarks

Modularity is well known for effective evaluation of the clusters in graph data. Modularity maximization comes from the observation that the number of edges between groups is significantly smaller than expected if the edges are generated randomly according to a uniform distribution between any two nodes [16]. The nodes within each group are closely connected, and thus each group forms a cluster. Eq. (1) formulates this observation, and therefore a higher modularity score means that there are more edges within clusters and fewer between them [17]. In the following paragraphs, we will analyze how the objective of modularity maximization affects the representation and attention scores learned by graph attention.

Specifically, from the definition of modularity (Definition 1), we observe that if node i and j are in different clusters, $\boldsymbol{S}'_i$ and $\boldsymbol{S}'_j$ should be different (e.g., $\boldsymbol{S}'_i\boldsymbol{S}'^T_j = 0$). As $\boldsymbol{S}_i$ and $\boldsymbol{S}_j$ are the approximations of $\boldsymbol{S}'_i$ and $\boldsymbol{S}'_j$, $\boldsymbol{S}_i$ and $\boldsymbol{S}_j$ should also be very different. If node i and node j are in the same cluster, $\boldsymbol{S}_i$ and $\boldsymbol{S}_j$ should be similar. As $\boldsymbol{X} = \boldsymbol{D}^{\frac{1}{2}}\boldsymbol{S}$ (defined in Lemma 1), where $\boldsymbol{D}$ is a fixed matrix given a graph, $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ should also be distinct if node i and node j are in different clusters and vice versa.

In DCAT, we replace $\boldsymbol{X}_i$ with $\hat{\boldsymbol{Z}}_i$, where

$$\hat{\boldsymbol{Z}}_i = \text{Norm}(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\boldsymbol{W}\boldsymbol{H}_j).$$

According to the analysis presented above, $\hat{\boldsymbol{Z}}_i$ and $\hat{\boldsymbol{Z}}_j$ should be distinct if node i and node j are in different clusters and vice versa. This can be achieved by attending less to neighbors in different clusters and attending more to neighbors in the same cluster. In other words, α_{ij} should be decreased if node i and node j are in different clusters and vice versa. The effect of the clustering loss on the attention scores is further shown in our experiment (Section 6.4).

4.8 When do we expect this approach to work?

This paper focuses on scenarios wherein each node is allocated to a disjoint cluster, and our experiments demonstrate that, clustering can indeed improve node classification. For a node belonging to multiple communities (i.e., cluster) or across the community boundary, we can assign the most probable community to it. We believe that considering nodes belonging to multiple communities or across the community boundary is an interesting future direction.

5 Proofs

In this section, we present the proofs for Lemma 1 and Lemma 2.

5.1 Proofs for Lemma 1

Here, we provide the full proof for Lemma 1. We follow [45] and extend its proof of the relaxed solution for modularity maximization to matrix form.

Proof. The modularity is defined as follows,

$$Q = \frac{1}{2m}\sum_{i,j}\boldsymbol{B}_{i,j}\boldsymbol{S}'_i\boldsymbol{S}'^T_j,$$

where m is the number of edges in the graph, and the matrix $\boldsymbol{S}'$ is defined as follows,

$$\boldsymbol{S}'_{i,k} := \begin{cases} 1, & \text{when node } i \text{ belongs to cluster k,} \\ 0, & \text{otherwise.} \end{cases}$$

Let $\boldsymbol{S}$ be the relaxed solution of the multi-way modularity maximization problem. Then we also follow the proof of two clusters [45] to avoid finding a trivial solution (e.g., assigning all the nodes in one cluster) by a constraint as,

$$\sum_i d_i\boldsymbol{S}_i\boldsymbol{S}_i^T = c,$$

where d_i is the degree of node i and c is a constant. Then we adopt a Lagrange multiplier $(1-\lambda)$, and get the following form of relaxed modularity maximization

$$L = \sum_{i,j}\boldsymbol{B}_{i,j}\boldsymbol{S}_i\boldsymbol{S}_j^T - (1-\lambda)\left(\sum_i d_i\boldsymbol{S}_i\boldsymbol{S}_i^T - c\right).$$

Note that we omit the constant factor $1/2m$, which does not affect the solution. The maximum point can be found by setting the derivatives to 0.

$$\frac{\partial L}{\partial \boldsymbol{S}_l} = \frac{\partial}{\partial \boldsymbol{S}_l}(\sum_{i,j}\boldsymbol{B}_{i,j}\boldsymbol{S}_i\boldsymbol{S}_j^T - (1-\lambda)\sum_i d_i\boldsymbol{S}_i\boldsymbol{S}_i^T) = \boldsymbol{0},$$

where $\boldsymbol{0}$ is a vector with all elements being 0. The solution of this equation is

$$\sum_j \boldsymbol{B}_{i,j}\boldsymbol{S}_j = (1-\lambda)d_i\boldsymbol{S}_i, \quad \forall i$$

which is equivalent to

$$\boldsymbol{B}\boldsymbol{S} = \boldsymbol{D}\boldsymbol{S}(\boldsymbol{I} - \boldsymbol{\Lambda}),$$

where $\boldsymbol{D}$ is the diagonal degree matrix with $\boldsymbol{D}_{i,i} = d_i$. Meanwhile the matrix form of the constraint ($\sum_i d_i\boldsymbol{S}_i\boldsymbol{S}_i^T =$

c) is $\boldsymbol{S}^T\boldsymbol{D}\boldsymbol{S} = c\boldsymbol{I}$, which constrains $\boldsymbol{S}$ to fall on a hyperellipsoid. This equation can be further simplified as follows. Replacing $\boldsymbol{B}_{i,j}$ with $\boldsymbol{A}_{i,j} - \frac{d_i d_j}{2m}$, for each row of $\boldsymbol{B}\boldsymbol{S} = \boldsymbol{D}\boldsymbol{S}(\boldsymbol{I} - \boldsymbol{\Lambda})$, we have

$$\sum_j (\boldsymbol{A}_{i,j} - \frac{d_i d_j}{2m})\boldsymbol{S}_j = d_i \boldsymbol{S}_i(\boldsymbol{I} - \boldsymbol{\Lambda})$$

$$\sum_{i,j} \boldsymbol{A}_{i,j}\boldsymbol{S}_j - \sum_{i,j} \frac{d_i d_j}{2m}\boldsymbol{S}_j = \sum_i d_i \boldsymbol{S}_i(\boldsymbol{I} - \boldsymbol{\Lambda})$$

$$\sum_j (\boldsymbol{S}_j \sum_i \boldsymbol{A}_{i,j}) - (\sum_i \frac{d_i}{2m})(\sum_j d_j \boldsymbol{S}_j) = \sum_i d_i \boldsymbol{S}_i(\boldsymbol{I} - \boldsymbol{\Lambda})$$

As $\sum_i \boldsymbol{A}_{i,j} = d_i$ and $\sum_i \frac{d_i}{2m} = 1$, the left side is equal to zero. Thus we have

$$\sum_i d_i \boldsymbol{S}_i(\boldsymbol{I} - \boldsymbol{\Lambda}) = 0$$

According to the Lagrange multiplier, $(\boldsymbol{I} - \boldsymbol{\Lambda})$ cannot be a matrix of 0s, so $\sum_i d_i \boldsymbol{S}_i = \boldsymbol{0}$. Then $\boldsymbol{B}\boldsymbol{S} = \boldsymbol{D}\boldsymbol{S}(\boldsymbol{I} - \boldsymbol{\Lambda})$ can be simplified with

$$\sum_j \boldsymbol{A}_{i,j}\boldsymbol{S}_j - \frac{d_i}{2m}\sum_j d_j \boldsymbol{S}_j = d_i \boldsymbol{S}_i(\boldsymbol{I} - \boldsymbol{\Lambda})$$

$$\sum_j \boldsymbol{A}_{i,j}\boldsymbol{S}_j = d_i \boldsymbol{S}_i(\boldsymbol{I} - \boldsymbol{\Lambda}),$$

which is equivalent to

$$\boldsymbol{A}\boldsymbol{S} = \boldsymbol{D}\boldsymbol{S}(\boldsymbol{I} - \boldsymbol{\Lambda}).$$

Let $\boldsymbol{X} := \boldsymbol{D}^{-1/2}\boldsymbol{S}$, we have

$$\boldsymbol{L}\boldsymbol{X} = \boldsymbol{X}\boldsymbol{\Lambda}.$$

Meanwhile the constraint for $\boldsymbol{X}$ is $\boldsymbol{X}^T\boldsymbol{X} = c\boldsymbol{I}$ (from $\boldsymbol{S}^T\boldsymbol{D}\boldsymbol{S} = c\boldsymbol{I}$), where c is a constant. □

5.2 Proof for Lemma 2

In Lemma 2, we prove the effect of normalization, which shows that normalization can achieve the spherical constraint (i.e., fix the scale of columns of matrix). As a by-product, Lemma 2 also gives the settings of normalization.

Proof. For notation simplicity, let $\hat{\boldsymbol{z}} = \hat{\boldsymbol{Z}}_{:,j}$ and $\boldsymbol{z} = \boldsymbol{Z}_{:,j}$. Then $\boldsymbol{Z}_{:,j}^T\boldsymbol{Z}_{:,j} = \boldsymbol{z}^T\boldsymbol{z}$. With normalization, for each i, we have,

$$\hat{\boldsymbol{z}}_i = \frac{g_j}{\sigma_j}(\boldsymbol{z}_i - \mu_j),$$

with

$$\mu_j = \frac{1}{n}\sum_i \boldsymbol{z}_i, \quad \sigma_j^2 = \frac{1}{n}\sum_i (\boldsymbol{z}_i - \mu_j)^2.$$

g_j can be a learnable or predefined hyper-parameter. Then with the properties of variance, we have

$$\frac{1}{n}\sum_i \hat{\boldsymbol{z}}_i^2 = \left(\frac{1}{n}\sum_i \hat{\boldsymbol{z}}_i\right)^2 + \frac{1}{n}\sum_i \left(\hat{\boldsymbol{z}}_i - \frac{1}{n}\sum_i \hat{\boldsymbol{z}}_i\right)^2 = g_j^2.$$

Finally, we have $\hat{\boldsymbol{z}}^T\hat{\boldsymbol{z}} = ng_j^2$. For a certain graph, $n = |\mathcal{V}|$ is a fixed value. □

6 Experiments

In this section, we evaluate the effectiveness of DCAT against popular GNN models and state-of-the-art approaches on two representative semi-supervised learning tasks. We further analyze the distribution of attention scores, conduct sensitivity analysis on the hyper-parameters, and perform ablation studies. Additionally, we extend our method to graph Transformer, which is another attention-based GNN, and demonstrate its effectiveness. The code is available online [1]. We design our experiments with reference to [1], [11], [12].

6.1 Experimental Setup

6.1.1 Dataset description

Seven real-world datasets with ground truth labels are used in our experiments. They are `Cora` [49], `Pubmed` [50], `Wiki` [49], `Uai` [49], `Flickr` [51], `CoauthorCS` [52], `Flickr-large`. Detailed descriptions are listed as follows.

`Cora` [49] and `Pubmed` [50] are two widely used citation networks for the evaluation of GNNs, where nodes, edges, and node features represent research articles, article-article citations, and keywords, respectively. There are 2708 nodes, 5429 edges, and 1433 features for each node in `Cora` dataset. Each node belongs to 1 of 7 classes. There are 19717 nodes, 44324 edges, and 500 features for each node in `Pubmed` dataset. Each node belongs to 1 of 3 classes. However, a recent study [52] shows that the classical citation networks, such as `Cora` and `Pubmed`, are insufficient to test the predictive power of GNNs due to the homogeneous network type and limited data size. Hence for a more comprehensive evaluation of different GNN models, we also include `Wiki`, `Uai`, `Flickr`, `CoauthorCS`, and `Flickr-large`, which have different network types, edge density, and feature dimensions compared with the citation graph.

`Wiki` [49] and `Uai` [49] are two networks collected from various Internet services of knowledge sharing, including web blogs and Wikipedia. The nodes, edges, and node features in these datasets represent web pages, web hyperlinks, and descriptive information on these web pages, respectively. `Wiki` has 2405 nodes and 17981 edges. Each node in `Wiki` has 4973 features and belongs to 1 of 17 classes. `Uai` has 3363 nodes and 33300 edges. Each node in `Uai` has 4971 features and belongs to 1 of 19 classes.

`Flickr` [51] is constructed according to a group of active users in an online social networking site. In this dataset, social network users, online social ties among users, and the corresponding user profiles are represented as nodes, edges, and node features, respectively. There are 7575 nodes, 239738 edges, and 12047 features for each node in `Flickr` dataset and each node belongs to 1 of 9 classes.

`CoauthorCS` [52] is a social network, where the nodes denote authors, the edges denote collaborations between authors, and the node features denote the keywords of collaborated papers. `CoauthorCS` has 18333 nodes and 327576 edges. Each node in `CoauthorCS` has 6805 features and belongs to 1 of 7 classes. For brevity, we use `CS` to denote `CoauthorCS` in the following tables and figures.

1. https://anonymous.4open.science/r/DCAT-2B38/

TABLE 1
Dataset statistics

Dataset	Type	Nodes	Edges	Features	Classes	Modularity
Cora	Citation	2,708	5,429	1,433	7	0.807
Pubmed	Citation	19,717	44,324	500	3	0.727
Wiki	Web	2,405	17,981	4,973	17	0.617
Uai	Web	3,363	33,300	4,971	19	0.685
Flickr	Social	7,575	239,738	12,047	9	0.218
CoauthorCS	Collaboration	18,333	327,576	6,805	15	0.665
Flickr-large	Social	89,250	989,006	500	7	0.485

`Flickr-large` [53] dataset is collected by SNAP [2] and processed by [53]. A node represents an image, while an edge connects two images sharing common properties (e.g., user comments). There are 89250 nodes and 989006 edges in this graph, and the node features are 500-dimensional bag-of-words representations of images. This dataset is one of the most popular large graphs used to test the scalability of GNNs [53], [54].

Data statistics are summarized in Table 1. We also compute the modularity of these graphs for further analysis. We build the data splits for `Cora` and `Pubmed` following [1], [11], [12]. We use stratified sampling to generate data splits for the rest. For `Uai` and `Wiki`, we sample 500 nodes as the training set, 500 nodes as the validation set, and 1000 nodes as the test set. For `Flickr` and `CoauthorCS`, we sample 1000 nodes as the training set, 1000 nodes as the validation set, and 2000 nodes as the test set. For `Flickr-large` we sample 2000 nodes as the training set, 2000 nodes as the validation set, and 10000 nodes as the test set.

6.1.2 Learning tasks and evaluation metrics

Following the previous literature [1], [4], [12], we use two representative learning tasks, namely semi-supervised node classification and semi-supervised node clustering, to test the effectiveness of our approach. Semi-supervised node classification aims at classifying the proportion of testing nodes in a graph, where only a very small subset of nodes have labels for training. Semi-supervised node clustering attempts to cluster all the nodes into a given number of clusters, where the cluster labels of a small subset of nodes are available during training. Although these two learning tasks are different, in our experiments, we find that using the similar loss enables DCAT to perform well on both tasks. For both tasks, we use accuracy as the evaluation metric, as ground-truth labels are available.

6.1.3 Pre-processing

Due to the high dimensions of node features in the web and social graphs, many baselines suffer from the unstable learning process and severe overfitting. Therefore, we use a learnable linear layer without activation to reduce the dimension of node features before various GNN models.

6.1.4 Hyper-parameters

The hyper-parameters for semi-supervised node classification are set as follows. The proposed method introduces a new hyper-parameter β to balance the semi-supervised loss and clustering loss. We tune β in the range of $(0, 1)$, and set its value for the six datasets as 0.5, 0.7, 0.1, 0.1, 0.1, 0.6, 0.4, respectively. We will further show in Section 6.5 that the values of β are strongly correlated with the labelling ratio of each dataset. For other hyper-parameters, we follow the settings in previous attention-based graph neural networks [11], [12] and use the same hyper-parameters to train attention-based GNNs for fair comparisons. The details are listed as follows. For `Cora`, `Wiki`, `Uai`, `Flickr`, `CoauthorCS` and `Flickr-large` datasets, we use 1 hidden layer with 8 heads and 1 output layer with 1 head, while for `Pubmed`, we use 8 heads for the output layer as in [11]. The number of hidden dimensions in each head is 8. The linear dimension reduction is applied to `Wiki`, `Uai`, `Flickr`, and `CoauthorCS` datasets, and the dimensions of its output are set as 128, 128, 512, 512 for all the baselines. The training epochs are set to 1000 for `Cora` and `Pubmed`, and 2000 for the others. The learning rate is set as 0.005, and early stopping is also applied. The dropout ratios of the seven datasets are 0.6, 0.6, 0.2, 0.3, 0.6, 0.6, 0.6 respectively. The weight decay is 0.0005 on each dataset. We find that the hyper-parameters set for semi-supervised node classification also work well for semi-supervised node clustering, and hence use the same for the latter.

6.1.5 Baselines

We compare DCAT with several popular or state-of-the-art GNNs, including GCN [1], GraphSage [2], APPNP [4], JKNet [55] and MinCutPool [26]. As MinCutPool is an unsupervised method, we add an additional semi-supervised loss term for it in our experiments. To demonstrate the improvement of DCAT over those closely related models, we also include more attention-based GNNs, i.e., GAT [11], GATv2 [32], CAT [12], HardGAT [30], MAGNA [13] and GAT-k-Lap. GAT [11] is the basic model for attention-based graph neural networks. CAT [12] incorporates structural node embedding into attention by averaging the feature based attention and graph structural embedding based attention. HardGAT [30] selects the most important nodes (top k) for the query node when computing attention scores, and ignores unimportant nodes. MAGNA [13] computes attention scores for node pairs within k hops, which provides a much larger receptive field for each node in every attention layer. GAT-k-Lap concatenates the node features with Laplacian EigenMap [56] as the input for GNN models. [15] has $O(n^3)$ complexity where n is the number of nodes in a graph. As this is intractable for large graphs, we did not include it here. The codes for baselines are either from DGL [57] or from their official repositories.

6.1.6 Settings for baselines

Here we show the details of the implementation of all the approaches. In general, we use the example code (https://github.com/dmlc/dgl/tree/master/examples/pytorch) of Deep Graph Library (DGL) [57], except GCN from https://github.com/tkipf/pygcn and https://github.com/tkipf/gcn, MAGNA from https://github.com/xjtuwgt/GNN-MAGNA and CAT from https://github.com/he-tiantian/CATs. For fair comparisons, we set the neural network architecture of those approaches to be the same or similar, and meanwhile also follow the recommended settings from their papers.

2. https://snap.stanford.edu/data/web-flickr.html

TABLE 2
Comparing DCAT with **attention-based GNNs** in semi-supervised classification. The results show the accuracy (%), with the best results in bold and the best baselines underlined.

Method	Cora	Pubmed	Wiki	Uai	Flickr	CoauthorCS	Flickr-Large
GAT	82.85 ± 0.97	81.27 ± 0.32	70.03 ± 0.80	59.81 ± 0.66	46.32 ± 1.40	89.49 ± 0.20	47.54 ± 0.58
GATv2	83.43 ± 0.60	<u>82.41</u> ± 0.47	71.03 ± 0.66	<u>61.31</u> ± 0.80	<u>49.25</u> ± 3.14	91.70 ± 0.25	<u>49.51</u> ± 0.86
CAT	<u>83.56</u> ± 0.99	82.10 ± 0.42	<u>71.19</u> ± 0.60	61.26 ± 0.75	47.16 ± 1.89	<u>92.08</u> ± 0.28	49.23 ± 0.53
HardGAT	79.26 ± 0.39	81.48 ± 0.45	67.36 ± 0.73	55.04 ± 0.87	37.77 ± 1.34	89.84 ± 0.44	48.91 ± 1.09
MAGNA	77.26 ± 0.78	80.89 ± 0.87	66.07 ± 1.01	56.98 ± 1.56	48.30 ± 4.14	90.95 ± 0.41	49.22 ± 0.39
GAT-k-lap	82.06 ± 0.83	80.15 ± 0.35	66.66 ± 0.73	58.50 ± 0.93	43.25 ± 1.05	91.25 ± 0.19	48.91 ± 1.00
DCAT	**84.17** ± 0.33	**82.90** ± 0.34	**72.20** ± 0.88	**64.75** ± 0.86	**52.27** ± 0.93	**92.54** ± 0.29	**50.01** ± 0.28

General settings. For all the models, we include a linear layer before graph neural network layers on `Wiki`, `Uai`, `Flickr`, and `CoauthorCS` datasets, while the dimension of the linear layer is set as 128, 128, 512, and 512 on those four datasets respectively. In general, we use two graph neural network layers for each model for fair comparisons. The train epochs for `Cora` and `Pubmed` are set as 1000 as recommended by most of the baselines and set as 2000 on other datasets according to our preliminary experiments. For all the models, we tune the learning rate in {0.01, 0.005, 0.001} and choose the best results if there are no further descriptions. Meanwhile, early stopping is also applied for each model.

Attention-based GNNs. We generally use the same settings for all the attention-based GNNs in our experiments, i.e., GAT [11], GATv2 [58], CAT [12], HardGAT [30], MAGNA [13], GAT-k-Lap and our proposed DCAT, with the following exceptions. Here we list those different settings compared with GAT and DCAT. For CAT, we use the default settings of α, β, γ (i.e., $\alpha = 0.2, \beta = 1.0, \gamma = 0.01$). For HardGAT, we set $k = 8$. For MAGNA, we tune the number of hops in {2, 3, 4, 5} and choose the best results for each dataset. For GAT-k-Lap, we set k as the number of labels in each dataset.

Non-attention-based GNNs. For GCN [1], following the recommended settings, we use 2 graph layers with the hidden neuron as 16, and set the dropout ratio as 0.6. On `Cora` and `Pubmed`, we use the default learning rate 0.01 as in the GCN paper, while on other datasets we tune the learning rate in {0.01, 0.005, 0.001}. For GraphSage [2], we use 2 graph layers with the hidden neuron as 32, and set the dropout ratio as 0.5. For APPNP [4], we set the hidden neuron as 64, and the dropout ratio as 0.5. We tune the hyper-parameter k in {2, 3, 4, 5, 6, 7, 8, 9, 10}. For JKNet [55], we use 4 graph layers. The aggregation mode is set as "concatenation". We tune the hidden neuron in {16, 32, 64}. For MinCutPool [26], we tune the hidden neuron in {16, 32, 64}, and also tune the weights of its original loss and the additional semi-supervised loss. We present the best results in our paper.

6.2 Performance on Semi-supervised Node Classification

We first compare our proposed method with the baselines on the semi-supervised node classification task, which is the major testbed showing the predictive power of GNNs. The results are summarized in Table 2 and Table 3. We report the mean accuracy and its standard deviation of 10 runs. We first compare our proposed DCAT with other attention-based GNNs in Table 2 and show the improvement of DCAT. Specifically, DCAT improves the graph attention networks (GAT) by a margin of 1.59%, 2.01%, 3.10%, 8.26%, 12.85%, 3.37% and 5.20% on the 7 datasets respectively. Meanwhile, DCAT also outperforms the best attention-based GNNs by a margin of 0.73%, 0.59%, 1.42%, 5.61%, 6.13%, 0.47% and 1.01%. This empirically demonstrates that the clustering information learned from the clustering loss is beneficial to the representation and downstream tasks.

We then analyze the performance gap between other attention-based methods and DCAT. We notice that GAT-k-Lap sometimes performs worse than GAT, and the reason could be: (1) the node features and node embedding are in different spaces, and should not be concatenated directly; (2) more parameters result in worse overfitting. The results also suggest that an effective method will boost the impact of node clusters (or other forms of graph structure) in GAT. In DCAT, we incorporate node clusters into each hidden attention layer, so the cluster information will not decay over layers. After the training, the clusters are learned by the model. Given node features as the input, the output representation can effectively combine the features and clusters through the attention layers. DCAT outperforms CAT, the most recent work using structural information to improve GAT. DCAT also produces better results with the same hyper-parameters, without introducing additional parameters. This further demonstrates the superiority of DCAT in utilizing cluster information to improve GAT.

We also compare DCAT with other GNNs which do not adopt attention mechanism on graphs, and the results are summarized in Table 3. We noticed that DCAT also outperforms all the baselines by a large margin.

To verify the necessity of modelling clustering information, the modularity of each dataset is computed and listed in Table 1. We compute the Pearson correlation coefficient ρ between the accuracy of DCAT and modularity. The value of ρ is 0.764, which shows that DCAT performs better on graphs with larger modularity.

6.3 Performance on Semi-supervised Node Clustering

The experimental results on semi-supervised node clustering are summarized in Table 4 and Table 5. We report the mean accuracy and its standard deviation of 10 runs. We first compare DCAT with other attention-based GNNs in Table 4. We observe that DCAT improves GAT by a margin

TABLE 3
Comparing DCAT with **non**-attention-based GNNs in semi-supervised classification. The results show the accuracy (%), with the best results in bold and the best baselines underlined. "oom" means "out-of-memory".

Method	Cora	Pubmed	Wiki	Uai	Flickr	CoauthorCS	Flickr-Large
GCN	81.81 ± 0.83	80.24 ± 0.71	69.14 ± 0.70	59.00 ± 0.75	50.52 ± 2.42	90.96 ± 0.34	47.89 ± 1.08
GraphSage	82.83 ± 0.75	81.74 ± 0.40	70.55 ± 0.99	62.28 ± 0.47	48.45 ± 0.47	91.94 ± 0.21	48.57 ± 0.37
APPNP	83.67 ± 0.66	82.59 ± 0.46	67.69 ± 1.91	58.86 ± 1.49	46.58 ± 0.54	91.49 ± 0.32	49.06 ± 0.37
JKNet	81.02 ± 0.99	80.58 ± 0.64	69.36 ± 0.48	60.40 ± 0.94	50.20 ± 3.35	90.97 ± 0.40	49.19 ± 0.70
MinCutPool	71.62 ± 1.39	75.29 ± 0.62	60.12 ± 1.36	56.24 ± 1.18	50.76 ± 2.32	91.14 ± 1.09	oom
DCAT	**84.17** ± 0.33	**82.90** ± 0.34	**72.20** ± 0.88	**64.75** ± 0.86	**52.27** ± 0.93	**92.54** ± 0.29	**50.01** ± 0.28

TABLE 4
Comparing DCAT with **attention-based GNNs** in semi-supervised clustering. The results show the accuracy (%), with the best results in bold and the best baselines underlined.

Method	Cora	Pubmed	Wiki	Uai	Flickr	CoauthorCS	Flickr-Large
GAT	80.04 ± 0.77	80.39 ± 0.24	75.83 ± 0.98	61.74 ± 0.78	51.91 ± 1.30	89.23 ± 0.17	48.68 ± 0.95
GATv2	80.47 ± 0.55	81.74 ± 0.34	76.52 ± 0.48	63.09 ± 0.55	53.51 ± 2.88	92.11 ± 0.14	49.44 ± 0.96
CAT	80.68 ± 0.70	81.82 ± 0.27	75.36 ± 0.45	62.58 ± 0.42	53.21 ± 1.13	92.28 ± 0.10	49.46 ± 0.55
HardGAT	76.37 ± 0.41	80.58 ± 0.32	71.14 ± 0.69	55.74 ± 0.58	39.19 ± 1.30	90.06 ± 0.22	48.86 ± 0.92
MAGNA	75.37 ± 0.62	79.84 ± 0.65	71.00 ± 0.90	58.59 ± 0.93	53.92 ± 3.94	91.18 ± 0.33	49.25 ± 0.29
GAT-k-lap	78.75 ± 0.75	78.11 ± 0.46	72.32 ± 0.87	57.99 ± 0.86	50.64 ± 0.85	91.47 ± 0.09	49.07 ± 0.84
DCAT	**81.99** ± 0.51	**81.92** ± 0.22	**77.48** ± 0.67	**65.00** ± 0.77	**56.67** ± 0.84	**92.57** ± 0.13	**49.91** ± 0.25

TABLE 5
Comparing DCAT with **non**-attention-based GNNs in semi-supervised clustering. The results show the accuracy (%), with the best results in bold and the best baselines underlined. "oom" means "out-of-memory".

Method	Cora	Pubmed	Wiki	Uai	Flickr	CoauthorCS	Flickr-Large
GCN	79.51 ± 0.79	79.64 ± 0.58	74.40 ± 0.95	60.41 ± 0.62	53.51 ± 2.49	90.96 ± 0.38	48.01 ± 0.96
GraphSage	78.98 ± 0.59	81.34 ± 0.26	73.52 ± 0.86	62.46 ± 0.24	53.09 ± 0.43	92.38 ± 0.16	49.53 ± 0.33
APPNP	81.55 ± 0.63	82.85 ± 0.21	69.92 ± 1.87	58.29 ± 0.82	47.88 ± 0.51	91.51 ± 0.31	49.13 ± 0.57
JKNet	79.20 ± 0.87	80.43 ± 0.39	74.74 ± 0.48	61.63 ± 1.05	54.45 ± 3.24	90.91 ± 0.23	49.56 ± 0.68
MinCutPool	71.49 ± 1.35	75.56 ± 0.67	64.68 ± 1.06	58.04 ± 1.40	53.99 ± 2.73	91.59 ± 0.95	oom
DCAT	**81.99** ± 0.51	81.92 ± 0.22	**77.48** ± 0.67	**65.00** ± 0.77	**56.67** ± 0.84	**92.57** ± 0.13	**49.91** ± 0.25

of 2.44%, 1.90%, 2.18%, 5.28%, 9.17%, 3.63%, and 2.53%. Meanwhile, DCAT also performs on par or better than the best attention-based GNNs. The results in semi-supervised clustering are similar to the results in semi-supervised classification, which empirically demonstrates that learning from graph clusters is beneficial to graph attention.

We then compare DCAT with other GNNs that do not adopt attention mechanism on graph. In terms of accuracy, our proposed DCAT outperforms the best baselines on five of the six datasets. The only exception is on the PubMed dataset, where APPNP performs slightly better. As APPNP has much fewer parameters than attention-based GNNs, it tends to perform better when the training set is very small. (Recall that only 0.3% of nodes are in the training set of the PubMed dataset.)

6.4 Analysis of Attention Scores

Here we analyze the distributions of learned attention scores. We show that the DCAT can learn smaller attention scores for node pairs between clusters than those of GAT while learn larger attention scores for node pairs within each cluster. These observations are consistent with our demonstrations in Fig. 1 and our analysis in Section 4.7. We show the histograms of three representative datasets (`Cora`, `Uai` and `Flickr`) in the main manuscript (Fig. 3, Fig. 4 and Fig. 5) and put other histograms in supplementary materials due to page limit. We take Fig. 3 as an example to illustrate the axes of each subfigure. The histogram of attention scores from GAT on `Cora` dataset is shown in Fig. 3 (a) and Fig. 3 (b), while the histogram of attention scores from DCAT is shown in Fig. 3 (c) and Fig. 3 (d). We use the same ticks for GAT and DCAT for comparison. However, we notice that the range of y-axis is very different on smaller attention scores and larger attention scores, so we split the histograms for GAT and DCAT at the same x value (e.g., 0.4 for `cora`) and use different ticks for y-axis.

We observe the histograms on 5 out of 6 datasets show the expected difference between DCAT and GAT. We will use `Cora` dataset to illustrate the comparison. In Fig. 3 we have two observations: (1) DCAT has much more large attention scores within each cluster (in blue color); (2) for small attention scores (e.g., attention scores less than 0.05), the values of $\frac{\text{area of red}}{\text{area of blue}}$ is larger in DCAT, which indicates that a small attention score is more likely to appear on edges between clusters. The only exception is in `Pubmed` dataset, where the histograms of DCAT are very similar to those of GAT. The reason is that the clustering loss does not change too much, and the difference is not significant.

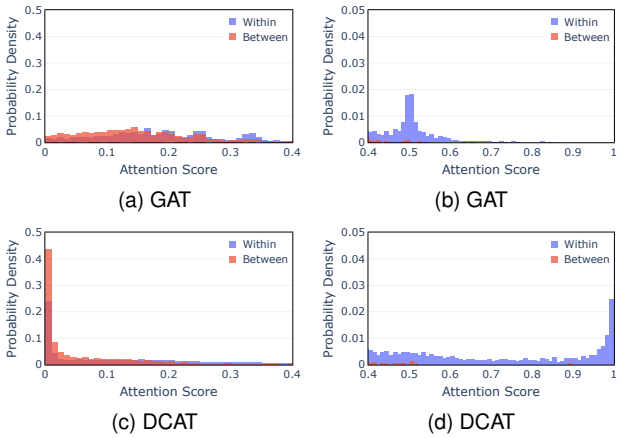


Fig. 3. Histogram of attention scores on Cora.

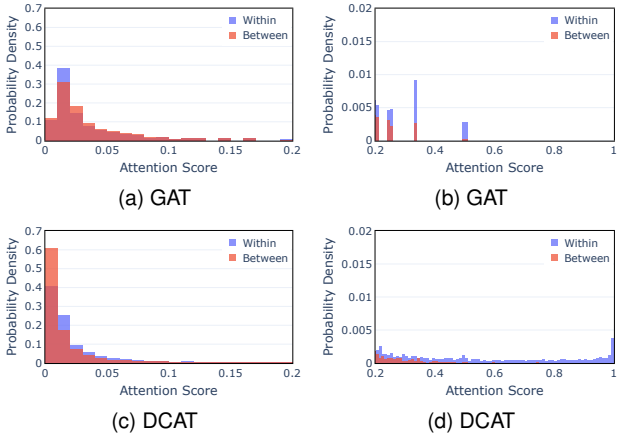


Fig. 4. Histogram of attention scores on Uai.

6.5 Sensitivity Test on Hyper-parameter β

The proposed model introduces a new hyper-parameter $\beta \in (0, 1)$ in the objective

$$\mathcal{L} = (1 - \beta)\mathcal{L}_{\text{semi}} + \beta\mathcal{L}_c$$

to balance the loss of semi-supervised objective and clustering objective. Here we investigate the hyper-parameter sensitivity of our model. In Fig. 6, the accuracy of semi-supervised classification and semi-supervised clustering tasks on the six datasets are displayed. In the figure, we observe that the performance trends of the proposed model according to β are very similar in the two tasks. We also observe that on each dataset, there is a range of β where the model consistently performs very well. Within the range, the change of β does not affect the performance significantly. Specifically, DCAT performs well for $\beta \in [0.5, 0.7]$ on the `Cora` dataset and for $\beta \in [0.1, 0.3]$ on the `Uai` dataset.

While the performance of the proposed model is stable on each dataset, we should mention that different datasets do not share the same setting of β. Recall that the effect of β is to balance the loss between the semi-supervised objective and the clustering objective. For different datasets, the importance of the two objectives is different, and the best values of β also vary. One factor that may influence the value of β is the labelling ratio (i.e., the number of nodes in the training set / the number of total nodes). We compute the Pearson correlation coefficient ρ between the labelling ratio and β, and its value is -0.887, which shows that β is strongly correlated to the labelling ratio. Thus if we have few labelled data for training (e.g., on the `Uai` dataset), we should choose a relatively large β, and vice versa (e.g., on the `Cora` dataset, where we only have around 5% of the total nodes in the training set). A possible reason is that, when there are few labelled data, the semi-supervised loss would cause severe overfitting. However, graph clusters, which may provide pseudo-labels for nodes, will relieve the overfitting and improve the generalization performance.

Fig. 5. Histogram of attention scores on Flickr.

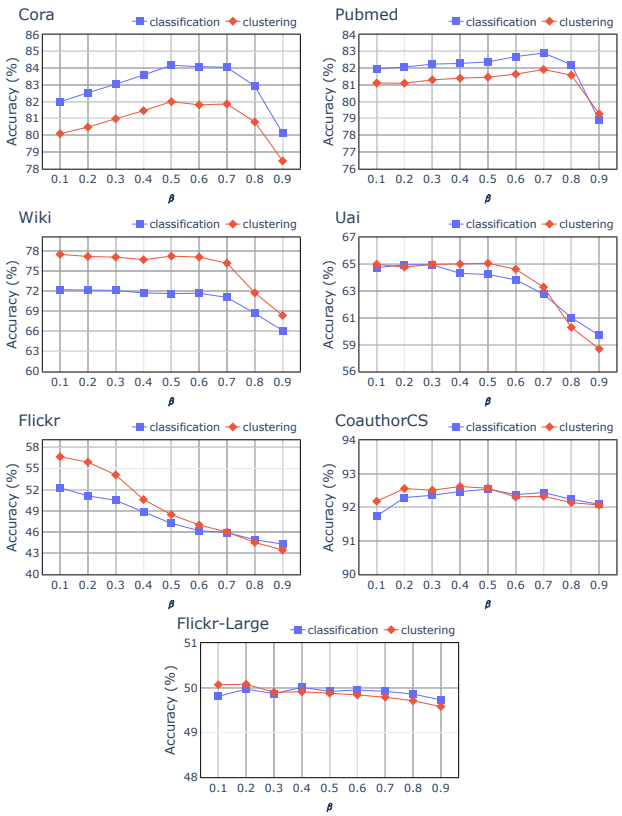


Fig. 6. Sensitivity analysis of hyper-parameter β.

TABLE 6
Ablation study on the components of DCAT. The results show the accuracy (%).

Semi-supervised node classification										
Model	**Components**			**Cora**	**Pubmed**	**Wiki**	**Uai**	**Flickr**	**CS**	**Flickr-large**
	$\mathcal{L}_{tr}$	norm	$\mathcal{L}_o$							
GAT				82.85 ± 0.97	81.27 ± 0.32	70.03 ± 0.80	59.81 ± 0.66	46.32 ± 1.40	89.49 ± 0.20	47.54 ± 0.58
Lap Smooth	√			83.30 ± 0.55	81.13 ± 0.30	71.04 ± 0.70	59.50 ± 0.98	47.70 ± 1.62	91.12 ± 0.30	48.75 ± 0.80
$\mathcal{L}_{tr}$ + ortho	√		√	82.51 ± 0.77	82.11 ± 0.45	71.00 ± 0.89	60.50 ± 0.72	49.14 ± 1.06	92.32 ± 0.19	49.15 ± 0.46
$\mathcal{L}_{tr}$ + norm	√	√		82.25 ± 0.89	81.44 ± 0.50	71.97 ± 0.74	64.67 ± 0.85	52.24 ± 1.14	91.61 ± 0.52	48.69 ± 0.19
DCAT	√	√	√	84.17 ± 0.33	82.90 ± 0.34	72.20 ± 0.88	64.75 ± 0.86	52.27 ± 1.05	92.54 ± 0.29	50.01 ± 0.28
Semi-supervised node clustering										
Model	**Components**			**Cora**	**Pubmed**	**Wiki**	**Uai**	**Flickr**	**CS**	**Flickr-large**
	$\mathcal{L}_{tr}$	norm	$\mathcal{L}_o$							
GAT				80.04 ± 0.77	80.39 ± 0.24	75.83 ± 0.98	61.74 ± 0.78	51.91 ± 1.30	89.23 ± 0.17	48.68 ± 0.95
Lap Smooth	√			80.92 ± 0.37	80.57 ± 0.20	76.23 ± 0.61	61.39 ± 0.60	52.71 ± 1.21	91.20 ± 0.21	48.87 ± 0.54
$\mathcal{L}_{tr}$ + ortho	√		√	79.93 ± 0.50	81.45 ± 0.20	76.51 ± 0.47	62.04 ± 0.62	53.69 ± 0.88	92.28 ± 0.11	49.28 ± 0.19
$\mathcal{L}_{tr}$ + norm	√	√		80.40 ± 0.76	80.73 ± 0.41	77.33 ± 0.60	64.81 ± 0.85	56.10 ± 1.11	92.04 ± 0.45	49.24 ± 0.23
DCAT	√	√	√	81.99 ± 0.51	81.92 ± 0.22	77.48 ± 0.67	65.00 ± 0.77	56.67 ± 1.15	92.57 ± 0.13	49.91 ± 0.25

6.6 Ablation Studies

We conduct ablation studies on the components of DCAT. There are three major components in DCAT: $\mathcal{L}_{tr}$, normalization, and $\mathcal{L}_o$, which are defined in Eq. (9). Accordingly, we consider the following models: GAT, GAT + $\mathcal{L}_{tr}$, GAT + $\mathcal{L}_{tr}$ + $\mathcal{L}_o$ (orthogonal loss), and GAT +$\mathcal{L}_{tr}$+ normalization. The experimental results are given in Table 6.

The Laplacian smoothing term is defined as $\mathcal{L}_{tr} = \mathrm{Tr}\left(\boldsymbol{Z}^T\boldsymbol{L}\boldsymbol{Z}\right)$, where $\boldsymbol{Z}$ is the representation of nodes and $\boldsymbol{L}$ is the graph Laplacian matrix. $\mathcal{L}_{tr}$ can be regarded as part of our optimization objective. However, without any constraint on $\boldsymbol{Z}$, optimizing $\mathcal{L}_{tr}$ directly can produce undesirable results. For example, even though reducing the length of $\boldsymbol{Z}$ consequently reduces the value of $\mathcal{L}_{tr}$, the result is trivial. As can be seen from Table 6, although Laplacian Smoothing can enhance the performance of GAT, the gain is much smaller compared with that achieved by our model DCAT.

Another ablation baseline is $\mathcal{L}_{tr} + \mathcal{L}_o$, where $\mathcal{L}_o$ is defined in Eq. (8). Here we rewrite the formula for convenient reference.

$$\begin{aligned}\mathcal{L}_o &= \left\|\boldsymbol{Z}^T\boldsymbol{Z} - n\boldsymbol{I}\right\|_F^2 \\ &= \sum_{i=j}\left\|\boldsymbol{Z}_{:,i}^T\boldsymbol{Z}_{:,j} - n\right\|_2^2 + \sum_{i\neq j}\left\|\boldsymbol{Z}_{:,i}^T\boldsymbol{Z}_{:,j} - 0\right\|_2^2\end{aligned}$$

Without normalization, the first term is not strictly zero, and thus the length of the columns of Z is not strictly of the same fixed value. Therefore the clustering information is not well preserved compared with DCAT, where normalization is applied.

Finally, we compare DCAT with $\mathcal{L}_{tr}$+ normalization. This comparison shows the effect of orthogonal loss. From the probabilistic perspective, the orthogonal loss penalizes the covariance of the distribution of hidden representation to the identity matrix. Thus, if the covariance is close to the identity matrix without orthogonal loss, the gap between DCAT and $\mathcal{L}_{tr}$+ normalization is small, and vice versa. However, it is intractable to estimate the covariance before training. Meanwhile, from the experimental results in Table 6, we observe that the orthogonal loss brings non-trivial performance gain on most of the datasets. Therefore the orthogonal loss is necessary for the proposed method.

6.7 Extension to Graph Transformer

The proposed method can also enhance the predictive performance of graph Transformer in semi-supervised learning. Recently, there has been a trend to adopt Transformers on graph data. However, most of them [14], [31], [36], [59] are designed for graph level tasks where the graphs are usually very small (e.g., small molecular graphs), and they have $O(n^3)$ complexity, which is intractable for node-level tasks on large graphs. Here, we implement a graph Transformer (GT) based on [60]. For each layer in GT, the representation can be learned as follows:

$$\begin{aligned}&\boldsymbol{Q} = \boldsymbol{H}\boldsymbol{W}_{\boldsymbol{Q}}^T,\ \boldsymbol{K} = \boldsymbol{H}\boldsymbol{W}_{\boldsymbol{K}}^T,\ \boldsymbol{V} = \boldsymbol{H}\boldsymbol{W}_{\boldsymbol{V}}^T,\\ &\boldsymbol{Z} = \mathrm{Softmax}(\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{d}\odot\tilde{\boldsymbol{A}})\boldsymbol{V},\\ &\boldsymbol{H}' = \mathrm{LayerNorm}(\sigma(\boldsymbol{Z})\cdot\boldsymbol{W}^T + \boldsymbol{H}),\end{aligned}$$

where $\tilde{\boldsymbol{A}} := \boldsymbol{A} + \boldsymbol{I}$ is a mask to consider attention only between connected node pairs. To compute clustering loss for graph Transformer, we use $\boldsymbol{Z}$ to parameterize the clustering loss as follows,

$$\mathcal{L}_c = \frac{1}{n}\mathrm{Tr}\left(\boldsymbol{Z}^T\boldsymbol{L}\boldsymbol{Z}\right) + \left\|\frac{1}{n}\boldsymbol{Z}^T\boldsymbol{Z} - \boldsymbol{I}\right\|_F^2.$$

As graph Transformer already has a normalization layer, we do not add the normalization in DCAT. We call the extension of DCAT on graph Transformer as Differentiable Clustering for Graph Transformer (DC-GT).

We run graph Transformers on all the testing datasets. The detailed experimental settings are listed as follows. The value of β for the six datasets are 0.6, 0.6, 0.2, 0.01, 0.2, and 0.1. The other hyper-parameters are the same for the six datasets. We use 8 graph transformer layers and 4 attention heads in each layer, with the dimension for each head as 64. All the models are trained with 2000 epochs with early stop, and the dropout ratio is set as 0.4. The experimental results of DC-GT and GT can be found in Table 7.

It is observed that the performance on semi-supervised node classification and clustering is improved on all the testing datasets when graph clusters are considered by graph Transformer. Specifically, Differentiable clustering for graph Transformer model achieves 5.09%, 2.80%, 1.98%, 2.68%, 0.47%, and 1.09% performance gain on the six datasets in

semi-supervised classification compared with graph Transformer, and achieves 6.47%, 3.74%, 1.48%, 0.86%, 0.35%, and 1.20% performance gain in semi-supervised clustering. The results indicate that our approach is very effective in enhancing the performance of various types of attention-based GNNs in semi-supervised learning tasks.

TABLE 7
The results of semi-supervised classification and semi-supervised clustering in terms of accuracy (%), with the best result in bold.

Task	Method	Cora	Pubmed	Wiki	Uai	Flickr	CS
Classification	GT	72.7	78.5	75.9	67.2	84.4	91.6
	DC-GT	**76.4**	**80.7**	**77.4**	**69.0**	**84.8**	**92.6**
Clustering	GT	71.1	77.6	81.2	69.8	86.6	91.9
	DC-GT	**75.7**	**80.5**	**82.4**	**70.4**	**86.9**	**93.0**

7 Conclusions

In this paper, we present Differentiable Clustering for Graph Attention (DCAT), which incorporates graph clustering into graph attention networks for semi-supervised learning on graph data. By parameterizing the continuous relaxed objective of modularity maximization with graph attention layers, DCAT can jointly learn with a semi-supervised loss and the objective of clustering. We further propose a novel solution to reformulate the orthonormal constraint introduced by the clustering objective to a new formulation, which is much easier for the network to learn than the original constrained optimization problem. We conduct empirical experiments to demonstrate the effectiveness of the proposed model. Several promising extensions can be considered for future work. For example, it would be interesting to consider nodes belonging to multiple communities or across the community boundary. Another potential direction is to investigate using both clusters and other graph structures to enhance the performance of graph attention networks.

Acknowledgments

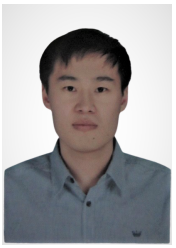
This study is supported under the RIE2020 Industry Alignment Fund – Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contribution from Singapore Telecommunications Limited (Singtel), through Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU).

References

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.
[2] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.
[3] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 249–270, 2022.
[4] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *ICLR*, 2019.
[5] B. Zheng, H. Wen, Y. Liang, N. Duan, W. Che, D. Jiang, M. Zhou, and T. Liu, "Document modeling with graph attention networks for multi-grained machine reading comprehension," in *ACL*, 2020, pp. 6708–6718.
[6] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *ICML*, vol. 70, 2017, pp. 1263–1272.
[7] Y. Wu, N. Choma, A. D. Chen, M. Cashman, É. T. Prates, V. G. M. Vergara, M. Shah, A. Clyde, T. S. Brettin, W. A. de Jong, N. Kumar, M. S. Head, R. L. Stevens, P. Nugent, D. A. Jacobson, and J. B. Brown, "Spatial graph attention and curiosity-driven policy for antiviral drug discovery," in *ICLR*, 2022.
[8] K. Wang, W. Shen, Y. Yang, X. Quan, and R. Wang, "Relational graph attention network for aspect-based sentiment analysis," in *ACL*, 2020, pp. 3229–3238.
[9] W. Fan, Y. Ma, Q. Li, J. Wang, G. Cai, J. Tang, and D. Yin, "A graph neural network framework for social recommendations," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 5, pp. 2033–2047, 2022.
[10] J. Yu, H. Yin, J. Li, M. Gao, Z. Huang, and L. Cui, "Enhancing social recommendation with adversarial graph convolutional networks," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 8, pp. 3727–3739, 2022.
[11] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.
[12] T. He, Y.-S. Ong, and L. Bai, "Learning conjoint attentions for graph neural nets," in *NeurIPS*, 2021.
[13] G. Wang, R. Ying, J. Huang, and J. Leskovec, "Multi-hop attention graph neural networks," in *IJCAI*, 2021, pp. 3089–3096.
[14] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform bad for graph representation?" in *NeurIPS*, 2021.
[15] K. Zhang, Y. Zhu, J. Wang, and J. Zhang, "Adaptive structural fingerprints for graph attention networks," in *ICLR*, 2020.
[16] M. E. Newman, "Fast algorithm for detecting community structure in networks," *Physical review E*, vol. 69, no. 6, p. 066133, 2004.
[17] ——, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.
[18] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *AAAI*, 2017, pp. 203–209.
[19] E. Min, R. Chen, Y. Bian, T. Xu, K. Zhao, W. Huang, P. Zhao, J. Huang, S. Ananiadou, and Y. Rong, "Transformer for graphs: An overview from architecture perspective," *arXiv preprint arXiv:2202.08455*, 2022.
[20] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *J. Mach. Learn. Res.*, vol. 18, pp. 1–43, 2018.
[21] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS 2017 Workshop Autodiff*, 2017.
[22] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, 2000.
[23] U. von Luxburg, "A tutorial on spectral clustering," *Stat. Comput.*, vol. 17, no. 4, pp. 395–416, 2007.
[24] A. Tsitsulin, J. Palowitch, B. Perozzi, and E. Müller, "Graph clustering with graph neural networks," *arXiv preprint arXiv:2006.16904*, 2020.
[25] S. Zhang, Z. Huang, H. Zhou, and Z. Zhou, "SCE: scalable network embedding from sparsest cut," in *SIGKDD*, 2020, pp. 257–265.
[26] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *ICML*, 2020, pp. 874–883.
[27] M. L. Casado and D. Martínez-Rubio, "Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group," in *ICML*, 2019.
[28] Z. Wen and W. Yin, "A feasible method for optimization with orthogonality constraints," *Mathematical Programming*, vol. 142, no. 1, pp. 397–434, 2013.
[29] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D. Yeung, "Gaan: Gated attention networks for learning on large and spatiotemporal graphs," in *UAI*, 2018, pp. 339–349.
[30] H. Gao and S. Ji, "Graph representation learning via hard and channel-wise attention networks," in *SIGKDD*, 2019, pp. 741–749.
[31] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *arXiv preprint arXiv:2012.09699*, 2020.
[32] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" in *ICLR*, 2022.
[33] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *ICLR*, 2015.
[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017, pp. 5998–6008.

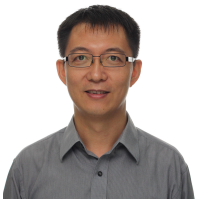
[35] D. Chen, L. O'Bray, and K. Borgwardt, "Structure-aware transformer for graph representation learning," in *ICML*, 2022, pp. 3469–3489.
[36] M. S. Hussain, M. J. Zaki, and D. Subramanian, "Edge-augmented graph transformers: Global self-attention is enough for graphs," *arXiv preprint arXiv:2108.03348*, 2021.
[37] W. Kim, A. Kanezaki, and M. Tanaka, "Unsupervised learning of image segmentation based on differentiable feature clustering," *IEEE Trans. Image Process.*, vol. 29, pp. 8055–8068, 2020.
[38] X. Wang, M. Zhu, D. Bo, P. Cui, C. Shi, and J. Pei, "AM-GCN: adaptive multi-channel graph convolutional networks," in *SIGKDD*, 2020, pp. 1243–1253.
[39] H. Sun, F. He, J. Huang, Y. Sun, Y. Li, C. Wang, L. He, Z. Sun, and X. Jia, "Network embedding for community detection in attributed networks," *ACM Trans. Knowl. Discov. Data*, vol. 14, no. 3, pp. 36:1–36:25, 2020.
[40] X. Zhang and M. E. Newman, "Multiway spectral community detection in networks," *Physical Review E*, vol. 92, no. 5, p. 052808, 2015.
[41] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner, "On finding graph clusterings with maximum modularity," in *International Workshop on Graph-Theoretic Concepts in Computer Science*. Springer, 2007, pp. 121–132.
[42] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.
[43] M. E. Newman, "Equivalence between modularity optimization and maximum likelihood methods for community detection," *Physical Review E*, vol. 94, no. 5, p. 052315, 2016.
[44] S. White and P. Smyth, "A spectral clustering approach to finding communities in graphs," in *ICDM*, 2005, pp. 274–285.
[45] M. E. Newman, "Spectral methods for community detection and graph partitioning," *Physical Review E*, vol. 88, no. 4, p. 042822, 2013.
[46] E. Abbe, "Community detection and stochastic block models: recent developments," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6446–6531, 2017.
[47] H. Lütkepohl, *Handbook of Matrices*. Chichester: Wiley., 1997.
[48] C. H. Q. Ding, X. He, and H. D. Simon, "Nonnegative lagrangian relaxation of *K*-means and spectral clustering," in *ECML*, vol. 3720, 2005, pp. 530–538.
[49] Q. Lu and L. Getoor, "Link-based classification," in *ICML*, 2003, pp. 496–503.
[50] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Mag.*, vol. 29, no. 3, pp. 93–106, 2008.
[51] X. Huang, J. Li, and X. Hu, "Label informed attributed network embedding," in *WSDM*, 2017, pp. 731–739.
[52] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.
[53] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. K. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *ICLR*, 2020.
[54] M. Fey, J. E. Lenssen, F. Weichert, and J. Leskovec, "Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings," in *ICML*, vol. 139, 2021, pp. 3294–3304.
[55] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *ICML*, 2018, pp. 5449–5458.
[56] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Comput.*, vol. 15, no. 6, pp. 1373–1396, 2003.
[57] M. Y. Wang, "Deep graph library: Towards efficient and scalable deep learning on graphs," in *ICLR workshop on representation learning on graphs and manifolds*, 2019.
[58] A. Karaaslanli and S. Aviyente, "Simultaneous graph signal clustering and graph learning," in *ICML*, vol. 162, 2022, pp. 10 762–10 772.
[59] G. Mialon, D. Chen, M. Selosse, and J. Mairal, "Graphit: Encoding graph structure in transformers," *arXiv preprint arXiv:2106.05667*, 2021.
[60] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *WWW*, 2020, pp. 2704–2710.

Haicang Zhou received the BSc degree from Fudan University. He is currently working toward the PhD degree in the School of Computer Science and Engineering, Nanyang Technological University. His current research interests include machine learning on graphs and data mining.



Tiantian He (*Member, IEEE*) received the Ph.D. degree in Computer Science from the Hong Kong Polytechnic University in 2017. Currently He is a Research Scientist in Center for Frontier AI Research (CFAR), Institute of High Performance Computing (IHPC), Singapore Institute of Manufacturing Technology (SIMTech), Agency for Science, Technology and Research (A*STAR). His research interests include artificial intelligence, machine learning, data-centric transfer optimization, and data mining.

Yew Soon Ong (*Fellow, IEEE*) received the PhD degree in artificial intelligence in complex design from the University of Southampton, U.K., in 2003. He is currently a president's chair professor in computer science with Nanyang Technological University (NTU), and is the chief artificial intelligence scientist of the Agency for Science, Technology and Research in Singapore. At NTU, he also serves as director of the Singtel-NTU Cognitive & Artificial Intelligence Joint Lab, and director of the Data Science and Artificial Intelligence Research Center. He was chair of the School of Computer Science and Engineering, NTU from 2016-2018. His research interests include artificial and computational intelligence, presently in machine learning, evolution computation, and optimization. He is the EIC of the IEEE Transactions on Emerging Topics in Computational Intelligence and AE of the IEEE Transactions on Neural Networks and Learning Systems, IEEE Cybernetics, IEEE Transactions on Evolutionary Computation, IEEE Transactions on Artificial Intelligence and others. He has received several IEEE outstanding paper awards, Nanyang Education Excellence Award and was listed as a Thomson Reuters highly cited researcher and among the World's Most Influential Scientific Minds.



Gao Cong is a professor with the School of Computer Science and Engineering, Nanyang Technological University (NTU). He is the director of Singtel Cognitive and Artificial Intelligence Lab for Enterprises at NTU. Prior to joining NTU, he worked with Aalborg University, Microsoft Research Asia, and the University of Edinburgh. His current research interests include geospatial data management, spatio-temporal data mining, recommendation, and mining social media.

Quan Chen received his PhD in computer engineering from Nanyang Technological University (NTU). He is a Principal Research Fellow in NTU and his research interests include computer-assisted animation, computer graphics, game related technologies, artificial intelligence and machine learning. He is Associate Director of Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU), managing various research projects in AI, data analytics, robotics and smart computing.