Graph Contrastive Learning Method with Sample Disparity Constraint and Feature Structure Graph for Node Classification

Gangbin Chen[1], Junwei Cheng[1], Wanying Liang[1], Chaobo He[1,2](✉), and Yong Tang[1,2]

[1] School of Computer Science, South China Normal University, Guangzhou 510631, China
{gbchan,jung,sylvialaung,hechaobo,ytang}@m.scnu.edu.cn
[2] Pazhou Lab, Guangzhou 510330, Guangdong, China

Abstract. Most of the existing graph contrastive learning methods for node classification focus on exploiting topological information of the attributed networks, with little attention to the attribute information of the networks. Furthermore, in most graph contrastive learning methods, both positive and negative sample representations are generated by the same encoder, which makes them somewhat coupled and not conducive to distinguishing between them. But to our knowledge, most of these methods ignore the coupling between positive and negative sample representations, which may negatively affect the performance of the methods. To address these two issues, we propose a new method called DCFSG, in which we generate a feature structure graph from the attribute matrix, and then learn the node representations jointly from the topological structure graph and the feature structure graph. In addition, we impose a disparity constraint on the positive and negative samples and add corresponding loss to the objective function, which can help to lower the coupling between positive and negative samples and improves the performance of our method. The experimental results on real networks validate the effectiveness of our method.

Keywords: Graph contrastive learning · Node classification · Attributed networks · Feature structure graph · Disparity constraint

1 Introduction

Attributed networks have been widely used to represent relationships between objects in the real world, and they consist of nodes that are enriched with attribute information. For example, in the citation network, each node represents a document with its own attributes (e.g., keywords), and the edges represent the citation relationships between documents. The analysis of attributed networks can capture the attributes of objects to help people better understand real-world relationships, which is clearly meaningful and challenging.

Graph node classification is one of the most popular topics in the analyses of attributed networks. It aims to partition nodes of the network into different categories, which can help us understand the relationships between different nodes and infer the category of the nodes. In general, there are several broad categories of methods for graph node classification, such as Graph Neural Networks-based (GNNs) methods [1,11,15,19], Graph Autoencoder-based (GAE) methods [6,8,9], Graph Contrastive Learning-based (GCL) methods [2,4,16] and so on. Among these, GCL has received much attention because of several advantages for classification tasks. One is that GCL does not require label information and relies heavily on the graph structure itself, so it can be applied to the datasets without label information, improving the applicability and scalability of the method. Moreover, GCL can learn features by purely comparing similarity between different nodes with a small number of positive and negative samples, which can enhance the robustness of the classification method.

Recently, numerous researchers have put forward novel methods to effectively tackle node classification task within the realm of GCL. Velikovi *et al.* [14] proposed a classical GCL framework (i.e., DGI), whose main idea is to maximize the mutual information between the node embedding and the global summary vector of the whole graph. Yu *et al.* [20] proposed the SimGCL method, which discards the graph enhancement mechanism and instead adds uniform noise to the embedding space to create a contrast view. And their experiments showed that graph augmentation operations play only a minor role in the recommendation domain, which used to be considered necessary. Jing *et al.* [5] proposed HDMI method, which combines extrinsic and intrinsic mutual information for learning node embeddings of attributed networks and attributed multiplex networks. They also introduced a new method, called High-order Deep Informax, to optimize the high-order mutual information, which can better capture the structure and interactions in complex networks to learn more optimal representations for classification. All of the aforementioned methods have played a pivotal role in driving the development of GCL for the task of node classification.

Though the above methods have proven successful in the field of classification based on GCL, they may still have two issues that can be addressed. The first one is that most GCL methods for classification tasks focus on exploiting topological information of the attributed network to extract representations of graph nodes, but fail to fully utilize the attribute information of the graph. This means that the node representations we learn do not accurately reflect the nodes well in the attributed networks. The second one is that there is a slight coupling between the positive and negative sample representations because both positive and negative sample representations are generated by the same graph encoder. And most GCL methods do not take into account whether there are couplings between the positive and negative samples. Therefore, if we do not make full use of the attribute information in the attributed networks and ensure the disparities of the positive and negative samples, then the graph encoder performance obtained by the GCL methods may be suboptimal, which is not conducive to our subsequent node classification task.

Addressing the aforementioned two issues is crucial for improving the quality of graph node embeddings and enhancing the performance of graph node classification method. In this paper, we propose a novel method called DCFSG, which is based on general framework of GCL and it can solve both issues mentioned above. For the issue of not making full use of the attribute information in the attributed networks, we propose to use the K-Nearest Neighbor (KNN) graph generated by the node attributes as the feature structure graph. Then we learn the embedding of graph nodes based on the feature structure graph and the topological structure graph. In this way, we can better integrate node attribute information with graph topological information, which helps us to learn high-quality node representations and improves the performance of node classification method. For another issue, we use the Hilbert-Schmidt Independence Criterion (HSIC) [12] as a constraint to measure the disparity of the node embeddings of positive and negative samples. And it can enhance the disparity between these two types of embeddings, which help us to obtain a optimal graph encoder for classification. Our main contributions are summarized as follows:

- We propose a novel method DCFSG for learning node representations to tackle node classification task. Our method makes full use of the graph attribute information and constrains the positive and negative samples, which can improve the performance of the method on node classification task.
- We propose to learn the representations of nodes simultaneously from the topological structure graph and feature structure graph, in which we generate the KNN graph as the feature structure graph. Also, we propose to use HSIC as a constraint to enhance the diparity of positive and negative samples. More specifically, we add the corresponding loss to our objective function to improve the performance of our classification method.
- We conduct sufficient experiments on a series of benchmark datasets and the results demonstrate that DCFSG is superior over most methods or on par performance with existing methods.

2 Preliminaries

2.1 Attributed Networks

Attributed networks are a type of network where each node contains a set of attributes, and each edge represents a relationship between nodes. The attributed networks can be represented by $G(\mathbf{A}, \mathbf{X})$, where $\mathbf{A} \in \mathbb{R}^{N \times N}$ indicates the adjacency matrix and N indicates the number of nodes. $\mathbf{X} \in \mathbb{R}^{N \times M}$ indicates the attribute matrix and M indicates the dimension of attributes.

2.2 Graph Contrastive Learning

GCL is a self-supervised learning method for learning the representations of the nodes of graphs. And it use the concept of mutual information, which can be used to measure the correlation between two random variables, and the mutual information of a and b can be expressed as $I(a, b)$. GCL also uses global summary vectors as a way to efficiently compute mutual information. By maximizing the mutual information between node embeddings h and global summary vectors S, GCL can improve the distinguishability of nodes in the embedding space, which contributes to achieving good performance on various graph data tasks.

In summary, most GCL methods first generate a negative network $\tilde{G}$ by a corruption function $\mathcal{C}$. Secondly they generate the positive samples $\{h_1, \ldots, h_N\}$ for positive network as well as the negative samples $\{\tilde{h}_1, \ldots, \tilde{h}_N\}$ for the negative network by the same graph encoder $\mathcal{E}$ (e.g., GCN [7]). Then it use a readout function $\mathcal{R}$ to get the summary vector $S = \mathcal{R}(\{h_1, \ldots, h_N\})$ of the positive network G. Lastly, with S as input, it use the discriminator $\mathcal{D}$ to distinguish the positive node embeddings $\{h_1, \ldots, h_N\}$ from those negative node embeddings$\{\tilde{h}_1, \ldots, \tilde{h}_N\}$.

In most GCL methods, the maximization of $I(h_n, S)$ can be expressed as the maximization of the objective function $\mathcal{L}$, defined as follows:

$$\mathcal{L} = \sup\mathbb{E}[\log\mathcal{D}(h_n; S)] + \mathbb{E}[\log(1 - \mathcal{D}(\tilde{h}_n; S))] \tag{1}$$

where the discriminator $\mathcal{D}$ is used to differentiate the real node embedding h_n from the negative node embedding $\tilde{h}_n$ and the expectation is denoted by $\mathbb{E}$.

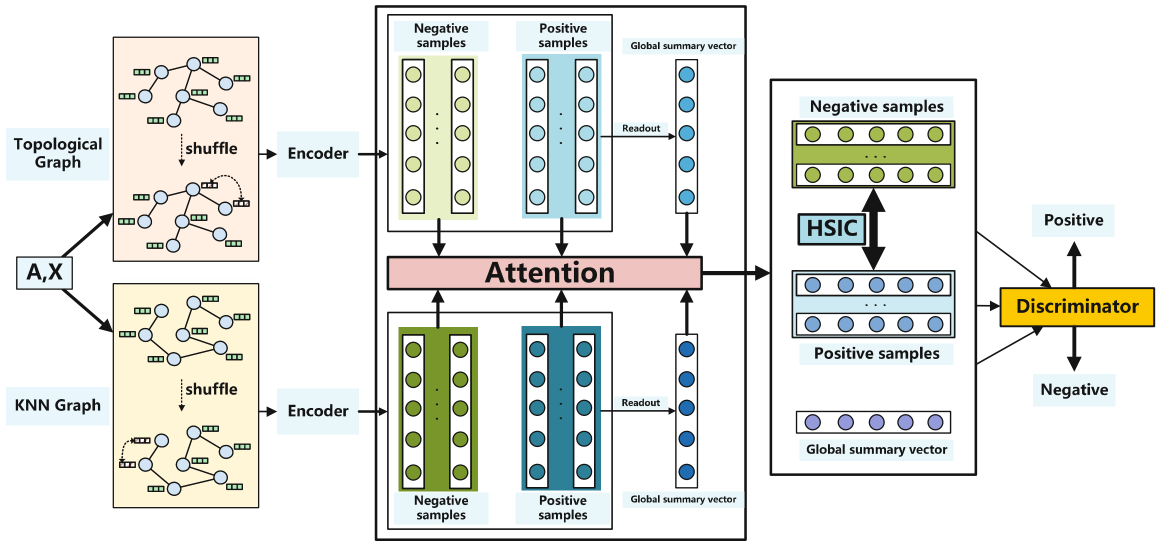


Fig. 1. The framework of DCFSG.

3 The Proposed Method

In this section, we introduce three main modules of our method DCFSG, which is shown in Fig. 1. In the first module, we construct the KNN graph as our feature structure graph, which is based on the node attributes by using the cosine similarity method. In the second module, we firstly use the topological and feature structure graphs to generate the corresponding positive and negative samples. Based on positive samples from both types of graphs, then we can obtain the global summary vectors for each graph. Lastly, we use an attention mechanism to adaptively fuse node embeddings of the same type, such as positive samples embeddings, negative samples embeddings, and global summary vectors. In the last module, we use HSIC to impose constraints on the representations of both positive and negative samples, which ensure the disparity between positive and negative sample representations. Finally, we add the corresponding loss to the objective function, which is beneficial for us to improve the performance of the method.

3.1 Feature Structure Graph Construction

To make full use of the attribute information of the nodes in the attributed networks, we construct KNN graph $\mathrm{G}_f = (\mathbf{A}_f,\mathbf{X})$ based on the attribute matrix $\mathbf{X}$, and $\mathbf{A}_f$ is the adjacency matrix of KNN graph. More specifically, we first calculate the similarity matrix $\mathbf{Y} \in \mathbb{R}^{N\times N}$ among N nodes by the cosine similarity method. And we present this method here, in which $\mathbf{y}_i$ and $\mathbf{y}_j$ are feature vectors of node i and j. Then we use the cosine value of the angle between two vectors to measure the similarity, the calculation formula is as follows:

$$\mathbf{Y}_{ij} = \frac{y_i \cdot y_j}{|y_i||y_j|} \tag{2}$$

After calculating the similarity matrix $\mathbf{Y}$, then we can select the top K similar node pairs for each node to establish edges, and subsequently obtain the adjacency matrix $\mathbf{A}_f$. Usually, the value of K here is chosen by us.

3.2 Node Embeddings Generation

Now we have the topological structure graph $\mathrm{G} = (\mathbf{A},\mathbf{X})$ and the feature structure graph $\mathrm{G}_f = (\mathbf{A}_f,\mathbf{X})$. Taking these two graphs as positive samples, we can use the corruption function $\mathcal{C}$ to construct negative samples for these two graphs, i.e., $(\tilde{\mathbf{A}},\tilde{\mathbf{X}}) = \mathcal{C}(\mathbf{A},\mathbf{X})$ and $(\tilde{\mathbf{A}}_f,\tilde{\mathbf{X}}) = \mathcal{C}(\mathbf{A}_f,\mathbf{X})$.

In this paper, we employ the corruption function based on random node permutation and specifically randomize the rows of the attribute matrix $\mathbf{X}$.

We use the graph encoder $\mathcal{E}$ to encode the graph nodes. Specifically, we use a single layer GCN as $\mathcal{E}$:

$$\mathbf{H} = ReLU(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\boldsymbol{\Theta}) \tag{3}$$

where $\hat{\mathbf{A}} = \mathbf{A}+\mathbf{I}_N$ denotes the adjacency matrix with inserted self-loops and $\hat{\mathbf{D}}$ denotes its degree matrix. $\mathbf{H} \in \mathbb{R}^{N\times d}$ denotes the node embedding matrix, where d denotes the dimension of the node embeddings. Also, $\boldsymbol{\Theta}$ is a learnable linear transformation that is applied to every node and ReLU denotes the nonlinear function.

Similarly we can obtain $\tilde{\mathbf{H}}$, $\mathbf{H}_f$ and $\tilde{\mathbf{H}}_f$ via Eq. 3. In general, we now have four node embedding matrices. $\tilde{\mathbf{H}}$ denotes the node embedding matrix obtained from the topological structure graph. Also, $\mathbf{H}_f$ and $\tilde{\mathbf{H}}_f$ represent the node embedding matrices generated by the feature structure graph. Notably, $\mathbf{H}$ and $\mathbf{H}_f$ denote positive sample embedding matrices while $\tilde{\mathbf{H}}$ and $\tilde{\mathbf{H}}_f$ denote negative sample embedding matrices.

Finally, we obtain the global summary vectors S and S_f by using average pooling as $\mathcal{R}$:

$$S = \mathcal{R}(\mathbf{H}) = \frac{1}{N}\sum_{n=1}^{N} h_n \tag{4}$$

where $h_n \in \mathbb{R}^{1\times d}$ is the n-th rows of $\mathbf{H}$. S and $S_f \in \mathbb{R}^{1\times d}$ represent the global summary vectors of topological structure graph and feature structure graph, respectively. Now we have two categories of positive sample node embeddings (i.e., h and h_f) from topological and feature structure graphs, respectively, as well as two classes of global summary vectors (i.e., S and S_f). In addition, we have $\tilde{h}$ and $\tilde{h}_f$ as negative sample node embeddings from above two mentioned graphs.

Considering that the node information should be learned from the topological and feature structure graphs, we use the attention mechanism to learn their corresponding importance as follows:

$$(\alpha_t, \alpha_f) = att(\mathbf{H}, \mathbf{H}_f) \tag{5}$$

where α_t and $\alpha_f \in \mathbb{R}^{N\times 1}$ denote the attention values of positive samples $\mathbf{H}$ and $\mathbf{H}_f$. Similarly, we can get the (β_t, β_f) and (γ_t, γ_f) through $att(\tilde{\mathbf{H}}, \tilde{\mathbf{H}}_f)$ and $att(S, S_f)$. β_t and β_f here denote the attention values of negative samples. γ_t and γ_f denote the attention values of different global summary vectors.

Specifically, we focus on node i, whose embedding in $\mathbf{H}$ is represented by $h^i \in \mathbb{R}^{1\times d}$. Firstly we transform the embedding through a nonlinear transformation, and then use one shared vector $q \in \mathbb{R}^{d^{'}\times 1}$ to calculate the attention value, where $d^{'}$ is the dimension of the output layer of the first linear layer in the attention mechanism. The specific calculation of the attention value ω_t^i is as follows:

$$\omega_t^i = q_\omega \cdot tanh(\mathbf{W}_\omega \cdot (h^i)^T + b_\omega) \tag{6}$$

Similarly, we can get the attention values ω_f^i for node i in embedding matrix $\mathbf{H}_f$ as follows:

$$\omega_f^i = q_\omega \cdot tanh(\mathbf{W}_\omega \cdot (h_f^i)^T + b_\omega) \tag{7}$$

Moreover, by replacing $(h^i)^T$ in Eq. (6) with $(\tilde{h}^i)^T$ and $(s^i)^T$ respectively, we can obtain the corresponding attention values ψ_t^i and ϕ_t^i. Also, by replacing $(h_f^i)^T$ in Eq. (7) with $(\tilde{h}_f^i)^T$ and $(s_f^i)^T$ separately, we can obtain the attention values ψ_f^i and ϕ_f^i. Here $\mathbf{W} \in \mathbb{R}^{d^{'}\times d}$ is the weight matrix and $b \in \mathbb{R}^{d^{'}\times 1}$ denotes the bias vector. Then we can normalize the attention values ω_t^i and ω_f^i with softmax function to calculate the final weights, as follows:

$$\alpha_t^i = softmax(\omega_t^i) = \frac{exp(\omega_t^i)}{exp(\omega_t^i) + exp(\omega_f^i)} \tag{8}$$

$$\alpha_f^i = softmax(\omega_f^i) = \frac{exp(\omega_f^i)}{exp(\omega_t^i) + exp(\omega_f^i)} \tag{9}$$

Similarly, we can calculate β_t^i and β_f^i from ψ_t^i and ψ_f^i. Also, we can obtain γ_t and γ_f from ϕ_t^i and ϕ_f^i. Finally, we have the learned weights $\alpha_t = [\alpha_t^i]$, $\alpha_f = [\alpha_f^i] \in \mathbb{R}^{N\times 1}$. Similarly we can get β_t, β_f, γ_t and γ_f. Then we can combine these embeddings with the attention values to obtain the final embeddings, as follows:

$$\mathbf{H}^* = \alpha_t \cdot \mathbf{H} + \alpha_f \cdot \mathbf{H}_f \tag{10}$$

$$\tilde{\mathbf{H}}^* = \beta_t \cdot \tilde{\mathbf{H}} + \beta_f \cdot \tilde{\mathbf{H}}_f \tag{11}$$

$$S^* = \gamma_t \cdot S + \gamma_f \cdot S_f \tag{12}$$

3.3 Disparity Constraint and Optimisation of Objective Functions

After we have the final node embeddings as described above (i.e., $\mathbf{H}^*, \tilde{\mathbf{H}}^*, S^*$), we can use the discriminator $\mathcal{D}$ [14] to score global summary vector S^* and positive samples $\mathbf{H}^*$, as well as global summary vector and negative samples $\tilde{\mathbf{H}}^*$, as follows:

$$\begin{aligned} \mathcal{D}(h_n^*, S^*) &= \sigma({h_n^*}^T \mathbf{M}_E S^*) \\ \mathcal{D}(\tilde{h}_n^*, S^*) &= \sigma((\tilde{h}_n^*)^T \mathbf{M}_E S^*) \end{aligned} \tag{13}$$

where $\mathbf{M}_E \in \mathbb{R}^{d\times d}$ is parameter matrix and σ is the sigmoid activation function. $h_n^* \in \mathbf{H}^*$ and $\tilde{h}_n^* \in \tilde{\mathbf{H}}^*$ represent positive and negative sample node embeddings, respectively. Although h_n^* and $\tilde{h}_n^*$ are node embeddings learned from different graph structures, they are generated by the same graph encoder $\mathcal{E}$. The two types of node embeddings are likely to have a certain degree of coupling. Therefore, if we want to learn a good discriminator, then we need to ensure that the positive and negative samples input to the discriminator have some disparities. In this paper, we use HSIC to enhance the disparity of these two embeddings. Formally, we define the disparity constraint $\mathcal{L}_h$ as follows:

$$\mathcal{L}_h = HSIC(\mathbf{H}^*, \tilde{\mathbf{H}}^*) = (n-1)^{-2} tr(\mathbf{R}\mathbf{K_T}\mathbf{R}\mathbf{K_{CT}}) \tag{14}$$

where $\mathbf{K}_T$ and $\mathbf{K}_{CT}$ are matrices defined as $k_{T,ij} = k_T(\mathbf{h}^i, \mathbf{h}^j)$ and $k_{CT,ij} = k_{CT}(\tilde{\mathbf{h}}^i, \tilde{\mathbf{h}}^j)$, respectively. $\mathbf{R} = \mathbf{I}$ - $\frac{1}{n}ee^T$, where e is an all-one column vector and $\mathbf{I}$ is an identity matrix.

Ultimately, imposing the loss corresponding to the disparity constraint on our objective function based on Eq. (1), we can obtain the final objective function as follows:

$$\mathcal{L}_E = \mathbb{E}[log\mathcal{D}(h_n^*, S^*)] + \mathbb{E}[log(1 - \mathcal{D}(\tilde{h}_n^*, S^*))] \tag{15}$$

$$\mathcal{L} = \mathcal{L}_E + \lambda_h \mathcal{L}_h \tag{16}$$

where λ_h is tunable coefficient. Our method is based on the basic GCL framework, which maximizes the mutual information of the final node embeddings

h_i^* and S^*. Also, it imposes disparity constraint on the positive and negative samples. Finally, we can update parameters involved in $\mathcal{E}$, $\mathcal{D}$ and $\mathcal{R}$ by applying gradient descent to maximize Eq. (16). We list the whole process of our proposed method in Algorithm 1.

Algorithm 1. Framework of DCFSG.

```
Input: Original Graph $G = (\mathbf{A}, \mathbf{X})$, Maximum epoch value $T_{max}$;
Output: Node embeddings $\mathbf{H}$, Classification results;
1: ▶ Generate a KNN graph:
2:   G = $(\mathbf{A}_f, \mathbf{X})$ via Eq. (2) ;
3: while epoch $< T_{max}$ or our method has not yet converged do
4:   ▶ Generate representations
5:     Use encoder $\mathcal{E}$ to generate $\mathbf{H}$, $\tilde{\mathbf{H}}$, $\mathbf{H}_f$ and $\tilde{\mathbf{H}}_f$ via Eq. (3);
6:     Use Readout function $\mathcal{R}$ to generate $S$ and $S_f$ via Eq. (4);
7:   ▶ Using the attention mechanism
8:     Calculate the values of $\alpha_t, \alpha_f, \beta_t, \beta_f, \gamma_t$ and $\gamma_f$ via Eq. (5),
       Eq. (6), Eq. (7), Eq. (8) and Eq. (9);
9:     Obtain $\mathbf{H}^*$, $\tilde{\mathbf{H}}^*$ and $S^*$ via Eq. (10), Eq. (11) and Eq. (12);
10:  ▶ Calculate the loss $\lambda$
11:    Calculate the HSIC via Eq. (14);
12:    Use the discriminator $\mathcal{D}$ to score via Eq. (13);
13:    Calculate the loss via Eq. (15) and Eq. (16);
14:    Update parameters of the attention mechanism and HSIC;
15:    Update parameters involved in $\mathcal{E}$, $\mathcal{D}$ and $\mathcal{R}$;
16: end while
17: return Node embeddings $\mathbf{H}$, Classification results through inputting $\mathbf{H}$ into the MLP.
```

4 Experiments

4.1 Experimental Setup

(1) Datasets. We conduct experiments on three real-world datasets, i.e., Cora, Citeseer and Pubmed[1], which are summarized in Table 1.

(2) Baselines. We compare the metrics (i.e., Accuracy (ACC) and macro F1-score (F1)) of our method with other methods. These methods are: DeepWalk [10], Planetoid [18], GCN [7], GraphSAGE [3], GAT [13], GIN [17] ,DGI [14]. And we report the ACC and F1 of all these methods after several runs of training. Lastly we summary the results in Table 2.

(3) Parameter Settings. For our method, we use the learned representations to tackle node classification task through an MLP. Also, we use the Adam optimizer and set the learning rate to 0.001. For the Cora, Citeseer and Pubmed datasets, we set the dimension of the hidden layer in the encoder $\mathcal{E}$ to 896, 512 and 512, respectively. And the dimension of the hidden layer in the attention mechanism is 16. Also, the value of K is fixed as 7, which is used to generate the KNN graph as the feature structure graph. Lastly, for Cora, Citeseer and Pubmed datasets, we set the dimension of the output layers of the MLP to 7, 6 and 3, respectively.

[1] https://github.com/GDM-SCNU/Datasets.

Table 1. Summary of the datasets used in our experiments.

Dataset	Nodes	Edges	Attributes	Classes
Cora	2708	5429	1433	7
Citeseer	3327	4732	3703	6
Pubmed	19717	44338	500	3

Table 2. Classification results on three datasets. (Bold: best)

Dataset	Cora		Citeseer		Pubmed	
Metric	ACC	F1	ACC	F1	ACC	F1
Planetoid	0.771	0.766	0.646	0.616	0.753	0.749
DeepWalk	0.689	0.683	0.454	0.437	0.748	0.729
GCN	0.816	0.809	0.708	0.679	0.773	0.679
GraphSAGE	0.791	0.788	0.708	0.679	0.760	0.758
GAT	0.792	0.782	0.722	0.689	0.780	0.769
GIN	0.748	0.751	0.637	0.601	0.774	0.770
DGI	**0.823**	**0.810**	0.718	0.675	0.771	0.773
DCFSG	0.820	**0.810**	**0.724**	**0.693**	**0.812**	**0.810**

4.2 Classification Performance Analysis

The node classification results are reported in Table 2, from which we have the following observations:

(1) Our proposed DCFSG achieves better results than most methods on the three real datasets. For the Cora, Citeseer and Pubmed datasets, DCFSG's ACC improved by a maximum of 13.1%, 27% and 6.4% as well as the F1 of DCFSG improved by a maximum of 12.1%, 25.3% and 13.1%. Furthermore, our proposed method achieves the best performance on both the Citeseer and Pubmed datasets when compared to other methods. And the ACC on the Cora dataset is only 0.3% lower than DGI.

(2) Overall, we achieve the greatest performance advantage on the larger dataset, Pubmed. On the smaller dataset Cora, our proposed method also performs well compared to other methods, but does not reach the most optimal. This shows that our model performs well in learning node embeddings and classifying nodes on large graphs, but does not perform well on small graphs. Probably because when the graph is small, the number of positive and negative samples is small, which is not conducive to learning a good encoder for the graph.

(3) On the other hand, the classification performance of the DCFSG outperforms most of the supervised and unsupervised learning methods, which indicates that our method can extract graph information better than other methods and can learn better node representations.

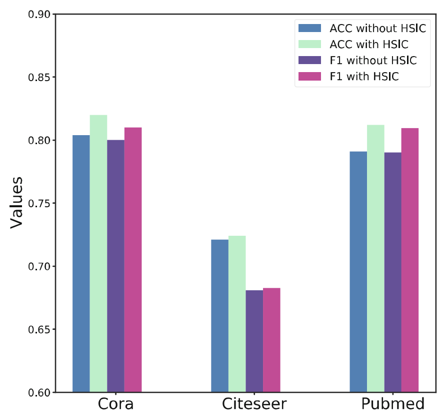


Fig. 2. Ablation study.

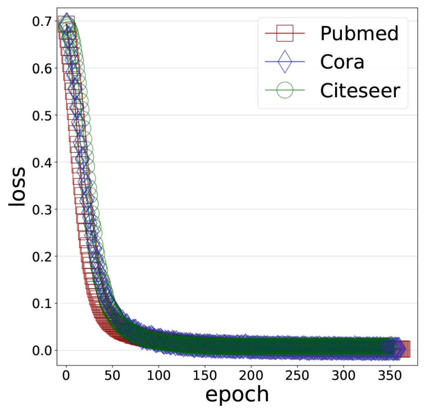


Fig. 3. Convergence experiment.

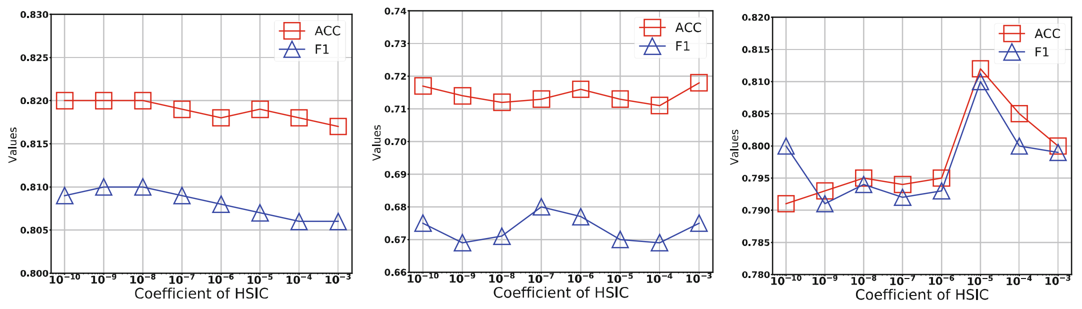
4.3 Hyperparameter Analysis

Hyperparameter Analysis of $\boldsymbol{\lambda_h}$. The analysis of hyperparameter λ_h is divided into two parts, namely ablation study analysis and sensitivity experimental analysis, based on whether λ_h in Eq. (16) is zero or λ_h is greater than zero.

(I) When λ_h=0, this experiment represents the ablation study analysis. We compare DCFSG with its one variant on all benchmark datasets. From the result in Fig. 2, we can have the following observations:

- The experimental results show that our method with HSIC performs better than that without HSIC on the three datasets. The method with HSIC has significantly better ACC than the method without HSIC on the three datasets, and for F1, the performance of the method with HSIC is significantly better on Pubmed and Cora, while the difference with and without HSIC is not significant on the Citeseer dataset.
- The experimental results demonstrate that our use of HSIC to enhance the disparity between positive and negative samples is effective, which indicates that our use of HSIC can improve the ability of our method to learn graph node embeddings and improve the accuracy of classification.

(II) When $\lambda_h > 0$, this experiment represents the sensitivity analysis and the result is shown in Fig. 4. We analyze the effect of hyperparameter λ_h on classification performance and the result shows that the performance of our method changes very little as the value of λ_h increases. In general, experiments on the performance of our method under different hyperparameters λ_h show that our method is not sensitive to hyperparameter λ_h. In a way, it means that our method is more stable, and our insensitivity to this hyperparameter λ_h reduces the complexity of parameter tuning and speeds up the training of our method.



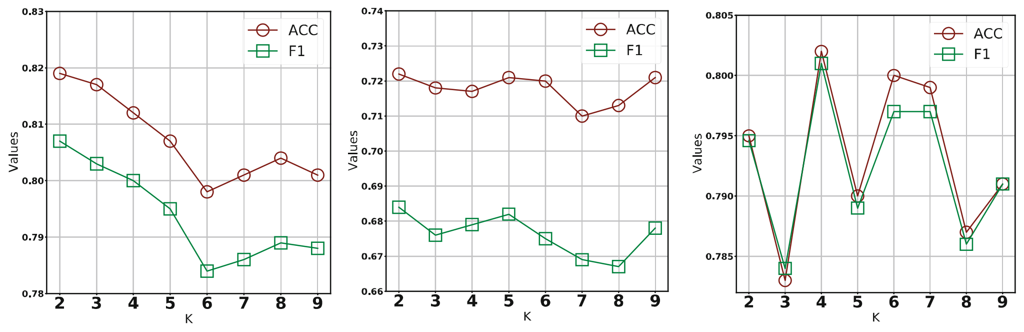
(a) Cora (b) Citeseer (c) Pubmed

Fig. 4. λ_h-sensitivity analysis on three datasets.

K-Sensitivity Analysis. We analyze the impact of the value of K on the classification performance, where K is used to determine the construction of the KNN graph. From the result shown in Fig. 5, we can have the following observations:

(1) In Cora dataset, ACC and F1 tend to decrease overall as the value of K increases. But for Citeseer dataset, ACC and F1 exhibit minimal variation. As the value of K increases, we find that there is significant variation in both ACC and F1 in Pubmed dataset.

(2) Within a certain range of the value of K changes, this hyperparameter K has a relatively small effect on changes in the model performance. This means that even if the value of this hyperparameter K changes, the performance of our method may not change significantly.



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

Fig. 5. Results of K-sensitivity analysis on three datasets.

4.4 Convergence Analysis

We verify the convergence of our method using Cora, Citeseer and Pubmed datasets, respectively. The experimental results shown in Fig. 3 demonstrate that our method converges quickly, with the losses calculated on the three datasets starting to converge at around 50 epochs and stabilising at around 200 epochs. This indicates that our method has a good generalisation ability to a certain extent, which enables us to facilitate hyperparameter tuning and experimentation. Since the method converges quickly, it allows us to verify various hyperparameter combinations more rapidly, ultimately reducing the time required for hyperparameter optimization.

5 Conclusion

In this paper, we find that most GCL methods focus on using graph topological information to extract the representations of nodes, without making full use of the node attribute information. Moreover, most existing methods do not explicitly consider the impact of the disparity of positive and negative samples within a graph on method performance. Therefore, we propose a novel method, called DCFSG, which exploits both graph topological and attribute information to jointly learn node representations, and enhances the disparities between the positive and negative node embeddings by adding the disparity constraint. The extensive experimental data demonstrates the effectiveness of our method.

Acknowledgement. This work was supported in part by the National Natural Science Foundation of China under Grant 62077045, and Grant U1811263.

References

1. Abu-El-Haija, S., Kapoor, A., Perozzi, B., Lee, J.: N-GCN: multi-scale graph convolution for semi-supervised node classification. In: Uncertainty in Artificial Intelligence, pp. 841–851 (2020)
2. Brzozowski, U., Siemaszko, K.: Representation learning on graphs (2021)
3. Hamilton, W.L., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Annual Conference on Neural Information Processing Systems, pp. 1024–1034 (2017)
4. Hassani, K., Ahmadi, A.H.K.: Contrastive multi-view representation learning on graphs. In: ICML. Proceedings of Machine Learning Research, vol. 119, pp. 4116–4126 (2020)
5. Jing, B., Park, C., Tong, H.: HDMI: high-order deep multiplex infomax. In: WWW: The Web Conference, pp. 2414–2424 (2021)
6. Kipf, T.N., Welling, M.: Variational graph auto-encoders. arXiv preprint abs/1611.07308 (2016)
7. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017)
8. Niknam, G., Molaei, S., Zare, H., Clifton, D., Pan, S.: Graph representation learning based on deep generative gaussian mixture models. Neurocomputing **523**, 157–169 (2023)
9. Pan, S., Hu, R., Long, G., Jiang, J., Yao, L., Zhang, C.: Adversarially regularized graph autoencoder for graph embedding. In: IJCAI, pp. 2609–2615 (2018)
10. Perozzi, B., Al-Rfou, R., Skiena, S.: DeepWalk: online learning of social representations. In: ACM, pp. 701–710 (2014)
11. Rong, Y., Huang, W., Xu, T., Huang, J.: DropEdge: towards deep graph convolutional networks on node classification. In: ICLR (2020)

12. Song, L., Smola, A.J., Gretton, A., Borgwardt, K.M., Bedo, J.: Supervised feature selection via dependence estimation. In: ICML, vol. 227, pp. 823–830. ACM (2007)
13. Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: ICLR (2018)
14. Velickovic, P., Fedus, W., Hamilton, W.L., Liò, P., Bengio, Y., Hjelm, R.D.: Deep graph infomax. In: ICLR (2019)
15. Wang, X., Zhu, M., Bo, D., Cui, P., Shi, C., Pei, J.: AM-GCN: adaptive multi-channel graph convolutional networks. In: SIGKDD, pp. 1243–1253 (2020)
16. Wang, Z., Yan, S., Zhang, X., da Vitoria Lobo, N.: Self-supervised visual feature learning and classification framework: based on contrastive learning. In: ICARCV, pp. 719–725. IEEE (2020)
17. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: ICLR (2019)
18. Yang, Z., Cohen, W.W., Salakhutdinov, R.: Revisiting semi-supervised learning with graph embeddings. In: Balcan, M., Weinberger, K.Q. (eds.) ICML, vol. 48, pp. 40–48 (2016)
19. Yao, L., Mao, C., Luo, Y.: Graph convolutional networks for text classification. In: AAAI, pp. 7370–7377 (2019)
20. Yu, J., Yin, H., Xia, X., Chen, T., Cui, L., Nguyen, Q.V.H.: Are graph augmentations necessary?: Simple graph contrastive learning for recommendation. In: SIGIR, pp. 1294–1303 (2022)