SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

June 2024, Vol. 67, Iss. 6, 162101:1–162101:16
https://doi.org/10.1007/s11432-023-3897-2

A novel graph oversampling framework for node classification in class-imbalanced graphs

Riting XIA[1,2], Chunxu ZHANG[1,3], Yan ZHANG[4], Xueyan LIU[1,3*] & Bo YANG[1,3*]

[1]*Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun* 130012*, China;*
[2]*College of Artificial Intelligence, Jilin University, Changchun* 130012*, China;*
[3]*College of Computer Science and Technology, Jilin University, Changchun* 130012*, China;*
[4]*College of Communication Engineering, Jilin University, Changchun* 130012*, China*

Received 23 May 2023/Revised 7 October 2023/Accepted 31 October 2023/Published online 15 April 2024

Abstract Graph neural network (GNN) is a promising method to analyze graphs. Most existing GNNs adopt the class-balanced assumption, which cannot deal with class-imbalanced graphs well. The oversampling technique is effective in alleviating class-imbalanced problems. However, most graph oversampling methods generate synthetic minority nodes and their edges after applying GNNs. They ignore the problem that the representations of the original and synthetic minority nodes are dominated by majority nodes caused by aggregating neighbor information through GNN before oversampling. In this paper, we propose a novel graph oversampling framework, termed distribution alignment-based oversampling for node classification in class-imbalanced graphs (named Graph-DAO). Our framework generates synthetic minority nodes before GNN to avoid the dominance of majority nodes caused by message passing in GNNs. Additionally, we introduce a distribution alignment method based on the sum-product network to learn more information about minority nodes. To our best knowledge, it is the first to use the sum-product network to solve the class-imbalanced problem in node classification. A large number of experiments on four real datasets show that our method achieves the optimal results on the node classification task for class-imbalanced graphs.

Keywords graph neural network, class-imbalanced graphs, sum-product network, oversampling, node classification

1 Introduction

Complex networks or graphs have emerged in various scenarios, such as traffic networks [1], social networks [2], neural networks [3], and citation networks [4], in recent years. The study of graphs has far-reaching scientific significance and essential application value. The semi-supervised node classification task in graphs, which refers to inferring the unknown labels of some nodes by using the graph information (including topology and node features) and the labels of partial nodes, has been widely concerned.

Class-imbalanced problem is common in node classification tasks. That is, some classes may have a smaller number of nodes compared to others. For example, the hot research topic includes more studies than the unpopular topic in the citation network [4]. Social networks whose categories are hobbies show a category imbalance due to the prevalence of hobbies. Most of the users in social networks are authentic, and only a small fraction of users are fake accounts [5]. However, most of the existing studies on node classification tasks do not consider the class-imbalanced problem in the class-imbalanced graphs[1)].

Graph neural networks (GNNs) have shown their effectiveness in semi-supervised node classification tasks. Typically, GNNs [6, 7] employ a message-passing mechanism where each node in the graph recursively receives and aggregates messages such as features from its neighboring nodes to update the node representations. For example, graph convolutional network (GCN) [8] is the popular network embedding method, which involves combining the features of neighboring nodes to obtain node representations. In

* Corresponding author (email: xueyanliu@jlu.edu.cn, ybo@jlu.edu.cn)

1) Class-imbalanced graphs refer to the graph that the number of nodes in different classes varies significantly, as defined in Definition 2. In addition, we focused on the node classification task.

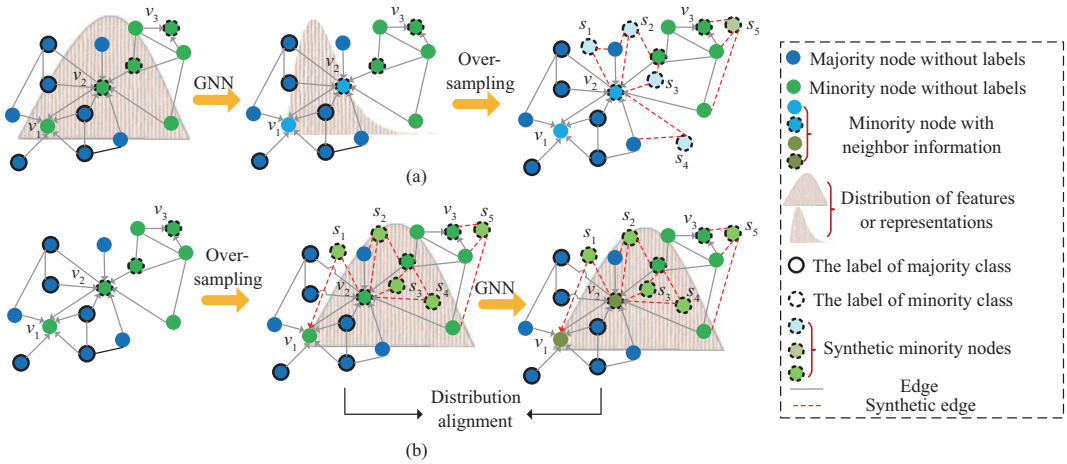

Figure 1 (Color online) Comparative analysis of current oversampling methods and our proposed method in GNN through schematic diagrams. The arrow directions indicate the aggregation of neighbor information. (a) Schematic diagram of most oversampling methods currently used; (b) schematic diagram of our oversampling method.

order to learn better representations and use them for various downstream tasks, researchers have proposed many variants of GCN [9–14]. Unlike GCN, which uses full-size neighboring nodes to obtain node representations, GraphSAGE [15] samples and aggregates local neighboring node features to generate representations. Although GNNs have made significant progress in node classification tasks, the existing work rarely considers the class-imbalanced problem for graphs. However, just like the imbalanced data in many other domains [16,17], there are also a large number of class-imbalanced graphs in the real world, as previously mentioned. The massage-passing mechanism of GNNs magnifies the misclassification through the topology of the class-imbalanced graphs on node classification tasks [18–20].

In recent years, researchers have put forward some improved GNN-based methods for the class-imbalanced problem in graphs [18,20,21]. Similar to the categories in other domains, we divide the class-imbalanced methods for node classification on graphs into data-level and algorithm-level approaches. The data-level methods usually use the oversampling technique to balance the distribution of classes [22,23]. Algorithm-level methods typically leverage the inherent characteristics of class-imbalanced graphs to augment existing algorithms [18,24]. For the data-level methods, the oversampling methods are designed to generate synthetic minority nodes. This is an effective method to alleviate the graph class-imbalanced problem. Researchers have proposed some graph oversampling methods [20, 22, 23, 25] to obtain better node classification for the minority nodes. For example, GraphSMOTE [20] learns the representation of the nodes by using a GNN encoder and generates synthetic minority nodes by SMOTE [26], and adds edges to these nodes by a decoder. GraphMixup [23] constructs semantic relation space through the way of disentanglement and synthesizes minority nodes by implementing the oversampling strategy at the semantic level. In addition, GraphMixup adaptively determines the number of minority nodes needed to be generated by using a reinforcement learning mechanism. To maintain the information of minority nodes, Wu et al. [25] proposed a minority-weighted GNN named mGNN, which utilizes a weighted aggregation method to increase the impact of minority nodes on node representations during the aggregation process.

Generally speaking, the steps of most oversampling methods currently used for graph class-imbalanced problem are as follows. (1) Learn the representations of nodes through GNN. (2) Generate synthetic minority nodes from the node representations. (3) Add edges to the newly synthesized minority nodes. However, these oversampling methods rarely consider that the aggregation of neighbor information in GNNs might affect the node representations in the scenarios of the class-imbalanced graphs. We have analyzed this problem in detail and concluded that it would lead to the following problems. (1) The distribution of node representations has shifted (named as distribution mismatch), i.e., some minority nodes' representations skew to majority nodes' representations. As shown in Figure 1(a), the original representations of nodes v_1 and v_2 are similar to minority nodes' representations. After applying GNN, their representations are preferred to the representations of majority nodes. Furthermore, owing to the complexity of real-world data distributions and the lack of prior knowledge, determining the distribution of node representations is a challenging task. (2) Aggregating neighbor information using GNN before oversampling may produce some confusing synthetic minority nodes, i.e., the synthetic nodes with minority labels but their representations are similar to the representations of majority nodes. The nodes

s_1–s_4 in Figure 1(a) are these kinds of confusing nodes after applying oversampling. These two problems will generally lead to inaccurate synthesized edges and unsatisfied node classification results.

To address the above problems, we propose a novel oversampling-based framework for class-imbalanced node classification on graphs, named Graph-DAO. Graph-DAO can learn better representations of class-imbalanced graphs in three ways. (1) To mitigate the distribution mismatch of node representations caused by the aggregation of neighbor information in GNN in class-imbalanced scenarios, as well as the generation of confusing synthetic minority nodes through oversampling after applying GNN, we utilize an autoencoder based on multilayer perceptron (MLP-based autoencoder) to obtain node representations and employ oversampling techniques in the latent space to generate synthetic minority nodes before aggregating neighbor information in GNN, as shown in Figure 1(b). In the oversampling process, the dual decoder reconstructs both the node features and adjacency matrix. (2) We utilize another GNN-based autoencoder to obtain node representations. Specifically, we merge the features and edge information of the synthetic minority nodes obtained by the dual decoder into the original graph as input for GNN to learn node representations. Additionally, We utilize the decoder to decode the adjacency matrix to constrain the node representations. (3) To alleviate the mismatch between the distributions of the representations obtained by MLP-based autoencoder and GNN-based autoencoder, we utilize KL-divergence to perform distribution alignment between the representations obtained by the two kinds of autoencoders. Specifically, we adopt sum-product network (SPN)[2)] to obtain the probability distribution of the node representations. After obtaining the topological structure and node representations incorporating both the synthetic and existing nodes, we use the GNN classifier for node classification.

The main contributions of this article are as follows.

- We propose a novel oversampling framework for class-imbalanced scenarios, and for the first time, we consider oversampling before aggregating neighbor information in GNNs. Our method alleviates the distribution mismatch of node representations resulting from aggregation neighbor information in GNNs and the confusion of synthetic minority nodes caused by oversampling after applying GNN.
- We propose a distribution alignment method based on SPNs, which mitigates the risk of distribution mismatch in node representations. To our knowledge, this is the first time that SPNs have been considered in class-imbalanced graphs.
- Through extensive experiments, we show that our approach outperforms state-of-the-art graph-based oversampling algorithms on several real-world class-imbalanced datasets.

2 Related work

In this section, we introduce the studies for class-imbalanced graphs and SPNs.

2.1 Methods for class-imbalanced graphs

In real-world scenarios, class-imbalanced graphs are prevalent and have gained significant attention in research. Similar to the image field, existing algorithms for addressing the class imbalance in graphs can be categorized into two main approaches: algorithm-level and data-level methods.

The algorithm-level methods utilize the inherent characteristic of class-imbalanced graphs to enhance existing algorithms [18, 21, 24, 27–32]. For example, Shi et al. [21] introduced a dual-regularized GCN named DR-GCN, which utilizes adversarial training to facilitate the differentiation of node representations from distinct classes. To tackle the challenge of decision boundaries being skewed towards majority classes due to topological imbalance, ReNode [18] introduces an influence conflict metric to identify annotated nodes that potentially lie in proximity to class boundaries and to reduce their influence by assigning weights to these nodes. Shi et al. [31] proposed to utilize the boosting algorithm to address the challenge of quantity imbalance in the node classification task. To overcome the issue of false positives on majority classes that arise from compensating for minority nodes, Song et al. [24] proposed the topology-aware margin, which adjusts the marginal nodes based on the degree of deviation in the connectivity patterns.

The data-level methods usually balance the classes of the graph by generating synthetic minority nodes [19, 20, 22, 23, 25, 33]. One of the most representative methods is oversampling on graphs. For example, GraphSMOTE [20] utilizes an oversampling method to generate synthetic minority nodes and employs an edge predictor to establish edge connectivity relationships for newly synthesized minority

2) For a comprehensive understanding of the specific definition and details of SPNs, we direct readers to Subsection 3.1.

nodes. To address the issue of overfitting in minority classes and avoid losing information from the majority class, GNN-CL [22] introduces the idea of curriculum learning into the model. mGNN [25] proposes a novel approach to address the limitation of GraphSMOTE, which fails to discern the significance of majority and minority nodes during the process of information aggregation of GNN. Specifically, mGNN utilizes GNN-weighted learning to improve the representation of minority nodes, and then performs oversampling based on this representation. The GraphMixup [23] constructs semantic relation space through the way of disentanglement, and synthesizes minority nodes by oversampling strategy at the semantic level. In addition, GraphMixup adaptively determines the number of synthetic minority nodes using a reinforcement learning mechanism. In addition to the above graph oversampling methods, to address the issue of neighbor overfitting in minority class nodes, GraphENS [19] proposes a novel augmentation strategy that generates minority nodes and their edges through ego networks. ImGAGN [33] generates nodes and edges by introducing a generative adversarial network.

In this paper, we focus on utilizing oversampling techniques to tackle class imbalance problems in graphs. The common feature of these oversampling techniques mentioned above is that they utilize the structure and attribute information of the graph to learn node representations through GNN, and perform oversampling based on this. Although these methods have been proven to be effective, they fail to consider the aggregation of neighbor information in GNNs may affect the node representations, resulting in a distribution mismatch of node representations. Oversampling based on this bias has the potential to generate nodes with labels of minority classes, but similar representations to minority nodes. That may lead to unsatisfied node classification results.

2.2 SPNs

SPNs [34] have gained significant attention due to their ability to model complex probability distributions effectively and their tractability. Recently, several studies have focused on developing SPN methods with better performance on many tasks [35–38]. Among these, random and tensorized SPN (RAT-SPN) [37] and einsum network (EiNet) [38] have shown to be effective in different domains. To make the SPNs easy to extend, and easy to integrate with the deep learning framework, RAT-SPN introduces a straightforward and expandable approach to construct SPNs by utilizing the region graph [39] as the structure. Despite the relatively favorable performance and successful application of RAT-SPN in many tasks, its training on real-world data remains challenging due to the sparsely connected computational graphs, which poses difficulties in scaling SPNs to larger datasets. To address this limitation, EiNet consolidates numerous arithmetic operations in a single monolithic einsum-operation[3)] [38,40]. Both methods have demonstrated exceptional performance in several benchmarks and have shown great potential for future research on SPN-based models.

At present, there exist three works that implement SPNs in the context of graphs. Nath et al. [41] introduced a relational SPN named RSPN, which is utilized to address the link prediction task. Similarly, Zheng et al. [42] introduced GraphSPN to effectively model noisy and intricate graph data in the robotics domain. Xia et al. [43] proposed a graph variational autoencoder [44] that is subject to the constraints of SPNs in order to obtain the dependencies among hidden variables.

However, SPNs are seldom applied in graph-related tasks and have not been utilized in class-imbalanced graphs. Since SPNs can learn the distribution directly from data and its inference is tractable [36], in this paper, we consider using SPNs to learn the distribution of node representations with unknown distribution. Although probabilistic graphical models (PGMs) [45] can also learn the distribution from data, their inference is often intractable (approximate reasoning is usually required), unless it is a simple problem or a specific type of model with limited expressiveness. Therefore, adopting SPNs to learn the distributions of node representations with unknown distributions is appropriate.

3 Preliminaries and problem definition

3.1 Preliminaries

This subsection will provide the necessary background information about SPNs and class-imbalanced graphs.

3) The einsum operation (as shown in reference [38]) implements the Einstein notation [40] of tensor-product contraction, which serves to unify various standard linear algebra operations, including but not limited to matrix multiplication and outer product.

Definition 1 (SPNs)**.** An SPN $S = (\boldsymbol{N}, \boldsymbol{E}, \boldsymbol{\theta})$ is a rooted directed acyclic graph, which characterizes joint distributions of random variables (RVs) $\mathcal{X}$ (i.e., $p(\mathcal{X})$). $\boldsymbol{N}$ and $\boldsymbol{E}$ represent the set of nodes and edges, respectively. $\boldsymbol{\theta}$ represents the parameters of SPN. An SPN is composed of three distinct types of nodes: leaf nodes (n_l), product nodes (n_p), and sum nodes (n_s). These nodes are defined as follows.

(1) The leaf node (n_l) value is determined by computing its probability distribution individually on a subset of $\mathcal{X}_L \subset \mathcal{X}$, denoted as $p_{n_l} = p(\mathcal{X}_L|\theta_L)$, where θ_L is the parameter of the distribution at n_l.

(2) The product node (n_p) value is obtained by computing the product of all its children, i.e., $p_{n_p} = \prod_{c\in \mathrm{ch}(n_p)} p_c$. p_c is the probability value of node c and $\mathrm{ch}(n_p)$ is the set of child nodes of n_p.

(3) The sum node (n_s) value is determined by calculating the sum of its all children, i.e., $p_{n_s} = \sum_{c\in \mathrm{ch}(n_s)} w_{n_s,c} p_c$, where the nonnegative weight $w_{n_s,c} \in \boldsymbol{\theta}$ is associated with the edge $(n_s, c) \in \boldsymbol{E}$ and n_s is normalized if $\sum_{c\in \mathrm{ch}(n_s)} w_{n_s,c} = 1$.

EiNet [38] leverages a single einsum operation that combines many arithmetic operations, resulting in greater speed and memory efficiency compared to RAT-SPN [37]. Therefore, we will employ EiNet to design our method, Graph-DAO.

Definition 2 (Class-imbalanced graphs)**.** A graph $\boldsymbol{G}_{\mathrm{im}} = (\boldsymbol{V}, \boldsymbol{C}, \boldsymbol{F}, \boldsymbol{E})$ is a class-imbalanced graph, which contains node set $\boldsymbol{V} = \{v_1, \ldots, v_n\}$, class set $\boldsymbol{C} = \{C_1, \ldots, C_k\}$, feature set $\boldsymbol{F} = \{f_1, \ldots, f_m\}$, and edge set $\boldsymbol{E} = \{e_{ij} = (v_i, v_j)|\text{there exists a link between } v_i \text{ and } v_j\}$, where n, k, and m represent the quantity of nodes, class types, and feature types, respectively. An adjacency matrix $\boldsymbol{A} \in \{0, 1\}^{n\times n}$ is used to model the edges, where $a_{ij} = a_{ji} = 1$ if $e_{ij} \in \boldsymbol{E}$, otherwise $a_{ij} = 0$. $\boldsymbol{A}_i$ is the i-th row vector of $\boldsymbol{A}$. The matrix $\boldsymbol{X} \in \mathbb{R}^{n\times m}$ is used to describe the node features, where each row $\boldsymbol{X}_i$ of $\boldsymbol{X}$ represents the features of node v_i. $|C_k|$ represents the quantity of nodes in class k, and $\rho_c = \frac{\min_k |C_k|}{\max_k |C_k|}$ is class imbalance ratio [23]. $\boldsymbol{G}_{\mathrm{im}} = (\boldsymbol{V}, \boldsymbol{C}, \boldsymbol{F}, \boldsymbol{E})$ is an imbalanced network if ρ_c is small, i.e., there exists $|C_i| \ll |C_j|$.

3.2 Node classification with GNNs

GNNs have made remarkable achievements in real-world scenarios [46–51]. We introduce GNNs for node classification tasks. The GNNs consist of three functions, including the message function, feature aggregation function, and node feature update function. The expression of a GNN at layer l is

$$x_v^{l+1} = h_l(x_v^l, \varphi_l(\{m_l(x_v^l, x_u^l, e_{v,u})|u \in N(v)\})), \tag{1}$$

where m_l, φ_l, and h_l are the message, feature aggregation, and node feature update functions of layer l in GNNs, respectively. x_v^l and x_u^l are the representations of node v and u at layer l. $N(v)$ is the set of adjacent nodes that are directly connected to v, $e_{v,u}$ is the edge weight of edge $\{v, u\} \in \boldsymbol{E}$.

We obtained our GNN classifier f by feeding the node representations obtained by a GNN into a linear classifier. Specifically, given a graph $\boldsymbol{G}_{\mathrm{im}}$ and the labels $\boldsymbol{Y}_t$ of partial nodes $\boldsymbol{V}_t$, we aim to make predictions for the labels $\boldsymbol{Y}_u$ of nodes without labels, i.e., $f(\boldsymbol{G}_{\mathrm{im}}, \boldsymbol{Y}_t) \to \boldsymbol{Y}$, where $\boldsymbol{Y}_t \cup \boldsymbol{Y}_u = \boldsymbol{Y}, \boldsymbol{Y} \in \mathbb{R}^n$ is the label set of all nodes. In summary, our goal is to learn a GNN classifier that performs effectively for both the minority and majority classes.

4 Methodology

In this part, we first present the overall framework of Graph-DAO. Then, we present our proposed Graph-DAO in detail. Finally, the training and optimization of Graph-DAO are introduced.

4.1 Graph-DAO framework

The general architecture of Graph-DAO is illustrated in Figure 2, which includes four parts: MLP-based autoencoder M-AE, GNN-based autoencoder G-AE, SPN-based distribution alignment module, and GNN-based classifier. (1) MLP-based autoencoder M-AE is employed to obtain a graph that includes synthetic minority nodes. Specifically, the MLP encoder obtains node representations, and the synthetic minority nodes are generated in the latent space utilizing oversampling techniques. Subsequently, a dual decoder is utilized to reconstruct the feature matrix and adjacency matrix containing synthetic minority nodes. (2) The synthetic minority nodes and edges obtained from the M-AE are integrated into the original graph $\boldsymbol{G}_{\mathrm{im}}$, resulting in a new graph $\hat{\boldsymbol{G}}$. The node representations on this new graph $\hat{\boldsymbol{G}}$ are achieved by the GNN-based autoencoder. Furthermore, the learning of node representations is

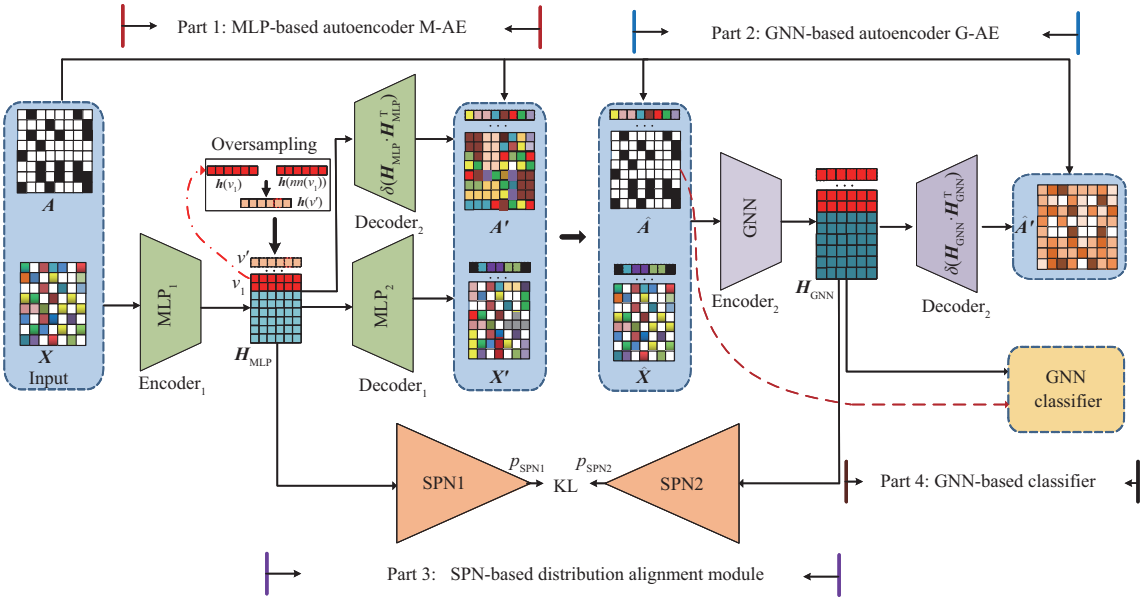

Figure 2 (Color online) The architecture of Graph-DAO. $\boldsymbol{A}$, $\boldsymbol{X}$, $\boldsymbol{A}'$, $\boldsymbol{X}'$, $\hat{\boldsymbol{A}}$, $\hat{\boldsymbol{X}}$, and $\hat{\boldsymbol{A}}'$ denote the adjacency matrix, feature matrix, reconstructed adjacency and feature matrix by M-AE, reconstructed adjacency and feature matrix with origin input, reconstructed adjacency matrix by G-AE, respectively. $\boldsymbol{H}_{\mathrm{MLP}}$ denotes the representation matrix by M-AE. $\boldsymbol{H}_{\mathrm{GNN}}$ denotes the representation matrix by G-AE. And KL denotes the Kullback-Leibler divergence.

constrained by the reconstruction decoder. (3) SPN-based distribution alignment module based on KL divergence is utilized to align the distributions of node representations obtained by M-AE and G-AE. In this regard, the distribution of the representations is obtained using the SPN method. (4) The node representation obtained by the GNN-based autoencoder and the new graph structure in $\hat{\boldsymbol{G}}$ are fed into the GNN classifier to obtain the results of node classification.

4.2 The proposed model

The four parts (MLP-based autoencoder M-AE, GNN-based autoencoder G-AE, SPN-based distribution alignment module, and GNN-based classifier) of Graph-DAO will be introduced in detail.

4.2.1 *MLP-based autoencoder M-AE*

In most existing graph oversampling methods, the oversampling procedure following the encoding of GNN tends to bias the representations of minority nodes (including labeled synthetic minority nodes) towards majority nodes. This results in confusion in the node classification of the graphs. Therefore, we propose an autoencoder based on MLP to obtain the synthetic minority nodes and their connections. This approach enables the representations of synthetic minority nodes to be obtained without the need for aggregating neighboring information, thereby mitigating the problem of minority nodes' representations and features being biased towards those of the majority nodes.

Specifically, we take the feature matrix $\boldsymbol{X}$ as input and learn the node representations through the MLP encoder as

$$\boldsymbol{H}_{\mathrm{MLP}} = \mathrm{MLP}_1(\boldsymbol{X}), \tag{2}$$

where $\boldsymbol{H}_{\mathrm{MLP}} = \{\boldsymbol{h}_{m_1}, \boldsymbol{h}_{m_2}, \ldots, \boldsymbol{h}_{m_n}\}$ represents the node representation matrix. $\boldsymbol{h}_{m_n}$ represents the representation of node n.

After obtaining the node representations by MLP encoder, the synthetic minority nodes are generated in the latent space utilizing oversampling techniques Mixup [52]. The Mixup can be formulated as

$$\boldsymbol{h}_{m_{v'}} = (1-\gamma)\cdot \boldsymbol{h}_{m_v} + \gamma \cdot \boldsymbol{h}_{m_{nn(v)}}, \tag{3}$$

$$nn(v) = \underset{u\in\{\boldsymbol{V}/v\}, y_u = y_v}{\operatorname{argmin}} \left\| \boldsymbol{h}_{m_u} - \boldsymbol{h}_{m_v} \right\|, \tag{4}$$

where v' is the synthetic minority node. We conduct interpolation on a given sample v belonging to the target minority class, by utilizing its nearest neighbor $nn(v)$. The nearest neighbors $nn(v)$ are obtained

using the Euclidean distance metric in the node representation space to identify the nearest neighbors that belong to the same class as (4). γ is a random variable that conforms to a uniform distribution between the range of 0 and 1. y_u and y_v denote the labels of nodes u and v, respectively. The synthetic minority node v' has the same label as the minority node v. We utilize a hyperparameter known as the oversampling scale to regulate the number of synthetic minority nodes for each minority class.

Subsequently, a dual decoder is utilized to generate the feature matrix $\boldsymbol{X'}$ and adjacency matrix $\boldsymbol{A'}$ that contain synthetic minority nodes. Specifically, we obtain the adjacency matrix $\boldsymbol{A'}$ by

$$p(\boldsymbol{A'}|\boldsymbol{H}_{\mathrm{MLP}}) = \prod_{i=1}^{n}\prod_{j=1}^{n} p(a'_{ij}|\boldsymbol{h}_{m_i}, \boldsymbol{h}_{m_j}), \tag{5}$$

$$p(a'_{ij} = 1|\boldsymbol{h}_{m_i}, \boldsymbol{h}_{m_j}) = \delta(\boldsymbol{h}_{m_i}^{\mathrm{T}} \boldsymbol{h}_{m_j}), \tag{6}$$

where a'_{ij} refers to the elements of $\boldsymbol{A'}$, and $\delta(\cdot)$ represents the sigmoid function. For the edges of the synthetic node, we use soft edges, that is

$$a'_{v'u} = e_{v',u}, \tag{7}$$

where $e_{v',u} = p(a'_{v'u} = 1|\boldsymbol{h}_{m_{v'}}, \boldsymbol{h}_{m_u}) = \delta(\boldsymbol{h}_{m'_v}^{\mathrm{T}} \boldsymbol{h}_{m_u})$ represents the soft edge between node v' and u.

The loss function of the edge predictor is

$$L_A = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{A}_i - \boldsymbol{A}'_i\|_2^2. \tag{8}$$

We also reconstruct the feature matrix $\boldsymbol{X'}$ by an MLP decoder:

$$\boldsymbol{X'} = \mathrm{MLP}_2(\boldsymbol{H}_{\mathrm{MLP}}). \tag{9}$$

The loss function of the feature predictor is

$$L_X = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{X}_i - \boldsymbol{X}'_i\|_2^2. \tag{10}$$

4.2.2 *GNN-based autoencoder G-AE*

In the GNN-based autoencoder stage, the newly synthetic minority nodes, features, and edges obtained from the autoencoder (M-AE) are integrated into the original graph $\boldsymbol{G}_{\mathrm{im}}$, resulting in a new graph $\hat{\boldsymbol{G}} = (\hat{\boldsymbol{V}}, \hat{\boldsymbol{C}}, \boldsymbol{F}, \hat{\boldsymbol{E}})$, where $\hat{\boldsymbol{V}} = \boldsymbol{V} \cup \boldsymbol{V}_s$, $\hat{\boldsymbol{C}} = \boldsymbol{C} \cup \boldsymbol{C}_s$, $\hat{\boldsymbol{X}} = \boldsymbol{X} \cup \boldsymbol{X}_s$, $\hat{\boldsymbol{X}}[:N, :N] = \boldsymbol{X}$, $\hat{\boldsymbol{E}} = \boldsymbol{E} \cup \boldsymbol{E}_s$, and $\hat{\boldsymbol{A}}[:N, :N] = \boldsymbol{A}$. The $\boldsymbol{V}_s$, $\boldsymbol{C}_s$, $\boldsymbol{X}_s$, and $\boldsymbol{E}_s$ represent node set, class set, feature set, and edge set of synthetic minority nodes, respectively. The $\boldsymbol{X}_s$ and $\boldsymbol{E}_s$ are reconstructed by autoencoder M-AE. Then, we take the $\hat{\boldsymbol{G}} = (\hat{\boldsymbol{V}}, \hat{\boldsymbol{C}}, \boldsymbol{F}, \hat{\boldsymbol{E}})$ as the input of GNN encoder:

$$\boldsymbol{H}_{\mathrm{GNN}} = \mathrm{GNN}(\hat{\boldsymbol{X}}, \hat{\boldsymbol{A}}), \tag{11}$$

where $\boldsymbol{H}_{\mathrm{GNN}} = \{\boldsymbol{h}_{g_1}, \boldsymbol{h}_{g_2}, \ldots, \boldsymbol{h}_{g_n}\}$ represents the node representation matrix. $\boldsymbol{h}_{g_n}$ represents the representation of node n. we utilize a single GraphSAGE layer to extract node representations. The message passing and fusing process of GraphSAGE can be expressed as

$$\boldsymbol{h}_{g_v} = \psi(\boldsymbol{W} \cdot \mathrm{CONCAT}(\hat{\boldsymbol{X}}[v, :], \hat{\boldsymbol{X}} \cdot \hat{\boldsymbol{A}}[:, v])), \tag{12}$$

where $\hat{\boldsymbol{X}}[v, :]$ is the feature of node v. $\hat{\boldsymbol{A}}[:, v]$ represents the v-th column of $\hat{\boldsymbol{A}}$, and $\boldsymbol{h}_{g_v}$ is the representation of node v. $\boldsymbol{W}$ represents the parameters of the weights, and ψ represents the activation function ReLU. Furthermore, the learning of node representations of $\hat{\boldsymbol{G}}$ is constrained by the reconstruction decoder. In the decoder stage of GNN-based autoencoder, The adjacency matrix $\hat{\boldsymbol{A}}'$ is reconstructed by

$$p(\hat{\boldsymbol{A}}'|\boldsymbol{H}_{\mathrm{GNN}}) = \prod_{i=1}^{n}\prod_{j=1}^{n} p(\hat{a}'_{ij}|\boldsymbol{h}_{g_i}, \boldsymbol{h}_{g_j}), \tag{13}$$

$$p(\hat{a}'_{ij} = 1|\boldsymbol{h}_{g_i}, \boldsymbol{h}_{g_j}) = \delta(\boldsymbol{h}_{g_i}^{\mathrm{T}} \boldsymbol{h}_{g_j}), \tag{14}$$

where $\hat{a}'_{ij}$ represents the elements of $\hat{\boldsymbol{A}}'$. The loss function is

$$L_{\hat{A}} = \frac{1}{n}\sum_{i=1}^{n} \left\|\boldsymbol{A}_i - \hat{\boldsymbol{A}}'_i\right\|_2^2. \tag{15}$$

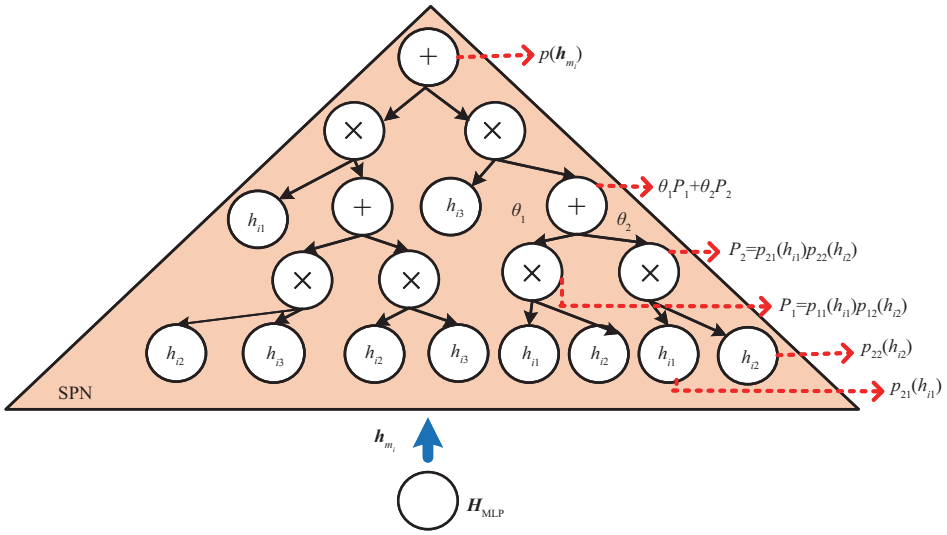

Figure 3 (Color online) SPN instantiation for node i. The symbols $\times$ and $+$ denote the product and sum node, respectively. The latent factors h_{i1}, h_{i2}, and h_{i3} correspond to the elements of the node representation.

4.2.3 *SPN-based distribution alignment module*

Graph-DAO uses SPN-based distribution alignment as a constraint term to mitigate the distribution mismatch of the representations resulting from GNN's aggregation of neighbor information. We utilize KL-divergence to perform distributional alignment between the M-AE representations $\boldsymbol{H}_{\mathrm{MLP}}$ and G-AE representations $\boldsymbol{H}_{\mathrm{GNN}}$. To obtain the distribution of node representation, we introduce the SPN due to its capability to learn distribution from data without requiring manual specification of the distribution type that the data follows, thus circumventing potential errors stemming from a priori distribution definition.

We take $\boldsymbol{H}_{\mathrm{MLP}}$ as the input of SPN, and model the joint probability distribution $p_{\mathrm{SPN1}}(\boldsymbol{H}_{\mathrm{MLP}};\boldsymbol{\theta})$ of $\boldsymbol{H}_{\mathrm{MLP}}$. The distribution of the SPN is obtained by the negative log-likelihood loss function:

$$L_S = -\frac{1}{n}\sum_{i=1}^{n} \log p_{\mathrm{SPN1}}(\boldsymbol{h}_{m_i};\boldsymbol{\theta}), \tag{16}$$

where $\boldsymbol{h}_{m_i} = (h_{m_{i1}}, \ldots, h_{m_{id}})^{\mathrm{T}}$ is the representation of node v_i. $\boldsymbol{\theta}$ represents the parameter set of SPN.

The computation of the joint distribution is performed by starting from the leaf nodes and proceeding upward in a bottom-up fashion, as depicted in Figure 3. The input of the leaf nodes is given by $\boldsymbol{h}_{m_i}$. The leaf nodes are modeled using Gaussian distributions.

We employ the distribution $p_{\mathrm{SPN1}}(\boldsymbol{H}_{\mathrm{MLP}};\boldsymbol{\theta})$ as the prior distribution, as this distribution is not subject to bias induced by aggregated neighbor information. Furthermore, we align the distribution $p_{\mathrm{SPN2}}(\boldsymbol{H}_{\mathrm{GNN}};\boldsymbol{\theta})$, learned from another SPN, with the distribution $p_{\mathrm{SPN1}}(\boldsymbol{H}_{\mathrm{MLP}};\boldsymbol{\theta})$ via (17).

$$\begin{aligned} L_{\mathrm{KL}} &= \mathrm{KL}(p_{\mathrm{SPN1}}(\boldsymbol{H}_{\mathrm{MLP}}) \,||\, p_{\mathrm{SPN2}}(\boldsymbol{H}_{\mathrm{GNN}})) \\ &= \sum p_{\mathrm{SPN2}}(\boldsymbol{H}_{\mathrm{GNN}}) \log \frac{p_{\mathrm{SPN1}}(\boldsymbol{H}_{\mathrm{MLP}})}{p_{\mathrm{SPN2}}(\boldsymbol{H}_{\mathrm{GNN}})}, \end{aligned} \tag{17}$$

where $\mathrm{KL}(\cdot)$ is the Kullback-Leibler divergence, $p_{\mathrm{SPN1}}(\boldsymbol{H}_{\mathrm{MLP}})$ and $p_{\mathrm{SPN2}}(\boldsymbol{H}_{\mathrm{GNN}})$ are the distributions of node representations in M-AE and G-AE, respectively.

The superiority of using SPN can be illustrated from two aspects. (1) SPN can learn the distribution from data without requiring manual specification of the type of distribution that the data follows, thus circumventing potential errors stemming from a priori distribution definition. (2) The inference of SPN is tractable. Although PGMs can also learn the distribution from data, their inference is often intractable unless it is a simple problem or a specific type of model with limited expressiveness. Overall, using SPNs to learn the distribution of node representations for class-imbalanced graphs is appropriate.

4.2.4 *GNN-based classifier*

To perform node classification on $\hat{\boldsymbol{G}}$, we utilize GraphSAGE followed by a linear layer as classifier. The GNN-based classifier can be formulated as follows:

$$\boldsymbol{h}_v = \psi(\boldsymbol{W}^{(1)} \cdot \mathrm{CONCAT}(\boldsymbol{h}_{g_v}, \boldsymbol{H}_{\mathrm{GNN}} \cdot \hat{\boldsymbol{A}}[:, v])), \tag{18}$$

$$p_c = \mathrm{softmax}(\boldsymbol{W}^{(2)} \cdot \boldsymbol{h}_v), \tag{19}$$

where $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$ represent the weight parameters, p_c denotes the probability distribution of class labels for a given node v. The optimization of the GNN classifier is achieved through the utilization of the cross-entropy loss function.

$$L_C = \sum_{v \in \hat{\boldsymbol{V}}} \sum_{c} (\boldsymbol{1}(\hat{Y}_v == c) \cdot \log p_c[c]). \tag{20}$$

4.3 Training and optimization

We minimize the objective function through stochastic gradient descent. Given that the quality of both the node representations and the generated edges significantly impact the performance of the model, a two-stage training paradigm is adopted to enhance the stability of the training process. During the first stage, we perform pretraining of the feature extractor and edge predictor by minimizing the reconstruction loss, which is composed of L_X, L_A, and $L_{\hat{A}}$. Additionally, we introduce SPNs to learn the distribution of the representation $\boldsymbol{H}_{\mathrm{MLP}}$. The loss function for this stage is

$$L_{\mathrm{pre}} = L_X + L_A + L_{\hat{A}} + L_S, \tag{21}$$

where L_X, L_A, and $L_{\hat{A}}$ are the reconstruction loss of the node features, network structure in M-AE, and network structure in G-AE, respectively. L_S is the negative log-likelihood loss, which encourages the SPN to learn a better distribution in the M-AE.

In the second stage, we train the classification module by using a GNN classifier with partially labeled nodes as guidance. Furthermore, we address the distribution mismatch issue in GNN caused by the aggregation of neighbor information, by employing a KL divergence-based distribution alignment method. The loss function for this stage is

$$L_{\mathrm{node}} = L_C + L_{\mathrm{KL}}, \tag{22}$$

where L_{KL} is the distribution alignment loss, which encourages the representation distribution of the G-AE to match the representation distribution of the M-AE. L_C is the cross-entropy loss.

To sum up, the overall loss function of the whole Graph-DAO can be expressed as

$$L = L_{\mathrm{pre}} + L_{\mathrm{node}}. \tag{23}$$

5 Experimental analysis

In this section, we demonstrate that Graph-DAO is an efficient oversampling-based framework for class-imbalanced graphs. We concentrate on the semi-supervised node classification task. We first present the experimental setup, including the datasets, evaluation metrics, baselines, and parameter settings. Then, we conduct the evaluation experiments and provide a detailed analysis of the experimental results.

5.1 Experimental details

5.1.1 *Datasets*

To assess the effectiveness of our proposed model, we conduct experiments on four benchmark datasets as shown in Table 1. The details of them are as follows:

• **Citation networks.** The class distributions of both Cora [53] and Citeseer [4] are mild class-imbalanced (e.g., the class imbalance ratio ρ_c of Cora and Citeseer are 0.22 and 0.36, respectively). To further investigate the efficacy of our model, we take the step imbalance setting [54, 55], where we randomly select three classes as minority classes and down-sample them. All majority classes have the same labeling size of 20 nodes, and all minority classes have the same labeling size $\rho \times 20$, where the

Table 1 A summary of graph datasets

Dataset	Classes	Nodes	Edges	Features	Class imbalance ratio
Cora	7	2708	5429	1433	0.22
Citeseer	6	3327	4732	3703	0.36
BlogCatalog	36	10312	333983	64	0.004
Wiki-CS	10	11701	216123	300	0.11

imbalance ratio ρ represents the proportion of labeled nodes in the smallest class to the labeled nodes in the largest class. We vary the ρ to evaluate the performance of our model under diverse imbalanced scenarios. A lower value of ρ indicates a higher level of imbalance, followed by a more difficult classification task.

• **BlogCatalog.** Notably, the BlogCatalog [53] lacks node features. Hence we adopt the approach in [56], where each node is assigned a 64-dimensional representation vector obtained from Deepwalk. It is worth mentioning that the class distribution in this dataset is genuinely imbalanced, with 14 classes having less than 100 instances and 8 classes with more than 500 instances. For training and validation, we use 25% nodes in each class, while the remaining 50% is used for testing.

• **Wiki-CS.** The class distribution of Wiki-CS [57] is genuinely imbalanced. Classes that have a lower number of samples compared to the average are classified as minority classes. For training and validation, we use 25% samples of each class, while the remaining 50% is used for testing.

5.1.2 *Evaluation metrics*

In accordance with prior researches on evaluating class-imbalanced node classification [20, 23], we utilize three metrics, namely classification accuracy (ACC), mean area under the receiver operating characteristic curve (AUC-ROC), and F Score, to assess the efficacy of our proposed model. While ACC is a widely used metric that ensures effective classification of the majority classes, the model's effectiveness on the minority classes is inadequately reflected. Therefore, to better reflect the performance on the minority classes, we incorporate the F Score and AUC-ROC metrics. Specifically, AUC-ROC measures the area under the ROC curve, while F Score provides the harmonic mean of precision and recall for every class.

5.1.3 *Baselines*

We have selected two categories of baseline algorithms related to class-imbalanced algorithms, particularly those that have been recognized as state-of-the-art. (1) The representative approaches. (2) The oversampling methods of class-imbalanced graphs because our method centers on mitigating the limitations of oversampling techniques on graphs.

• **Vanilla** [15]**.** Original implementation of GNN without additional tricks.

• **Oversampling.** The process of increasing the number of samples in the minority classes by directly repeating samples from these classes.

• **Re-weight** [58]**.** This approach is a cost-sensitive technique that applies greater weightage to the loss of samples belonging to the minority classes.

• **SMOTE** [26]**.** The approach creates synthetic minority nodes by interpolating within the input space, while ensuring that the edges of these synthetic nodes match those of the source nodes.

• **Embed-SMOTE** [59]**.** It generates synthetic minority nodes by interpolating in the embedding space and sets the edges of the synthetic minority nodes to be the same as the source nodes.

• **DR-GCN** [21]**.** This algorithm employs adversarial training to enhance the discriminability of node representations across multiple classes.

• **GraphSMOTE** [20]**.** It employs an oversampling method to generate synthetic minority nodes and uses an edge predictor to establish connectivity relationships for the synthetic minority nodes.

• **GraphMixup** [23]**.** It constructs semantic relation space through the way of disentanglement and synthesizes minority class nodes by implementing the oversampling strategy at the semantic level. In addition, GraphMixup adaptively determines the number of synthetic minority nodes by using a reinforcement learning mechanism.

Furthermore, to validate the distribution alignment module in Graph-DAO, we conducted an ablation experiment using a variant called Graph-O, which removes the distribution alignment module.

Table 2 ACC, AUC-ROC, and F Score values for class-imbalanced node classification

Method	Cora			Citeseer		
	ACC	AUC-ROC	F Score	ACC	AUC-ROC	F Score
Vanilla	0.681±0.038	0.912±0.018	0.680±0.040	0.545±0.020	0.817±0.023	0.537±0.033
Oversampling	0.733±0.013	0.928±0.011	0.728±0.016	0.535±0.046	0.829±0.023	0.532±0.055
Re-weight	0.723±0.014	0.923±0.010	0.722±0.018	0.555±0.028	0.827±0.021	0.557±0.032
SMOTE	0.722±0.018	0.932±0.009	0.723±0.019	0.539±0.044	0.829±0.023	0.536±0.056
Embed-SMOTE	0.735±0.025	0.931±0.011	0.737±0.026	0.567±0.040	0.830±0.025	0.567±0.043
DR-GCN	0.725±0.094	0.932±0.026	0.722±0.081	0.551±0.083	0.831±0.053	0.558±0.094
GraphSMOTE	0.747±0.018	0.943±0.013	0.743±0.020	0.575±0.030	0.867±0.016	0.567±0.024
GraphMixup	0.775±0.011	0.948±0.006	0.774±0.011	0.585±0.042	0.843±0.026	0.583±0.042
Graph-O	0.796±0.010	0.953±0.007	0.797±0.009	0.632±0.046	0.884±0.029	0.630±0.047
Graph-DAO	**0.802±0.009**	**0.957±0.005**	**0.802±0.010**	**0.658±0.019**	**0.896±0.016**	**0.656±0.023**
Method	BlogCatalog			Wiki-CS		
	ACC	AUC-ROC	F Score	ACC	AUC-ROC	F Score
Vanilla	0.218±0.011	0.601±0.009	0.101±0.009	0.763±0.021	0.942±0.013	0.734±0.009
Oversampling	0.214±0.009	0.598±0.014	0.102±0.008	0.767±0.020	0.956±0.007	0.741±0.021
Re-weight	0.210±0.010	0.598±0.011	0.099±0.007	0.768±0.016	0.957±0.003	0.738±0.018
SMOTE	0.213±0.009	0.601±0.010	0.101±0.006	0.774±0.008	0.958±0.004	0.747±0.007
Embed-SMOTE	0.204±0.011	0.596±0.009	0.095±0.008	0.770±0.011	0.954±0.005	0.740±0.008
DR-GCN	0.208±0.065	0.605±0.005	0.098±0.004	0.774±0.006	0.957±0.015	0.743±0.009
GraphSMOTE	0.248±0.009	0.651±0.005	0.125±0.004	0.789±0.013	0.958±0.024	0.756±0.023
GraphMixup	0.258±0.010	0.655±0.003	0.128±0.006	0.796±0.012	0.963±0.023	0.761±0.041
Graph-O	0.260±0.005	0.654±0.010	0.125±0.003	0.807±0.012	0.971±0.003	0.779±0.016
Graph-DAO	**0.263±0.002**	**0.657±0.007**	**0.130±0.003**	**0.809±0.008**	**0.972±0.002**	**0.780±0.010**

5.1.4 *Parameter settings*

The experimental evaluation is conducted using the PyTorch framework, with the ADAM optimizer [60] employed for optimization. All models are trained on an NVIDIA GeForce GTX 2080Ti GPU. Except for method DR-GCN [21], which employs the same experimental settings as described in the original paper, the settings of other baselines are obtained from the GraphMixup [23] open-source library[4)]. For our model, the hyperparameters are determined via grid search, with a dropout rate of 0.1 and a learning rate of 0.001. Furthermore, the weight decay is configured to 5e−4, with a maximum training epoch of 5000. Specifically, the layer number of the encoder for both M-AE and G-AE is set to 1, with a hidden dimension of 64. The layer number of the Decoder_1 for M-AE is set to 2. For Cora and Citeseer, the imbalance ratio is set to 0.5, and the oversampling scale is set to 1.0 unless otherwise specified. For Wiki-CS and BlogCatalog, the imbalance ratio is not considered, and the oversampling scale is set class-wise to ensure balance in the minority classes. The final results are obtained by averaging the outcomes of 5 runs.

5.2 Imbalanced node classification

The experimental results of Graph-DAO and the baseline methods on a node classification task are shown in Table 2. The best results achieved by the algorithms are highlighted in bold. Based on the results, we make the following observations. (1) Graph-DAO outperforms all baselines on all datasets, achieving the highest performance. For instance, on Cora, Graph-DAO outperforms GraphMixup by 2.7% in ACC, 0.9% in AUC-ROC, and 2.8% in F Score. This is because our method can obtain representations of minority nodes without the need to aggregate neighbor information, thereby alleviating the problem of minority nodes being biased towards majority nodes' representations. Additionally, Graph-DAO utilizes distribution alignment based on SPN as a constraint term, mitigating the issue of distribution mismatch in representations caused by GNN's aggregating neighbor information. (2) Graph-O, which removes the distribution alignment module, outperforms GraphMixup on all ACC, AUC-ROC, and F Score, except for the AUC-ROC and F Score on the BlogCatalog, where it performs comparable results to existing methods (AUC-ROC is 0.001 lower than GraphMixup, and F Score is 0.003 lower than GraphMixup). This observation underscores the effectiveness of our idea of oversampling before GNN. (3) Compared with

4) https://github.com/LirongWu/GraphMixup.

Table 3 Performance under different imbalance ratios

Method	Imbalance ratio on Cora			
	0.1	0.2	0.4	0.6
Oversampling	0.839±0.008	0.874±0.016	0.894±0.019	0.923±0.009
Re-weight	0.828±0.011	0.873±0.013	0.918±0.031	0.924±0.007
SMOTE	0.840±0.034	0.879±0.021	0.921±0.015	0.926±0.004
Embed-SMOTE	0.843±0.027	0.879±0.008	0.919±0.011	0.929±0.010
DR-GCN	0.863±0.014	0.899±0.013	0.926±0.010	0.931±0.032
GraphSMOTE	0.889±0.021	0.910±0.024	0.927±0.004	0.932±0.007
GraphMixup	0.893±0.031	0.913±0.012	0.932±0.013	0.934±0.015
Graph-O	0.943±0.011	0.951±0.010	0.955±0.008	0.959±0.011
Graph-DAO	**0.944±0.011**	**0.952±0.010**	**0.957±0.009**	**0.960±0.010**
Method	Imbalance ratio on Citeseer			
	0.1	0.2	0.4	0.6
Oversampling	0.738±0.013	0.760±0.009	0.805±0.016	0.835±0.021
Re-weight	0.738±0.012	0.745±0.014	0.786±0.027	0.835±0.019
SMOTE	0.741±0.012	0.767±0.004	0.814±0.020	0.838±0.021
Embed-SMOTE	0.732±0.010	0.758±0.014	0.807±0.016	0.815±0.024
DR-GCN	0.742±0.017	0.764±0.032	0.803±0.009	0.839±0.026
GraphSMOTE	0.747±0.006	0.776±0.011	0.813±0.015	0.822±0.012
GraphMixup	0.757±0.024	0.793±0.028	0.829±0.021	0.844±0.025
Graph-O	0.823±0.012	0.866±0.023	0.888±0.015	0.897±0.012
Graph-DAO	**0.825±0.035**	**0.870±0.016**	**0.888±0.011**	**0.898±0.013**

Graph-O, the node classification of Graph-DAO is improved after introducing the distribution alignment module. This finding demonstrates the effectiveness of the distribution alignment module. Summarily, the experimental results indicate that our proposed framework is superior for the node classification task.

5.3 Influence of imbalance ratio

To evaluate the robustness of Graph-DAO in relation to varying imbalance ratios, we maintain a fixed oversampling scale of 1.0 while varying the imbalance ratio across values of 0.1, 0.2, 0.4, and 0.6. We compare Graph-DAO and Graph-O with baselines on Cora and Citeseer. The experimental results of node classification under different imbalance ratios are illustrated in Table 3. Based on the results, we draw the following observations. (1) The performance of most algorithms improves with an increase in the imbalance ratio. As we mentioned in Subsection 5.1.1, the imbalance ratio ρ represents the proportion of labeled nodes in the smallest class to the labeled nodes in the largest class. As ρ increases, the graph becomes more balanced in terms of node class, thereby enhancing the overall effectiveness of the algorithms in the node classification task. (2) As the imbalance ratio increases, the performance of node classification for Graph-DAO consistently outperforms the other comparison methods. These results verify the robustness of our framework in relation to varying imbalance ratios. (3) The improvement achieved by Graph-DAO is more substantial when the extent of the imbalance is more extreme. For instance, in the Cora, when the imbalance ratio is 0.1, Graph-DAO outperforms GraphMixup by 5.1%, while the gap narrows to 2.6% when the imbalance ratio reaches 0.6. This finding proves that our framework is more effective for severe imbalance cases.

5.4 Influence of oversampling scale

To assess the effectiveness of Graph-DAO using different oversampling scales, we maintain a fixed imbalance ratio of 0.5 while varying the oversampling scale across values of 0.2, 0.4, 0.6, 0.8, 1.0, and 1.2. We compare Graph-DAO with the baselines on the Cora and Citeseer. The node classification results under different oversampling scales are illustrated in Figure 4. We arrive at the following observations. (1) The node classification performance under different oversampling scales of Graph-DAO is superior to that of other baselines. Moreover, our method yields more stable results compared to other baselines. Compared with other baselines, Graph-DAO shows more significant improvement when the oversampling scale is small. Because our framework alleviates the problem of majority nodes dominating the representation of

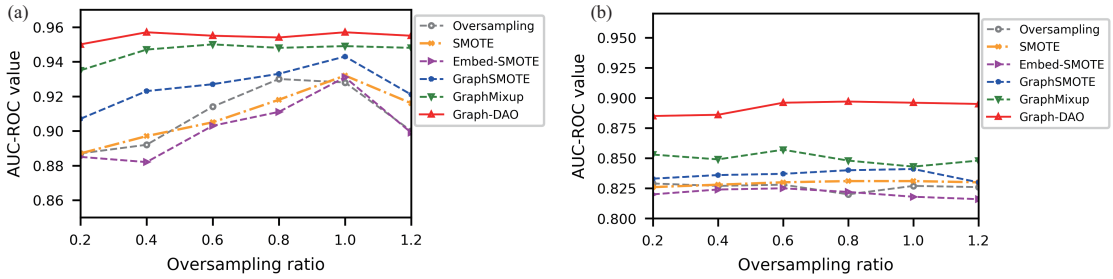

Figure 4 (Color online) AUC-ROC values for different oversampling scales. (a) Cora; (b) Citeseer.

Table 4 Performance comparison of algorithms with GCN as the base model

Method	Cora			Citeseer		
	ACC	AUC-ROC	F Score	ACC	AUC-ROC	F Score
Vanilla	0.667±0.025	0.904±0.011	0.664±0.029	0.517±0.023	0.801±0.014	0.516±0.012
Oversampling	0.651±0.038	0.906±0.017	0.647±0.042	0.501±0.003	0.803±0.010	0.496±0.008
Re-weight	0.690±0.033	0.912±0.017	0.688±0.037	0.523±0.024	0.810±0.021	0.533±0.015
SMOTE	0.700±0.021	0.920±0.006	0.701±0.022	0.502±0.009	0.804±0.021	0.500±0.026
Embed-SMOTE	0.726±0.027	0.919±0.010	0.728±0.028	0.518±0.042	0.791±0.044	0.524±0.036
GraphSMOTE	0.732±0.020	0.921±0.006	0.730±0.021	0.521±0.033	0.804±0.018	0.523±0.029
GraphMixup	0.756±0.033	0.930±0.015	0.757±0.032	0.527±0.016	0.811±0.019	0.529±0.013
Graph-O	0.765±0.030	0.931±0.007	0.763±0.031	0.524±0.021	0.808±0.012	0.530±0.036
Graph-DAO	**0.766±0.026**	**0.932±0.009**	**0.764±0.026**	**0.535±0.031**	**0.813±0.023**	**0.537±0.025**

minority nodes by oversampling before GNN and distribution alignment method. (2) When the oversampling scale is increased, adverse effects may arise. We observe that the performance remains constant or slightly degrades when increasing the oversampling scale from 1.0 to 1.2. This is due to the generation of too many synthetic nodes with redundant information. In summary, we observe that an oversampling scale between 0.8 and 1.0 is preferable, and we adopt a scale of 1.0 for our experiments.

5.5 Influence of base model

We evaluate the generalization capability of our framework by implementing it on another commonly used GNN, i.e., GCN. All methods in this section, including baselines and our proposed method, are implemented on GCN, and their performances are shown in Table 4. As shown in Table 4, Graph-DAO outperforms the baselines on the node classification task with GCN. Compared to GraphSMOTE and GraphMixup, Graph-DAO is highly competitive. Moreover, compared to Vanilla, Graph-DAO achieves approximately 10% improvement in ACC and F Score on Cora, which underscores the efficacy of our approach with GCN for the node classification task.

Totally, we use two base models, i.e., GraphSAGE and GCN, to show the performance of our proposed method in this paper. The corresponding results are shown in Tables 2 and 4. From these two tables, we can summarize two conclusions. (1) Graph-DAO performs best compared to the baselines, whether using GraphSAGE or GCN. The reasons are discussed in Subsection 5.2. (2) GraphSAGE-based models outperform GCN-based models for both Graph-DAO and baselines. In imbalanced scenarios, minority nodes have many majority neighbors. By using more neighbor information, minority-class nodes' representations become similar to the majority-class nodes, leading to unexpected results. Compared with GCN using all neighbor information, GraphSAGE samples local neighbors and aggregates their information, enhancing model performance. Moreover, GraphSAGE surpasses GCN in terms of node representation capabilities and flexibility in modeling relationship information, thereby acquiring superior node representations [20].

5.6 Graph visualization

To obtain a more comprehensive understanding of the node representations produced by Graph-DAO and baselines, we employ the t-SNE [61] to visualize the Cora. Specifically, we reduce the dimensionality of the node representations to a two-dimensional space for visualization purposes. The perplexity of t-SNE

Figure 5 (Color online) Visualization on the Cora dataset. (a) Vanilla; (b) GraphSMOTE; (c) GraphMixup; (d) Graph-DAO.

is set to 30 with 5000 iterations. Figure 5 displays the visualization results. We observe that the Graph-DAO has clear advantages. The classic Vanilla approach fails to distinguish nodes belonging to different categories in the class-imbalance graph. Compared to our framework, GraphSMOTE and GraphMixup struggle to distinguish different classes of nodes effectively. For instance, red category nodes are visibly mixed with other category nodes in these methods, whereas our method distinguishes them clearly. This is because our proposed method mitigates the problem of majority nodes dominating the representation of minority nodes by oversampling before GNN and the distribution alignment method.

6 Conclusion and future work

In this article, we propose a novel oversampling framework for class-imbalanced graphs named Graph-DAO. It can alleviate the problem that the encoding of the GNN tends to bias the representations of the minority nodes, including the labeled synthetic minority nodes, towards the majority nodes in the class imbalance scenarios by oversampling before GNN. In addition, a distribution alignment method based on SPNs is introduced, which mitigates the risk of distributional bias in the representations. We perform comprehensive experiments on node classification tasks and demonstrate that our proposed model, Graph-DAO, outperforms the baseline models in terms of performance.

There are several interesting directions for further research. First, we currently focus on node classification tasks. However, other tasks, such as graph classification and link prediction, also suffer from class imbalance. In addition, our framework cannot be used in very large networks due to the limitation of structure learning in SPNs. Therefore, we think it is an interesting direction to extend our framework to large graph datasets as well as to other types of tasks.

Acknowledgements This work was supported by National Key R&D Program of China (Grant No. 2021ZD0112500), National Natural Science Foundation of China (Grant Nos. U22A2098, 62172185, 62202200, 62206105), and Fundamental Research Funds for the Central Universities, JLU.

References

1 Li M, Zhu Z. Spatial-temporal fusion graph neural networks for traffic flow forecasting. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2021. 4189–4196
2 Wang Z, Wang C, Gao C, et al. An evolutionary autoencoder for dynamic community detection. Sci China Inf Sci, 2020, 63: 212205
3 Romanou A, Smeros P, Aberer K. On representation learning for scientific news articles using heterogeneous knowledge graphs. In: Proceedings of the Web Conference, Ljubljana, 2021. 422–425
4 Sen P, Namata G, Bilgic M, et al. Collective classification in network data. AI Mag, 2008, 29: 93–106
5 Mohammadrezaei M, Shiri M E, Rahmani A M. Identifying fake accounts on social networks based on graph analysis and classification algorithms. Secur Commun Netw, 2018, 2018: 1–8
6 Xu K, Hu W, Leskovec J, et al. How powerful are graph neural networks? In: Proceedings of the Learning Representations, New Orleans, 2019
7 Ye Y, Ji S. Sparse graph attention networks. IEEE Trans Knowl Data Eng, 2023, 35: 905–916
8 Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks. In: Proceedings of the Learning Representations, Toulon, 2017
9 Guo L, Tang L, Chen T, et al. DA-GCN: a domain-aware attentive graph convolution network for shared-account cross-domain sequential recommendation. In: Proceedings of the 30th International Joint Conference on Artificial Intelligence, Montreal, 2021. 2483–2489
10 Chen H, Zhuang F, Xiao L. AMA-GCN: adaptive multi-layer aggregation graph convolutional network for disease prediction. In: Proceedings of the 30th International Joint Conference on Artificial Intelligence, Montreal, 2021. 2235–2241
11 Eliasof M, Haber E, Treister E. PDE-GCN: novel architectures for graph neural networks motivated by partial differential equation. In: Proceedings of the Advances in Neural Information Processing Systems, 2021. 3836–3849
12 You H, Lu Z, Zhou Z, et al. Early-bird GCNs: graph-network co-optimization towards more efficient GCN training and inference via drawing early-bird lottery tickets. In: Proceedings of the 34th Conference on Innovative Applications of Artificial Intelligence, 2022. 8910–8918

13 Li S, Li W T, Wang W. CO-GCN for multi-view semi-supervised learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, New York, 2020. 4691–4698
14 Tian L, Wu H. MI-GCN: node mutual information-based graph convolutional network. In: Proceedings of the Web Conference, Lyon, 2022. 996–1003
15 Hamilton W L, Ying R, Leskovec J. Inductive representation learning on large graphs. In: Proceedings of the Advances in Neural Information Processing Systems, Long Beach, 2017. 1024–1034
16 Yin J, Gan C, Zhao K, et al. A novel model for imbalanced data classification. In: Proceedings of the AAAI Conference on Artificial Intelligence, New York, 2020. 6680–6687
17 Menon A K, Jayasumana S, Rawat A S, et al. Long-tail learning via logit adjustment. In: Proceedings of the Learning Representations, 2021
18 Chen D, Lin Y, Zhao G, et al. Topology-imbalance learning for semi-supervised node classification. In: Proceedings of the Advances in Neural Information Processing Systems, 2021. 29885–29897
19 Park J, Song J, Yang E. GraphENS: neighbor-aware ego network synthesis for class-imbalanced node classification. In: Proceedings of the Learning Representations, 2022
20 Zhao T, Zhang X, Wang S. GraphSMOTE: imbalanced node classification on graphs with graph neural networks. In: Proceedings of the Web Search and Data Mining, 2021. 833–841
21 Shi M, Tang Y, Zhu X, et al. Multi-class imbalanced graph convolutional network learning. In: Proceedings of the 29th International Joint Conference on Artificial Intelligence, 2020. 2879–2885
22 Li X, Wen L, Deng Y, et al. Graph neural network with curriculum learning for imbalanced node classification. 2022. arXiv:2202.02529
23 Wu L, Xia J, Gao Z, et al. GraphMixup: improving class-imbalanced node classification on graphs by self-supervised context prediction. In: Proceedings of the Machine Learning and Knowledge Discovery in Databases: European Conference, Grenoble, 2022. 519–535
24 Song J, Park J, Yang E. TAM: topology-aware margin loss for class-imbalanced node classification. In: Proceedings of the Machine Learning, Baltimore, 2022. 20369–20383
25 Wang K, An J, Zhou M, et al. Minority-weighted graph neural network for imbalanced node classification in social networks of internet of people. IEEE Int Things J, 2023, 10: 330–340
26 Chawla N V, Bowyer K W, Hall L O, et al. SMOTE: synthetic minority over-sampling technique. J Artif Intell Res, 2002, 16: 321–357
27 Wang Z, Ye X, Wang C, et al. RSDNE: exploring relaxed similarity and dissimilarity from completely-imbalanced labels for network embedding. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, New Orleans, 2018. 475–482
28 Wang Y. Fair graph representation learning with imbalanced and biased data. In: Proceedings of the 15th ACM International Conference on Web Search and Data Mining, 2022. 1557–1558
29 Ghorbani M, Kazi A, Baghshah M S, et al. RA-GCN: graph convolutional network for disease prediction problems with imbalanced data. Med Image Anal, 2022, 75: 102272
30 Huang Z, Tang Y, Chen Y. A graph neural network-based node classification model on class-imbalanced graph data. Knowl-Based Syst, 2022, 244: 108538
31 Shi S, Qiao K, Yang S, et al. AdaGCN: adaptive boosting algorithm for graph convolutional networks on imbalanced node classification. 2022. arXiv:2105.11625
32 Qian Y, Zhang C, Zhang Y, et al. Co-modality graph contrastive learning for imbalanced node classification. In: Proceedings of the Advances in Neural Information Processing Systems, 2022
33 Qu L, Zhu H, Zheng R, et al. ImGAGN: imbalanced network embedding via generative adversarial graph networks. In: Proceedings of the Conference on Knowledge Discovery and Data Mining, 2021. 1390–1398
34 Poon H, Domingos P. Sum-product networks: a new deep architecture. In: Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence, Barcelona, 2011. 337–346
35 Gens R, Domingos P. Learning the structure of sum-product networks. In: Proceedings of the Machine Learning, Atlanta, 2013. 873–880
36 Sánchez-Cauce R, París I, Díez F. Sum-product networks: a survey. IEEE Trans Pattern Anal Mach Intell, 2022, 44: 3821–3839
37 Peharz R, Vergari A, Stelzner K, et al. Random sum-product networks: a simple and effective approach to probabilistic deep learning. In: Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence, Tel Aviv, 2019. 334–344
38 Peharz R, Vergari A, Stelzner K, et al. Einsum networks: fast and scalable learning of tractable probabilistic circuits. In: Proceedings of the 37th International Conference on Machine Learning, 2020. 7563–7574
39 Dennis A, Ventura D. Learning the architecture of sum-product networks using clustering on variables. In: Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, 2012. 3239–3247
40 Papastavridis, John G. Tensor Calculus and Analytical Dynamics. New York: CRC Press, 1988
41 Nath A, Domingos P M. Learning relational sum-product networks. In: Proceedings of the National Conference on Artificial Intelligence, Austin, 2015. 2878–28867
42 Zheng K, Pronobis A, Rao R. Learning graph-structured sum-product networks for probabilistic semantic maps. In: Proceedings of the National Conference on Artificial Intelligence, New Orleans, 2018. 4547–4555
43 Xia R, Zhang Y, Zhang C. Multi-head variational graph autoencoder constrained by sum-product networks. In: Proceedings of the ACM Web Conference, Austin, 2023. 641–650
44 Kipf T N, Welling M. Variational graph auto-encoders. In: Proceedings of the Advances in Neural Information Processing Systems, 2017. 1–3
45 Koller D, Friedman N. Probabilistic Graphical Models: Principles and Techniques. Cambridge: MIT Press, 2009
46 Romanou A, Smeros P, Aberer K. On representation learning for scientific news articles using heterogeneous knowledge graphs. In: Proceedings of the ACM Web Conference, 2021. 422–425
47 Wang Z, Wang C, Li X, et al. Evolutionary Markov dynamics for network community detection. IEEE Trans Knowl Data Eng, 2022, 34: 1206–1220
48 Tian Y, Zhang C, Guo Z. Recipe2Vec: multi-modal recipe representation learning with graph neural networks. In: Proceedings of the 31st International Joint Conference on Artificial Intelligence, Vienna, 2022. 3473–3479
49 Chen Z, Chen F, Zhang L, et al. Bridging the gap between spatial and spectral domains: a survey on graph neural networks. 2020. doi: 10.1145/3627816
50 Wang X, Zhang M. How powerful are spectral graph neural networks. In: Proceedings of the Machine Learning, Baltimore, 2022. 23341–23362

51 Zhu H, Koniusz P. Simple spectral graph convolution. In: Proceedings of the Learning Representations, 2021
52 Verma V, Lamb A, Beckham C, et al. Manifold mixup: better representations by interpolating hidden states. In: Proceedings of the 36th International Conference on Machine Learning, Long Beach, 2019. 6438–6447
53 Lu Q, Getoor L. Link-based classification. machine learning. In: Proceedings of the 20th International Conference, Washington, 2003. 496–503
54 Buda M, Maki A, Mazurowski M A. A systematic study of the class imbalance problem in convolutional neural networks. Neural Netw, 2018, 106: 249–259
55 Cao K, Wei C, Gaidon A, et al. Learning imbalanced datasets with label-distribution-aware margin loss. In: Proceedings of Annual Conference on Neural Information Processing Systems, 2019. 1565–1576
56 Perozzi B, Al-Rfou R, Skiena S. Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, 2017. 701–710
57 Mernyei P, Cangea C. Wiki-CS: a wikipedia-based benchmark for graph neural networks. In press, 2022. arXiv:2007.02901
58 Yuan B, Ma X. Sampling + reweighting: boosting the performance of adaBoost on imbalanced datasets. In: Proceedings of the International Joint Conference on Neural Networks, Brisbane, 2012. 1–6
59 Ando S, Huang C Y. Deep over-sampling framework for classifying imbalanced data. In: Proceedings of the Machine Learning and Knowledge Discovery in Databases: European Conference, Skopje, 2017. 770–785
60 Khan A H, Cao X, Li S, et al. BAS-ADAM: an ADAM based approach to improve the performance of beetle antennae search optimizer. IEEE CAA J Autom Sin, 2020, 7: 461–471
61 Shen X, Zhu X, Jiang X, et al. Visualization of non-metric relationships by adaptive learning multiple maps t-SNE regularization. In: Proceedings of the IEEE BigData, Boston, 2017. 3882–3887