Location-Adaptive Generative Graph Augmentation for Fraud Detection

Lin Meng*, Xiaonan Zhang†, Jiawei Zhang‡ and Philip S. Yu§
*†Department of Computer Science, Florida State University, Tallahassee, FL, USA
‡Department of Computer Science, University of California, Davis, Davis, CA, USA
§Department of Computer Science, University of Illinois, Chicago, Chicago, IL, USA
*lm18h@fsu.edu, †xzhang@cs.fsu.edu, ‡jiawei@ifmlab.org, §psyu@uic.edu

2023 IEEE 5th International Conference on Cognitive Machine Intelligence (CogMI) | 979-8-3503-2383-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/COGMI58952.2023.00014

***Abstract*—Class imbalance is a well-recognized challenge in GNN-based fraud detection. Traditional methods like re-sampling and re-weighting address this issue by balancing class distribution. However, node class balancing with simple re-sampling or re-weighting may significantly distort the data distributions and eventually lead to the ineffective performance of GNNs. In this paper, we propose a novel approach named Location-Adaptive Generative Node Augmentation (LAGA), which improves GNN-based fraud detection models by augmenting synthetic nodes of the minority class. To increase the variety for the synthetic nodes, LAGA utilizes the GAN framework to synthesize node features and related edges of fake fraudulent nodes. Specifically, LAGA include feature generation and edge generation modules. In order to make sure the generated features are consistent in terms of representing fraudulent nodes, we also design a location network that uses node features to find the neighborhood of synthetic nodes. Our empirical results on two real-world fraud datasets demonstrate that LAGA improves the performance of GNN-based fraud detection models by a large margin with much fewer nodes than traditional class balance methods, and outperforms recent graph augmentation methods with the same number of synthetic nodes.**

***Index Terms*—Graph Neural Networks, Fraud Detection, Node Augmentation, Generative Adversarial Network**

I. Introduction

Fraud detection has become increasingly crucial due to a surge in fraudsters stealing unsuspecting individuals for personal gain [6], [12]. Such deceptive actions manifest in diverse ways. For instance, there is the distortion of consumers' shopping decisions via the spread of deceptive reviews by opinion fraudsters [3], [29] and financial deceptions linked to credit cards [13] and illicit money laundering activities [9]. In this domain, graphs serve as valuable tools. They show pivotal information useful in fraudulent entities and activities. A typical graph contains both benign and fraudulent participants and the connections between them. Thus, fraud detection is often regarded as a node classification task [14], [23], [29]. Here, the mission is to categorize nodes (for instance, users) within a graph as either benign or fraudulent. The rise of graph neural networks (GNNs) [7], [25] has introduced innovative methods to tackle graph data, accelerating the process of integrating GNNs into fraud detection [3], [16], [23], [27], [29].

In real-world fraud detection, datasets commonly suffer from class imbalance problem, where the number of fraudulent entities is significantly fewer than the benign ones [1]. As a result, GNN-based fraud detection models often lean towards predicting the predominant class, in other words, boosting accuracy and simultaneously diminishing recall. Traditional strategies to mitigate the class imbalance for i.i.d. data, including re-sampling [13], [18] and re-weighting techniques [3], [23], find limited applicability in graph data due to the inherent interconnectedness of its nodes. Specifically, under-sampling decreases the training data for the majority class, and over-sampling not only replicates nodes but also their associated edges within the minority class. This can lead to overfitting because of "neighborhood memorization" [19]. Re-weighting, on the other hand, by merely adjusting weights in the loss function, tends to overlook the graph edges. This oversight disrupts the information exchange between neighboring nodes, potentially causing overfitting in the minority class and underfitting in the majority class [19]. To deal with these challenges, more recent studies [5], [19], [30] suggest the augmentation of the minority class using node synthesis, aiming at balancing class distribution in graph data.

However, node class balancing techniques using mere re-sampling or re-weighting can significantly distort data distributions, which in turn may adversely affect the effectiveness of GNNs. While node synthesis techniques can improve performance, they often neglect task-specific information during node generation. For fraud detection, a better approach would be to generate node features and neighborhoods that have the structural characteristics of fraudsters.

In this paper, we introduce the **L**ocation-**A**daptive **G**enerative Graph **A**ugmentation (LAGA) model. This model synthesizes nodes and their associated edges for the minority class, with the objective of enhancing GNN-based fraud detection performance. At its core, LAGAadopts a generative adversarial network (GAN) framework, comprising feature and edge generators along two discriminators. The feature generator generates node features, ensuring they do not merely interpolate features of existing nodes. The edge generator generates edges for these synthetic nodes, which connect to nodes and their neighborhood searched by a location network. Different from conventional GANs, which employ a single discriminator to differentiate between real and fake nodes, our model incorporates an additional discriminator, which determines if nodes are fraudulent or benign, thereby reducing noise

979-8-3503-2383-2/23/$31.00 ©2023 IEEE
DOI 10.1109/CogMI58952.2023.00014

generation. Instead of balancing class distribution, our method recommends the integration of a limited set of synthesized nodes for the minority class to improve GNN performance in fraud detection. We summarize our contributions of this paper as follows:

- We propose to augment a few nodes in minority class that is better than balancing node class for graph neural networks.
- We propose a generative graph model for fraud detection that generates fraudulent nodes without external knowledge.
- We provide extensive experimental results to demonstrate the effectiveness of the proposed model.

II. Proposed Method

The proposed LAGA automatically generates fake fraudsters $\mathcal{V}^f$, their corresponding node features $\mathcal{X}^f$ and edge set $\mathcal{E}^f$, and then places them into the original graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X})$, making the original GNN model work better. The overall framework is shown in Fig. 1. It contains a generator (G) including feature generator (G_a) and edge generator (G_e), a discriminator (D_1) that tells if generated nodes are real or not, and another discriminator (D_2) that determines if nodes are benign or not. Two discriminators are the same GNN model for fraud detection (Here, we use BWGNN [23] as our discriminators). Next, we talk about how to generate feature $\mathbf{x}_f \in \mathcal{X}^f$ and edges $\mathcal{P}_f \subset \mathcal{E}^f$ for a generated node v_f.

A. Feature Generation

When generating node features, there are three options: (1) interpolating node features of fraudulent nodes, (2) interpolating node features between fraudsters and other nodes from all classes, and (3) using learnable variables. Previous works [19] point out that only fraudulent nodes are often limited, causing loss of generalizability while generating it from other classes may be extremely hard examples for discriminators. Therefore, we believe that the third option is the best choice. Formally, we randomly sample a noise vector $\mathbf{z}_f \sim p(\mathbf{z})$, and we generate a hidden embedding for a fake node by an Encoder composed of MLP layers.

$$\mathbf{h}_f = \text{Encoder}(\mathbf{z}_f). \tag{1}$$

We regard the embedding to be the potential graph embedding of the generated nodes in the potentially augmented graph. Later, we combine it with the node embeddings of real nodes to generate fake edges for generated nodes. To get the feature vector of a node $\mathbf{x}_f \in \mathbb{R}^d$ in the original feature space, we use a decoder composed by MLP, too.

$$\mathbf{x}_f = \text{Decoder}(\mathbf{h}_f). \tag{2}$$

B. Edge Generation

The fake fraudulent nodes should be in some 'communities' that can provide positive contributions to the second discriminator D_2. Therefore, by finding the 'communities' of fraudulent nodes, we can synthesize reliable fraudulent nodes. Moreover, the generated nodes need to match the distribution of the original graph. Inspired by the idea, we propose three steps to generate edges for the fake nodes including embedding real nodes, finding insertion nodes, and generating edges.

1) Embedding Real Nodes: To make sure the model can generate edges that mimic the real fraudulent nodes, the node embeddings for real nodes should contain feature-level and structure-level behavior information. Therefore, we propose feature-attentive graph convolutional networks (FA-GCN) to learn node embeddings, which contain a feature-attentive layer and graph convolutional layers. Fraudsters often camouflage themselves by having similar features to benign ones [3], thus, we believe that each feature should be learned separately to mitigate the feature camouflage issue. Thus, we propose a feature-attentive layer to vary the contribution of each feature for implying the true label. Formally, for the feature vector $\mathbf{x}_i$ of v_i, we embed the individual feature $\mathbf{x}_i(j) \in \mathbb{R}$ into a vector, where $1 \leq j \leq d$, and then add corresponding position vector p_j to get the individual embedding $\mathbf{h}_{ij} \in \mathbb{R}^h$ for j-th feature of i-th real node.

$$\mathbf{h}_{ij} = \text{FeatureEncoder}_j\left(\mathbf{x}_i(j)\right) + \mathbf{p}_j. \tag{3}$$

Note that $\mathbf{p}_j$ is a constant position vector on index j [24]. After that, we use an attention vector $\mathbf{a}_f \in \mathbf{R}^h$ to calculate the importance of each feature vector. Formally, the weight $\alpha_{ij} \in \mathbb{R}$ can be calculated as

$$\alpha_{ij} = \frac{\exp\left(\gamma\left(\mathbf{a}_f^\top \mathbf{h}_{ij}\right)\right)}{\sum_{j=1}^{d} \exp\left(\gamma\left(\mathbf{a}_f^\top \mathbf{h}_{ij}\right)\right)}, \tag{4}$$

$$\hat{\mathbf{h}}_i = \gamma\left(\sum_{j=1}^{d} \alpha_{ij} \mathbf{h}_{ij}\right), \tag{5}$$

where γ is LeakyReLU and $\mathbf{a}_f^\top$ means the transposition of the attention vector. To make sure the real node embedding contains structure information, we use graph convolutional layers as follows.

$$\mathbf{h}_i^{(l)} = \gamma\left(\sum_{v_k \in \mathcal{N}_i} \mathbf{h}_k^{(l-1)} \mathbf{W}^{(l)}\right) \tag{6}$$

where $\mathbf{W}^{(l)} \in \mathbb{R}^{h \times h}$ is a learnable weight at layer l, and $1 \leq l \leq L$. The input of the GCN layers $\mathbf{h}_i^{(0)}$ is the output of feature-attentive layer $\hat{\mathbf{h}}_i$. $\mathcal{N}_i$ is the neighboring node set for node v_i. We can stack L graph convolutional layers and take the average of outputs of all layers as the final embedding $\mathbf{h}_i$ of real node i.

$$\mathbf{h}_i = \frac{\sum_{l=1}^{L} \mathbf{h}_i^{(l)}}{L}. \tag{7}$$

2) Finding Insertion Nodes: To locate where the fake node should be inserted, we use a location network to find the nodes to be connected. Additionally, we want to improve the variety of 'communities' as well, so we want to sample a small of number of nodes from the real nodes $\mathcal{V}$. We use a locator to locate the sampled nodes, which is consisted of simple MLP layers. Assume we sample k nodes, then we have a vector

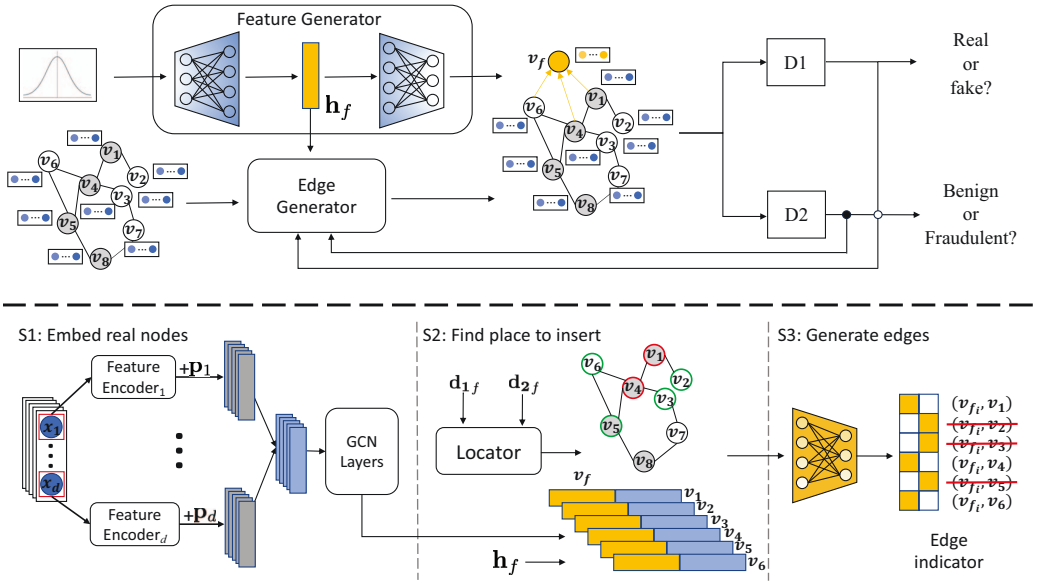


Fig. 1: **Model Architecture**. The grey nodes are fraudulent nodes and the white nodes are benign nodes. Note that the generated edges are single-directional so that the neighborhoods of the real nodes remain unchanged. The **upper** plot is the overall framework of LAGA. The generator G uses a feature generator and an edge generator to generate node features and edges for fake node v_f, respectively. Then, the discriminator D_1 determines if nodes are real or fake, and the discriminator D_2 determines if nodes are benign or fraudulent. The **lower** plot shows the details in the edge generator. Firstly, the edge generator embeds the real nodes via feature encoders and GCN layers. Then, it uses a locator to locate the potential neighborhood of the generated node with $\mathbf{d_1}_f$ of fake node v_f learned by D_1 and $\mathbf{d_2}_f$ of v_f learned by D_2. Finally, it concatenates the embedding of v_f and the embeddings of potential neighbors to generate edges.

$\mathbf{l}_e \in [0,1]^k$ to represent indices of sampled nodes for the fake node v_f at e-th epoch. The sampled nodes for next epoch is

$$\mathbf{l}_{(e+1)} = \text{Locator}(\mathbf{d_1}_f \sqcup \mathbf{d_2}_f \sqcup \mathbf{l}_e), \tag{8}$$

where $\mathbf{l}_{(e+1)} \in [0,1]^k$, and $\mathbf{d_1}_f$ and $\mathbf{d_2}_f$ are the embeddings of v_f from D_1 and D_2, respectively $\sqcup$ denotes concatenation operation. To map the location vector into real indices, we use the same methods stated in [17]. Then, we take the k sampled nodes and their ego networks as the candidate set $\mathcal{C}_f$ to be connected with the fake node.

3) Generating Edges: As we have the candidate set $\mathcal{C}_f$, we generate potential edges from it. Assume node v_m is in $\mathcal{C}_f$, an edge embedding $\hat{\mathbf{e}}_{(f,m)}$ for a potential edge between v_f and v_m can be obtained by concatenating the target generated node embedding and the candidate node embedding as

$$\hat{\mathbf{e}}_{(f,m)} = \text{EdgeGenerator}\left(\mathbf{h}_f \sqcup \mathbf{h}_m\right), \tag{9}$$

where $\hat{\mathbf{e}}_{(f,m)} \in \mathbb{R}^2$. To get the edge indicator, a Gumbel-softmax is applied as

$$\mathbf{e}_{(f,m)}(i) = \frac{\exp\left(\frac{\hat{\mathbf{e}}_{(f,m)}(i)}{\tau}\right)}{\sum_i \exp\left(\frac{\hat{\mathbf{e}}_{(f,m)}(i)}{\tau}\right)}, \tag{10}$$

where τ is the temperature and $\mathbf{e}_{(f,m)} \in [0,1]^2$. If $\mathbf{e}_{(f,m)}(0) > \mathbf{e}_{(f,m)}(1)$, then edge (v_f, v_m) is generated, otherwise, the edge is not generated. However, to facilitate the training process on edge generator, we treat $\mathbf{e}_{(f,m)}(0)$ as the edge weight and pass it to discriminators as the input later. The same process is applied on all candidate nodes in $\mathcal{C}_f$ to obtain $\mathcal{P}_f$. After finishing generating all nodes, we can have the augmented graph $\mathcal{G}' = (\mathcal{V} \cup \mathcal{V}^f, \mathcal{E} \cup \mathcal{E}^f, \mathcal{X} \cup \mathcal{X}^f)$. If dealing with multi-relational graphs, we just adopt the same process for each single-relational graph and combine them together as the final augmented graphs. It is worth to mention that the generated edges are directed edges only pointing to generated nodes so that the structures of the real nodes remain unchanged.

C. Framework Training

The adversial loss $\mathcal{L}_{adv}$ is adopted to generator and D_1, which is given by

$$\mathcal{L}_{adv} = -\frac{1}{|\mathcal{V}^f|}\sum_{m=1}^{|\mathcal{V}^f|}\left[\log D_1(v_m|\mathcal{G}) + \log\left(1 - D_1\left(v_{f_m}|\mathcal{G}'\right)\right)\right]. \tag{11}$$

where v_m can be any fraudsters in training data and $v_{f_m} \in \mathcal{V}^f$. While for training D_2 and G, we use the cross-entropy loss $\mathcal{L}_{ce}$, which is

$$\mathcal{L}_{ce} = -\sum_{v_i \in \mathcal{V}_{train} \cup \mathcal{V}^f}\sum_{c=1}^{2}\mathbf{y}_i(c)\log\hat{\mathbf{y}}_i(c), \tag{12}$$

where $\mathbf{y}_i$ are ground truth while $\hat{\mathbf{y}}_i$ are predicted label of D_2. However, with $\mathcal{L}_{adv}$ and $\mathcal{L}_{ce}$, locator is not trained so far.

TABLE I: Dataset Statistics

Dataset	#Nodes	#Edges			Fraud%
Yelp	45,954	R-U-R 49,315	R-T-R 573,616	R-S-R 3,402,743	14.5%
T-finance	39,357	21,222,543			4.58%

Therefore, we use policy gradient of reinforcement learning to update parameters in the generator and locator. In this view, the agent needs to learn a policy π that, at each epoch e, maps the previous ineraction with the environment $\mathbf{s}_e = \mathbf{l}_{e-1}, \mathbf{d_1}_f, \mathbf{d_2}_f$ to a distribution over locations. In our case, we regard the actions as the locations and rewards are determined by the predictions of two discriminators. That is to say, only the predictions of two discriminators are 1, the reward $R = 1$, otherwise, $R = -1$. The objective is to maximize the reward under the distribution $\mathcal{J}(\theta) = \mathbb{E}_{p(\mathbf{s}_e;\theta)}[R]$, where θ includes all parameters in G_a, G_e and the locator. Therefore, the reinforcement loss $\mathcal{L}_{rl}$ is given by

$$\mathcal{L}_{rl} = \frac{1}{|\mathcal{V}^f|}\sum_{m=1}^{|\mathcal{V}^f|}\nabla_\theta \log \pi(\mathrm{l}_{e+1}^m|\mathbf{s}_e^m;\theta)R^m. \tag{13}$$

D_1 is is firstly optimized to minimize the adversarial loss $\mathcal{L}_{adv}$ and G is subsequently trained to maximize it. Then, both G and D_2 are trained to minimize $\mathcal{L}_{ce}$. Lastly, G is trained to maximize $\mathcal{L}_{rl}$ with a hyperparameter λ to update G only.

III. Experiment

A. Experimental Setting

1) Experiment Setup and metrics: We conduct experiments on Yelp and T-finance, whose statistics are shown in Table I. We use 10% data of T-finance and 20% data of Yelp as the training set, respectively. 20% of the rest data is the validation set, and the remaining data is the testing set. All baselines share the same embedding dimension $h = 64$. For feature generation, Encoder contains 2 MLP layers while Decoder contains 1 MLP layer. The sigmoid function is the activation function in it. For edge generation, 2 GCN layers are applied for real node embeddings, and 2 MLP layers for generating edges. We also set $\lambda = 0.001$ for $\mathcal{L}_{rl}$. For methods only adding a few nodes, the number of generated nodes are respect to sample ratio α and the number of real fraudsters n in the training dataset, i.e., $\alpha * n$. Same number of augmentated nodes is applied to all imbalanced methods. We set learning rates to be 1×10^{-3}, 1×10^{-4}, 1×10^{-2} for feature and edge generators, D_1 and D_2, respectively. We choose Adam to be the optimizer. For all experiments, we set a fixed random seed.

Since the datasets are class imbalanced, we use AUC, F1 and average precision (AP) and accuracy (ACC) as our evaluation metrics.

2) Comparison Methods: We have ten baselines to compare with LAGA. MLP with node features and the original GNN model without augmentation as the first two baselines. Three traditional class imbalance techniques including under-sampling, over-sampling and re-weight. Three class balance methods based on GNNs, including PC-GNN [12], FRAUDRE [29], and DAGAD [11]. Moreover, we compare with GraphSMOTE [30] and GraphENS [19], which are two recent GNN-based methods that use node synthesis for class imbalance. All the baselines follow the settings in the corresponding papers.

B. Overall Performance

The overall performance is shown in Table II. To get this table, we set the number of sampled nodes $k = 3$ and temperature $\tau = 1 \times 10^{-3}$, sample ratio $\alpha = 0.2$ for Yelp, and set $k = 3$ and $\tau = 1 \times 10^{-4}$, sample ratio $\alpha = 1.0$ for T-finance. From this table, we see the performances of GNNs are generally better than MLP, which indicates graph structure provides useful information on classifying node labels. Obviously, on the Yelp dataset, the traditional class balance methods perform worse on AUC, ACC, and AP while better on F1 compared to the original, which verifies that GNNs easily overfit fraudulent nodes and underfit benign nodes with class balancing methods with traditional methods. For GNN-based class-balanced methods, like PC-GNN, FRAUDRE, and DAGAD on the Yelp dataset, even though their GNNs are not the same as the original, their performances are better than the base model (original) and traditional class-balanced methods, indicating the effectiveness of their models. However, the performance of LAGA is better than those GNN-based class-balanced methods, showing the proposed model enjoys better effectiveness. However, class-balanced methods can improve the performance but GNN-based class-balanced methods get similar or lower results compared to the original one on the T-finance dataset. Moreover, compared with GraphENS, GraphSMOTE is the better performer on both datasets, and both GraphENS and GraphSMOTE are both better than class-balanced methods which illustrates synthesizing nodes is better since they adaptively generate 'new' nodes that can avoid overfitting for the minority class. LAGA outperforms all baselines with only a few generated nodes, showing task-specific information is learned by LAGA, which proves the effectiveness of the proposed model.

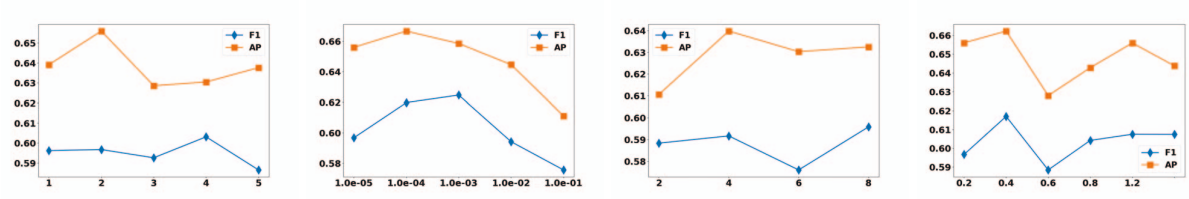
C. Parameter Analysis

We conduct experiments on Yelp to discuss how three important parameters affect the model, including the number of GCN layers in FA-GCN, τ in edge generation, sample size k in locator and the sample ratio α.

1) Number of GCN Layers: We show how the number of GCN layers affects the performance of the proposed model on Yelp with the number of GCN layers chosen from $\{1, 2, 3, 4, 5\}$. From Fig. 2a, we see that F1 is relatively steady with the first three layers and reaches a peak at the fourth layer. While AP gets the best score with two GCN layers. Therefore, adding more layers cannot improve the performance due to oversmoothing problem. Since the difference on F1 is not big, two GCN layers are the best option for LAGA.

TABLE II: Classification results on two real-world datasets. The best scores are bold.

Datasets	Metrics	MLP	Original	Balanced Methods						Imbalanced Methods		
				Under-sampling	Over-sampling	Re-weight	PC-GNN	FRAUD-DRE	DAGAD	Graph-SMOTE	Graph-ENS	LAGA
Yelp	AUC	82.70	81.70	80.22	81.32	81.68	85.39	88.55	89.20	89.43	88.94	**90.46**
	F1	42.26	42.46	45.14	47.13	47.88	54.76	59.39	59.60	57.87	58.82	**62.46**
	ACC	87.08	86.23	76.80	78.29	82.84	84.60	87.09	86.95	88.78	87.06	**88.93**
	AP	49.96	48.16	44.90	45.74	49.15	55.82	62.86	62.95	64.12	62.82	**65.85**
T-finance	AUC	92.72	96.20	96.14	95.53	96.14	90.40	95.19	96.18	96.34	96.17	**96.47**
	F1	70.37	83.50	80.44	81.16	80.23	68.06	83.62	83.43	83.92	82.69	**84.18**
	ACC	97.84	98.63	98.11	98.21	98.08	97.04	98.61	98.51	98.60	98.43	**98.66**
	AP	73.04	87.55	87.83	87.56	87.84	72.47	86.22	87.78	87.19	87.33	**88.35**



(a) Number of GCN Layers (b) Temperature τ (c) Sample Size k (d) Sample Ratio α

Fig. 2: Parameter Analysis.

2) Temperature τ in Edge Generator: τ in the Gumbel-softmax function is a key parameters in edge generation. Fig. 2b shows LAGA performance at different temperatures. We select $\tau \in \{1 \times 10^{-5}, 1 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2}, 1 \times 10^{-1}\}$, the performance increase from 1×10^{-5} to 1×10^{-3} and then decreases. F1 reaches peak when $\tau = 1 \times 10^{-4}$ while AP gets to the top when 1×10^{-3}, indicating small τ is better for the proposed model.

3) Sample Size k: We choose sample size k in locator from $\{2, 4, 6, 8\}$. The effect of k is shown in Fig. 2c. We observed that 4 is the best sample size among the four choices. It might be because the communities found by the locator include few fraudsters or the variety is relatively low when k is small (e.g., 2 in this case). While having many sample nodes (i.e., $k = 6 or 8 in this$) results in too complex communities which introduce noise instead.

4) Sample Ratio α: We choose sample ratio α from $\{0.2, 0.4, 0.6, 0.8, 1.0, 1.2\}$. The effect of α (i.e., the number of generated nodes) is shown in Fig. 2d. The performance goes to the top when only generating nodes that are 40% of the real fraudsters in the datasets. Then, the performance goes down at first and then goes up along with the sample ratio increases, and gradually becomes steady. Therefore, more nodes do not guarantee better performance. With only a few nodes, the performance can be improved by a large margin.

IV. Related Work

A. Graph Neural Networks for Fraud Detection

Graph Neural Networks (GNNs) are applied to fraud detection widely. Since fraudsters often camouflage themselves both features and relations, works like GraphConsis [14], CARE-GNN [3], DC-GNN [16], and DIGNN [10] consider dropping neighborhood when aggregating information. For example, CARE-GNN utilizes the BMBA module to control the neighborhood selection threshold. GraphConsis samples top-k neighbors by calculating the consistency score between node embeddings. H^2-FDetector [21] identifies homophilic and heterophilic connections and then propagates similar information on homophilic connections and different information on heterophilic connections. DAGNN [8] augments disparity between the target node and its heterogenous neighbors and similarity to homogenous neighbors. Other works like DCI [27] assume the decoupling representation learning and classification training is less biased by the hard instances, so they propose an SSL training scheme to learn node embeddings. Different from other GNNs in the spatial domain, BWGNN [23] has Beta Wavelet kernels to enable spectral and spatial localized band-pass filter. AO-GNN [6] decouples the AUC maximization process on GNNs to classifier parameter searching and edge pruning policy searching in order to maximize AUC. NGS [20] uses a differentiable neural architecture search to determine the optimal message-passing graph structure. Many works also explored specific methods for various tasks. GEM [15] adopts an attention module to discriminate the importance of node types in a heterogeneous graph to detect malicious accounts. FD-NAG [26] embeds non-target entities into edge features and pre-trains a GNN with the graph with contrastive learning to leverage unlabeled data to detect fraudsters in ride-sharing services.

B. Class Imbalance Learning for Node Classfication

Class imbalance learning for node classification is a crucial research area as imbalanced class distributions can significantly impact the performance of Graph Neural Networks

(GNNs). Recent studies in this field can be broadly categorized into data-level and algorithm-level approaches. At the data level, besides traditional methods including under-sampling and over-sampling, recent works generate nodes for the minority class. For example, GraphSMOTE [30] synthesizes nodes and edges for the minority class by interpolating between two labeled training instances from the same class. SORAG [5] synthesizes both labeled minority nodes and unlabeled nodes enriched by label correlations. To mitigate the "neighborhood memorization" issue, GraphENS [19] employs a gradient-based feature saliency method to mix features from a real minority node and a randomly selected target node. GraphMixup [28] generates features, edges and labels by mixing up their embeddings in the constructed semantic space. On the algorithm level, methods focus on redesigning the loss function to address class imbalance. In addition to the traditional re-weighting technique, FRAUDRE [29] proposes an imbalanced distribution-oriented loss that explicitly considers the imbalance ratio between classes. ReNode [2] tackles both quantity and topology imbalance by adaptively re-weighting the influence of labeled nodes based on their relative positions to class boundaries. DR-GCN [22] proposes two novel regularization terms to handle multi-class imbalance graphs. GCLE [4] transforms the original logical labels to numerical labels by utilizing local label correlations among the neighborhood and then uses the relative labeling importance of the numeric label to handle the class imbalance problem. DAGAD [11] uses an imbalance-tailored loss function for the imbalance problem. While these approaches have made significant contributions, the field of algorithm-level solutions for class imbalance in node classification is still relatively limited, calling for further research and exploration in the future.

V. Conclusion

In this paper, we address the class imbalance problem in GNN-based fraud detection. To mitigate this problem, we propose a novel approach called LAGA, which generates fake fraudulent nodes and their corresponding edges to augment the original graph. This augmentation enriches the information of the graph, leading to improved performance of GNN-based fraud detection models. LAGA model adopts the GAN framework, consisting of generators for node features and edges, as well as two discriminators. The discriminators distinguish between real and fake, as well as benign and fraudulent. The feature generator focuses on generating node features using learnable variants, while the edge generators generate edges within the neighborhood of fraudulent nodes. To learn node embeddings that capture both feature-level and structural information, we introduce FA-GCN incorporating a feature-attentive layer before the GCN layers and a locator to search nodes adaptively. Then, we utilize the Gumbel-softmax function to generate the edges. The results on two real-world fraud detection datasets demonstrate that our proposed approach outperforms existing methods and effectively tackles the class imbalance problem in GNN-based fraud detection.

VI. Acknowledgement

This work is also partially supported by NSF under grants IIS-1763365 and IIS-2106972 through Prof. Jiawei Zhang, grant CCF-2312617 through Prof. Xiaonan Zhang. This work is also supported in part by NSF under grant III-2106758 through Prof. Philip S. Yu.

References

[1] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29:626–688, 2015.

[2] Deli Chen, Yankai Lin, Guangxiang Zhao, Xuancheng Ren, Peng Li, Jie Zhou, and Xu Sun. Topology-imbalance learning for semi-supervised node classification. *Advances in Neural Information Processing Systems*, 34:29885–29897, 2021.

[3] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S Yu. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 315–324, 2020.

[4] Guodong Du, Jia Zhang, Min Jiang, Jinyi Long, Yaojin Lin, Shaozi Li, and Kay Chen Tan. Graph-based class-imbalance learning with label enhancement. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[5] Yijun Duan, Xin Liu, Adam Jatowt, Hai-tao Yu, Steven Lynden, Kyoung-Sook Kim, and Akiyoshi Matono. Sorag: Synthetic data over-sampling strategy on multi-label graphs. *Remote Sensing*, 14(18):4479, 2022.

[6] Mengda Huang, Yang Liu, Xiang Ao, Kuan Li, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. Auc-oriented graph neural network for fraud detection. In *Proceedings of the ACM Web Conference 2022*, pages 1311–1321, 2022.

[7] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2016.

[8] Qiutong Li, Yanshen He, Cong Xu, Feng Wu, Jianliang Gao, and Zhao Li. Dual-augment graph neural network for fraud detection. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 4188–4192, 2022.

[9] Xiangfeng Li, Shenghua Liu, Zifeng Li, Xiaotian Han, Chuan Shi, Bryan Hooi, He Huang, and Xueqi Cheng. Flowscope: Spotting money laundering based on graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 4731–4738, 2020.

[10] Zhixun Li, Dingshuo Chen, Qiang Liu, and Shu Wu. The devil is in the conflict: Disentangled information graph neural networks for fraud detection. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 1059–1064. IEEE, 2022.

[11] Fanzhen Liu, Xiaoxiao Ma, Jia Wu, Jian Yang, Shan Xue, Amin Beheshti, Chuan Zhou, Hao Peng, Quan Z. Sheng, and Charu C. Aggarwal. Dagad: Data augmentation for graph anomaly detection. In *ICDM*, pages 259–268, 2022.

[12] Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. Pick and choose: a gnn-based imbalanced learning approach for fraud detection. In *Proceedings of the Web Conference 2021*, pages 3168–3177, 2021.

[13] Yang Liu, Xiang Ao, Qiwei Zhong, Jinghua Feng, Jiayu Tang, and Qing He. Alike and unlike: Resolving class imbalance problem in financial credit risk assessment. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2125–2128, 2020.

[14] Zhiwei Liu, Yingtong Dou, Philip S Yu, Yutong Deng, and Hao Peng. Alleviating the inconsistency problem of applying graph neural network to fraud detection. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pages 1569–1572, 2020.

[15] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. Heterogeneous graph neural networks for malicious account detection. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 2077–2085, 2018.

[16] Lin Meng, Yuxiang Ren, and Jiawei Zhang. Decoupling graph neural network with contrastive learning for fraud detection. In *Database Systems for Advanced Applications: 28th International Conference, DASFAA 2023, Tianjin, China, April 17–20, 2023, Proceedings, Part IV*, pages 397–414. Springer, 2023.

[17] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. *Advances in neural information processing systems*, 27, 2014.

[18] Lina Ni, Jufeng Li, Huixin Xu, Xiangbo Wang, and Jinquan Zhang. Fraud feature boosting mechanism and spiral oversampling balancing technique for credit card fraud detection. *IEEE Transactions on Computational Social Systems*, 2023.

[19] Joonhyung Park, Jaeyun Song, and Eunho Yang. GraphENS: Neighbor-aware ego network synthesis for class-imbalanced node classification. In *International Conference on Learning Representations*, 2022.

[20] Zidi Qin, Yang Liu, Qing He, and Xiang Ao. Explainable graph-based fraud detection via neural meta-graph search. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 4414–4418, 2022.

[21] Fengzhao Shi, Yanan Cao, Yanmin Shang, Yuchen Zhou, Chuan Zhou, and Jia Wu. H2-fdetector: A gnn-based fraud detector with homophilic and heterophilic connections. In *Proceedings of the ACM Web Conference 2022*, pages 1486–1494, 2022.

[22] Min Shi, Yufei Tang, Xingquan Zhu, David Wilson, and Jianxun Liu. Multi-class imbalanced graph convolutional network learning. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 2879–2885. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track.

[23] Jianheng Tang, Jiajin Li, Ziqi Gao, and Jia Li. Rethinking graph neural networks for anomaly detection. In *International Conference on Machine Learning*, pages 21076–21089. PMLR, 2022.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2017.

[26] Chen Wang, Yingtong Dou, Min Chen, Jia Chen, Zhiwei Liu, and S Yu Philip. Deep fraud detection on non-attributed graph. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 5470–5473. IEEE, 2021.

[27] Yanling Wang, Jing Zhang, Shasha Guo, Hongzhi Yin, Cuiping Li, and Hong Chen. Decoupling representation learning and classification for gnn-based anomaly detection. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pages 1239–1248, 2021.

[28] Lirong Wu, Haitao Lin, Zhangyang Gao, Cheng Tan, Stan Li, et al. Graphmixup: Improving class-imbalanced node classification on graphs by self-supervised context prediction. *arXiv preprint arXiv:2106.11133*, 2021.

[29] Ge Zhang, Jia Wu, Jian Yang, Amin Beheshti, Shan Xue, Chuan Zhou, and Quan Z Sheng. Fraudre: Fraud detection dual-resistant to graph inconsistency and imbalance. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 867–876. IEEE, 2021.

[30] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. Graphsmote: Imbalanced node classification on graphs with graph neural networks. In *Proceedings of the 14th ACM international conference on web search and data mining*, pages 833–841, 2021.