Generative Graph Augmentation for Minority Class in Fraud Detection

Lin Meng
lm18h@fsu.edu
Department of Computer Science
Florida State University
Tallahassee, FL, USA

Hesham Mostafa
hesham.mostafa@intel.com
Intel Labs
San Diego, CA, USA

Marcel Nassar
marcel.nassar@intel.com
Intel Labs
San Diego, CA, USA

Xiaonan Zhang
xzhang@cs.fsu.edu
Department of Computer Science
Florida State University
Tallahassee, FL, USA

Jiawei Zhang
jiawei@ifmlab.org
Department of Computer Science
University of California, Davis
Davis, CA, USA

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 greatly distort the data distributions and eventually lead to the ineffective performance of GNNs. In this paper, we propose a novel approach named **G**raph **G**enerative Node **A**ugmentation (GGA), which improves GNN-based fraud detection models by augmenting synthetic nodes of the minority class. GGA utilizes the GAN framework to synthesize node features and related edges of fake fraudulent nodes. To introduce greater variety in the generated nodes, we employ an MLP for feature generation. We also introduce an attention module to encode feature-level information before graph convolutional layers for edge generation. Our empirical results on two real-world fraud datasets demonstrate that GGA 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.

CCS CONCEPTS

• **Information systems** → **Information retrieval**; **Information systems applications**.

KEYWORDS

Fraud Detection; Data Augmentation; Graph Representation Learning; Data Mining

ACM Reference Format:
Lin Meng, Hesham Mostafa, Marcel Nassar, Xiaonan Zhang, and Jiawei Zhang. 2023. Generative Graph Augmentation for Minority Class in Fraud Detection. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3583780.3615255

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3615255

1 INTRODUCTION

Fraud detection [4, 7] has gained significant importance in response to the escalating number of fraudsters exploiting victims for personal gain. These fraudulent activities encompass various forms, including the manipulation of individuals' shopping behaviors through the dissemination of unreliable reviews by opinion fraudsters [2, 17] and financial scams involving credit cards [8] or illegal money laundering [6]. Graphs have been a useful tool in fraud detection, as they capture essential information that aids in identifying fraudulent users and activities. Graphs typically comprise both benign and fraudulent individuals, along with the interactions between them. Accordingly, fraud detection is commonly framed as a node classification task [9, 13, 17], where the objective is to classify nodes (e.g., users) in a graph as either benign or fraudulent. Recent advancements in graph neural networks (GNNs) [5, 15] have opened up new avenues for analyzing and addressing this problem, leading to a multitude of studies applying GNNs to fraud detection [2, 10, 13, 16, 17].

However, real-world fraud detection datasets often exhibit a substantial class imbalance, with fraudsters constituting a minority portion of the overall population [1]. Consequently, GNN-based fraud detection models tend to predict the majority class, resulting in high accuracy but low recall. Traditional approaches for handling class imbalance of i.i.d. data, such as re-sampling [8, 11] and re-weighting methods [2, 13], cannot be directly applied to graph data due to the interconnected nature of nodes in graphs. Under-sampling reduces the training data for the majority class, while over-sampling replicates nodes and their related edges in the minority class [2], leading to overfitting due to the phenomenon "neighborhood memorization" [12]. Re-weighting techniques simply assign weights in the loss function, disregarding the edges in the graphs. This approach impacts the knowledge exchange process among neighboring nodes, potentially resulting in overfitting the minority class and underfitting the majority class [12]. To address

these issues, recent works [3, 12, 18] propose to augment minority class by node synthesis to balance class distribution in graph data.

Nonetheless, node class balancing with simple re-sampling or re-weighting may greatly distort the data distributions and lead to the ineffective performance of GNNs. Moreover, methods that synthesize nodes can improve performance, but they ignore task-specific information when generating nodes. Therefore, we propose a **G**raph **G**enerative Node **A**ugmentation (GGA) model that synthesizes nodes and their corresponding edges for minority class in order to improve the performance of GNN-based fraud detection models. Specifically, GGA is a generative adversarial network (GAN)-based graph augmentation model that has feature and edge generators and two discriminators. The model first generates the node features that do not interpolate node features of real nodes and then generates edges of fake nodes around the 'communities' that real fraudsters are in. Note that we introduce an additional discriminator to determine whether nodes are fraudulent or benign. This additional discriminator mitigates the generation of noise that could potentially compromise the quality of the generated graphs. Rather than focusing on balancing the class distribution by adding minority nodes, our approach suggests adding a small number of generated nodes belonging to the minority class to enhance the performance of GNNs for fraud detection.

2 PROPOSED METHOD

The proposed GGA 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 feature generator (G_a) and edges 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 [13] 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.

2.1 Feature Generation

Previous works [12] point out that combining features of minority nodes are often limited, causing loss of generalizability, and generating with other classes may be extremely hard examples for discriminators. In our paper, we simply adopt learnable variables to improve variety. Formally, we randomly sample a noise vector $\mathbf{z}_i \sim p(\mathbf{z})$, and we generate a hidden embedding for a fake node by an Encoder composed of MLP layers.

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

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.

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

2.2 Edge Generation

We want to imitate the relational behavior of real fraudsters. Therefore, by finding the 'communities' of fraudulent nodes, we synthesize reliable fraudulent nodes. Also, the generated nodes need to match the distribution of the original graph. Therefore, we propose three steps to generate edges for fake nodes including embedding real nodes, finding insertion nodes, and generating edges.

Embedding Real Nodes. To make sure the model 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 use a feature-attentive layer and graph convolutional layers. Since fraudsters often camouflage themselves by having similar features to benign ones [2], we believe each feature should be learned separately to mitigate the feature camouflage issue. Thus, we use a feature-attentive layer to vary the contribution of each feature for implying the true label. Formally, we embed the individual feature $\mathbf{x}_i(j) \in \mathbb{R}$ of v_i into a vector, where $1 \le j \le d$, and then add a corresponding position vector p_j to get the 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 [14]. 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

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

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.

$$\mathbf{h}_i^{(l)} = \gamma\left(\sum_{v_k \in \mathcal{N}_i} \mathbf{h}_k^{(l-1)}\mathbf{W}^{(l)}\right) \tag{5}$$

where $\mathbf{W}^{(l)} \in \mathbb{R}^{h\times h}$ is a learnable weight at layer l, and $1 \le l \le L$. 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.

$$\mathbf{h}_i = \frac{\sum_{l=1}^{L}\mathbf{h}_i^{(l)}}{L}. \tag{6}$$

Finding Insertion Nodes. To find where the fake node should be inserted, we use the most similar fraudulent nodes and their 'communities'. Here, we define the 1-hop neighborhood of real fraudsters as their 'communities'. Formally, we caculate the similarities between a fake fraudulent node v_f and all known fraudulent nodes from training set $\{v_m | y_m = 1 \cup v_m \in \mathcal{V}_{train}\}$. Then, we take the top-$k$ most similar nodes and their ego networks as the candidate set C_f to be connected with the fake node.

Generating Edges. As we have the candidate set C_f, we generate potential edges from it. Assume node v_m is in C_f, an edge embedding $\hat{\mathbf{e}}_{(f,m)}$ for a potential edge between v_f and v_m can be

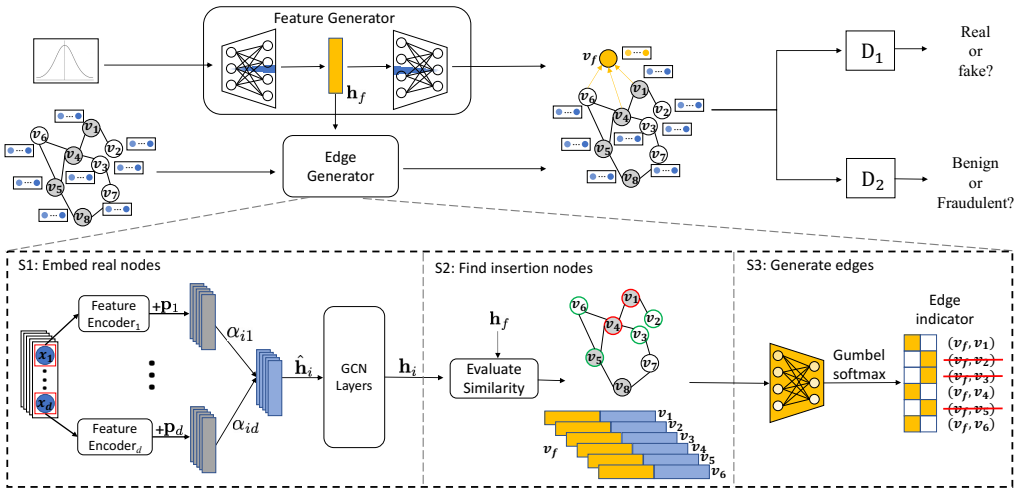


Figure 1: Model Architecture. The grey nodes are fraudulent nodes and the white nodes are benign nodes. Note that the generated edges are single-directional so that the neighborhoods of real nodes maintain unchanged.

obtained by concatenating the target generated node embedding and the candidate node embedding as

$$\hat{\mathbf{e}}_{(f,m)} = \text{EdgeGenerator}\left(\mathbf{h}_f \sqcup \mathbf{h}_m\right), \tag{7}$$

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

$$\mathbf{e}_{(f,m)}(i) = \frac{\exp\left(\frac{\hat{\mathbf{e}}_{(f,m)}(i)}{\tau}\right)}{\sum_i \exp\left(\frac{\hat{\mathbf{e}}_{(f,m)}(i)}{\tau}\right)}, \tag{8}$$

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. The same process is applied on all candidate nodes in C_f to obtain $\mathcal{P}_f$. After finishing generating all nodes, we can have the augmented graph $\mathcal{G}' = (\mathcal{V} \cup \mathcal{V}^f, \mathcal{E} \cup \mathcal{E}^f, \mathcal{X} \cup \mathcal{X}^f)$. Note that the generated edges are directed edges only pointing to generated nodes so that the structures of the real nodes remain unchanged. Moreover, in order to update the parameters in edge generation, we take the first element in edge indicators $\mathbf{e}_{(f,m)}(0)$ as the edge weight to be involved in the learning of discriminators, so that the parameters for edge generation can be updated.

2.3 Framework Training

We have two losses to train the model. For D_1, we adopt the adversarial loss $\mathcal{L}_{D_1}$ and cross-entropy loss $\mathcal{L}_{D_2}$ for D_2.

$$\mathcal{L}_{D_1} = -\sum_{x_i \in \mathcal{X}_{train}} \log\left(D_1\left(x_i\right)\right) - \sum_{z_i \in \mathcal{Z}} \left[1 - \log\left(D_1\left(G(z_i)\right)\right)\right]. \tag{9}$$

$$\mathcal{L}_{D_2} = -\sum_{v_i \in \mathcal{V}_{train} \cup \mathcal{V}^f} \sum_{k=1}^{2} \mathbf{y}_i(k) \log \hat{\mathbf{y}}_i(k), \tag{10}$$

where $\mathbf{y}_i$ is ground truth while $\hat{\mathbf{y}}_i$ is predicted label of D_2. We first train D_1, then we train feature generator G_a and edge generator G_e. Lastly, we train D_2 and two generators together. For a multi-relational graph, we apply the same process on each relation separately and then take the average node features of all relations as the final node feature.

Table 1: 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%

3 EXPERIMENT

3.1 Experimental Setting

3.1.1 Experiment Setup and metrics. We conduct experiments on Yelp and T-finance, whose statistics are shown in Table 1. We use 10% data as the training set and 20% of the rest data as the validation set, and the remaining data is the testing set. All baselines share the same embedding dimension $h = 64$. In G_a, Encoder contains 2 MLP layers while Decoder contains 1 MLP layer. The sigmoid function is the activation function in it. In G_e, 2 GCN layers are applied, $k = 2$ and $\tau = 1 \times 10^{-4}$. We set learning rates are 5×10^{-4}, 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 methods only adding a few nodes, we add 200 nodes for Yelp and 360 nodes for T-finance.

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

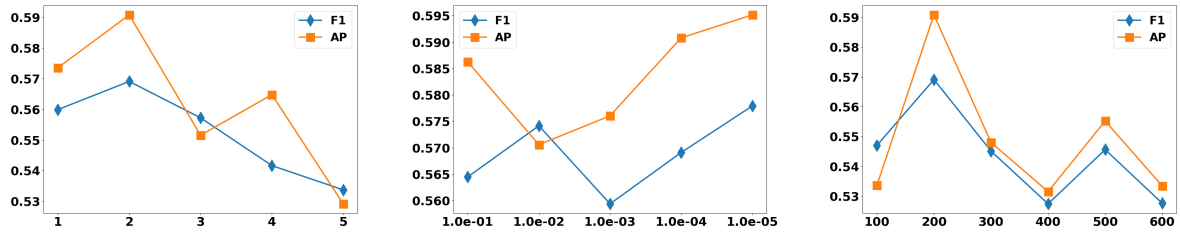
3.1.2 Comparison Methods. We have eight baselines. MLP only uses node features. Original GNN model without augmentation. Three traditional class imbalance techniques based on GNN, including under-sampling, over-sampling and re-weight. We also compare with over-sampling method which only adds a few nodes. Moreover, we compare with GraphSMOTE [18] and GraphENS [12], which are two recent GNN-based methods that use node synthesis for class imbalance.

3.2 Overall Performance

The overall performance is shown in Table 2. We see the performances of GNNs are generally better than MLP, which indicates graph structure provides good information on classifying node labels. Obviously, on the Yelp dataset, the traditional class balance

Table 2: Classification results on two real-world datasets. The best scores are bold and the second best scores are underlined.

Dataset	Metrics	MLP	Original	Class balanced methods			Adding few nodes			
				Under-sampling	Over-sampling	Re-weight	Over-sampling	GraphSMOTE	GraphENS	GGA
Yelp	AUC	80.72	<u>87.20</u>	86.61	85.24	86.30	86.35	84.75	86.12	**87.57**
	F1	32.11	<u>55.18</u>	54.20	54.31	53.49	53.15	51.35	54.04	**56.91**
	ACC	86.80	**86.67**	82.35	82.95	82.33	80.79	85.71	85.37	<u>86.66</u>
	AP	46.35	<u>56.87</u>	55.45	53.85	53.12	51.89	51.48	53.37	**59.08**
T-finance	AUC	92.72	93.79	96.14	95.53	96.14	<u>96.53</u>	96.34	96.17	**96.59**
	F1	70.37	72.85	80.44	81.16	80.23	79.47	<u>83.92</u>	82.69	**84.05**
	ACC	97.84	97.97	98.11	98.21	98.08	98.03	<u>98.60</u>	98.43	**98.67**
	AP	73.04	76.17	87.83	87.56	<u>87.84</u>	87.03	87.19	87.33	**88.64**



(a) Number of GCN Layers **(b) τ in Edge Generation** **(c) Number of Generated Nodes**

Figure 2: Parameter Analysis.

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. However, class-balanced methods can improve the performance on the T-finance dataset. Moreover, among methods only adding a few nodes, except GGA, GraphENS is the best performer on Yelp while GraphSMOTE is the best performer on T-finance, and they are both better than over-sampling a few nodes, which illustrates synthesizing nodes is better since they adaptively generate 'new' nodes that can avoid overfitting for the minority class. GGA outperforms all baselines by a large margin, which means GGA has the ability to generate 'unknown' useful node features and edges that not only improve GNN's performance but also avoid overfitting for minority class, indicating the effectiveness of GGA.

3.3 Parameter Analysis

We also conduct experiments on Yelp to discuss how three important parameters affect the model.

3.3.1 Number of GCN Layers. We show the results with different layers on Yelp. From Fig. 2a. We see two metrics increase until the number of layers is 2 and then decrease rapidly. It illustrates that 2-hop neighbors are positively related to the generation for GNN models, but higher-order neighborhood actually brings noise even negative information for two discriminators.

3.3.2 τ in Edge Generator. Fig. 2b shows GGA performance at different temperatures. We select $\tau \in \{1\times10^{-5}, 1\times10^{-4}, 1\times10^{-3}, 1\times10^{-2}, 1\times10^{-1}\}$, the performance decreases until 1×10^{-2} and then increase a little, and when $\tau = 1\times10^{-5}$, GGA gets the best score, which indicates the smaller τ can get better performance.

3.3.3 Number of Generated Nodes. The effect of the number of generated nodes is shown in Fig. 2c. The performance goes up within 200 nodes. Then the performance goes down along with the number of nodes increases, and the performance is similar to the scores by class-balanced methods. Therefore, GGA can get overfit when there are too many nodes injected even though the generated nodes may be new compared with the real ones.

4 CONCLUSION

In this paper, we address the class imbalance problem in GNN-based fraud detection. To mitigate this problem, we propose a novel framework called GGA, which generates 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. GGA model adopts a GAN framework to generate the graph node features and edges. The feature generator focuses on generating node features using learnable variants, while the edge generators generate edges within the neighborhood of fraudulent nodes where we incorporate a feature-attentive layer before the GCN layers and utilize the Gumbel-softmax to generate edges. The experimental results demonstrate that our proposed approach outperforms existing methods and effectively tackles the class imbalance problem in GNN-based fraud detection.

5 ACKNOWLEDGEMENT

This work is also partially supported by NSF under grants IIS-1763365 and IIS-2106972 through Prof. Jiawei Zhang, and grant CCF-2312617 through Prof. Xiaonan Zhang.

REFERENCES

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

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

[3] Yijun Duan, Xin Liu, Adam Jatowt, Hai-tao Yu, Steven Lynden, Kyoung-Sook Kim, and Akiyoshi Matono. 2022. SORAG: Synthetic Data Over-Sampling Strategy on Multi-Label Graphs. *Remote Sensing* 14, 18 (2022), 4479.

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

[5] Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

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

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

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

[9] Zhiwei Liu, Yingtong Dou, Philip S Yu, Yutong Deng, and Hao Peng. 2020. 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*. 1569–1572.

[10] Lin Meng, Yuxiang Ren, and Jiawei Zhang. 2023. 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*. Springer, 397–414.

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

[12] Joonhyung Park, Jaeyun Song, and Eunho Yang. 2022. GraphENS: Neighbor-Aware Ego Network Synthesis for Class-Imbalanced Node Classification. In *International Conference on Learning Representations*.

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

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

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

[16] Yanling Wang, Jing Zhang, Shasha Guo, Hongzhi Yin, Cuiping Li, and Hong Chen. 2021. 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*. 1239–1248.

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

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