Error-Bounded Graph Anomaly Loss for GNNs

Tong Zhao[1], Chuchen Deng[1], Kaifeng Yu[1], Tianwen Jiang[1,2], Daheng Wang[1], Meng Jiang[1]
[1]Department of Computer Science and Engineering, University of Notre Dame, USA
[2]Research Center for Social Computing and Information Retrieval, Harbin Institute of Technology, China
{tzhao2,cdeng,kyu2,dwang8,mjiang2}@nd.edu,twjiang@ir.hit.edu.cn

ABSTRACT

Graph neural networks (GNNs) have been widely used to learn node representations from graph data in an unsupervised way for downstream tasks. However, when applied to detect anomalies (e.g., outliers, unexpected density), they deliver unsatisfactory performance as existing loss functions fail. For example, any loss based on random walk (RW) algorithms would no longer work because the assumption that anomalous nodes were close with each other could not hold. Moreover, the nature of class imbalance in anomaly detection tasks brings great challenges to reduce the prediction error. In this work, we propose a novel loss function to train GNNs for anomaly-detectable node representations. It evaluates node similarity using global grouping patterns discovered from graph mining algorithms. It can automatically adjust margins for minority classes based on data distribution. Theoretically, we prove that the prediction error is bounded given the proposed loss function. We empirically investigate the GNN effectiveness of different loss variants based on different algorithms. Experiments on two real-world datasets show that they perform significantly better than RW-based loss for graph anomaly detection.

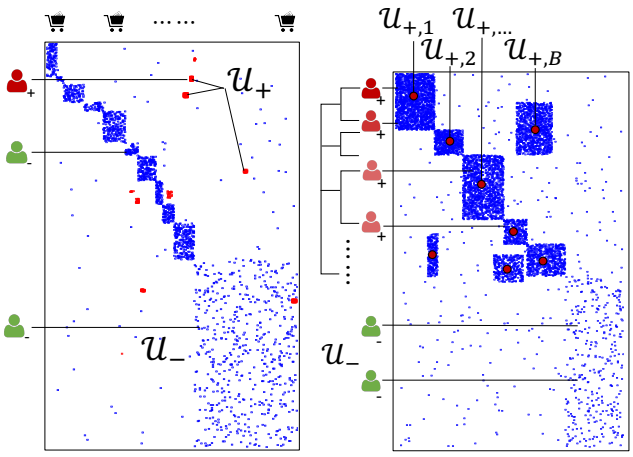
KEYWORDS

Graph neural network; Anomaly detection; Loss function

ACM Reference Format:
Tong Zhao[1], Chuchen Deng[1], Kaifeng Yu[1], Tianwen Jiang[1,2], Daheng Wang[1], Meng Jiang[1]. 2020. Error-Bounded Graph Anomaly Loss for GNNs. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3411979

1 INTRODUCTION

Detecting anomalies from large-scale graphs is an important task on many real-world applications. There are two main types of graph anomalies: *graph outliers* (e.g., fake reviewers) and *unexpected dense blocks* in graph's adjacency matrix (e.g., spammers, botnets). To learn node representations for such downstream tasks, Graph Neural Networks (GNNs) have been highly recognized for their abilities of aggregating attributed information from local neighborhood [29, 45, 48]. Usually, the models have two to four layers (i.e., #hops in the local neighborhood), sufficient for aggregation; to train the model parameters in an unsupervised manner (when labels are hardly available), random walk (RW) algorithms can discover more "global properties" (i.e., node pair-wise similarity in longer distance) that form *RW-based loss* on the last layer [17, 51].



(a) Anomalous node groups forming graph outliers **(b) Anomalous node groups forming dense blocks**

Figure 1: New global properties and new loss functions must be defined to train effective GNNs for graph anomaly detection. Nodes in different anomalous groups might not be reachable within random-walk distance.

However, we found that existing GNN models performed poorly on benchmarks in the task of graph anomaly detection. The reason is that graph anomalies do not have the aforementioned RW-based global properties. In other words, nodes of the same class might not be closer in the graph than those of different classes. For example, the graph outliers (e.g., fake reviewer group $\mathcal{U}_+$ in Figure 1a) do not have to be connected nor have common neighbors. They share global properties of being outliers (away from the majority) on the graph. Another example is that when a graph's adjacency matrix has multiple dense blocks (e.g., social botnet groups $\{\mathcal{U}_{+,i}\}_{i=1}^{B}$ in Figure 1b), the nodes in different blocks do not have to be connected while having similar global properties of creating unexpected density. How to effectively train GNNs for graph anomaly detection by capturing proper global properties is important and non-trivial.

Designing a proper loss is non-trivial because the GNN models, after sufficiently trained, are expected to accurately predict minority groups (e.g., outliers) from class-imbalanced data [7]. Compared with the node population of entire graph, graph outliers (e.g., fake reviewers) and/or nodes in dense blocks (e.g., botnet accounts) are the minority. Such severe imbalance is detrimental to model performance: when representations were not properly trained, the models

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3411979

would meet over-fitting on the minority classes and would perform poorly on test/unseen data [7]. Imbalanced machine learning has been studied from many perspectives [10, 20]; however, to the best of our knowledge, the research problem of reducing predictive error on imbalanced data for unsupervised graph representation learning is barely studied.

Present work. In this paper, we evaluate node pair-wise similarity using the "global properties" discovered by *graph mining algorithms*, and we present a novel *error-bounded* Graph Anomaly Loss (GAL) that is designed based on the similarities to learn effective node representations for the task of graph anomaly detection.

Here the graph mining algorithms refer to the algorithms that use heuristics (e.g., greedy search, spectral methods) to identify abnormal node groups from graph data. For example, Akoglu *et al.* [1] and Rayana *et al.* [39] proposed efficient algorithms to detect graph outliers by measuring the distance of their behavioral patterns from the pattern of the majority. Jiang *et al.* [25, 26] and Hooi *et al.* [22] identified botnets by measuring the unexpectedness of dense bipartite cores and greedily looking for them in large social networks. However, these algorithms ignore node individual identity and attributed information, assuming all the nodes in a group must have the same characteristics and the same label. Though the assumption is too strong to work on itself, these algorithms capture global structural information which can be used to train GNN parameters, specifically, to supervise the process of feature aggregation. The GNN models will take advantage of both the local and global contexts for effective graph anomaly detection.

Our proposed GAL can be generalized for both categories of graph mining algorithms, i.e., *graph outlier detection* [1, 32] and *dense block detection* [22, 25, 26] algorithms, to train GNNs for different purposes. It can be applied to all types of existing GNN algorithms such as GCN [29], GAT [45], and GraphSAGE [17]. Moreover, GAL automatically encourages larger margins for minority classes. It learns proper margins from the global imbalanced data distribution discovered by graph mining algorithms. So it creates better generalization on predicting minority classes. Theoretically, we obtain and prove the bound on prediction error.

Here we summarize the important features of our proposed GAL:

- **Generalizability:** GAL variants include loss functions for different tasks of graph anomaly detection such as outlier detection and unexpected density detection. It can be applied to train an arbitrary graph neural algorithm.
- **Effectiveness:** GAL captures global properties when random walk-based loss fails. It trains GNNs effectively for graph anomaly detection. Experiments on two real-world datasets, aiming at detecting two different kinds of anomalies, demonstrate that any GNN algorithm trained by the proposed GAL can perform significantly better than that trained by conventional loss.
- **Theoretical guaranteed performance:** GAL creates better generalization on patterns of minority groups. It maintains a bounded test prediction error on imbalanced data.

2 RELATED WORK

In this section, we survey research work of five related topics.

Imbalanced learning. Learning with imbalanced data has always been a challenging problem for machine learning. Most existing work focused on sampling and generating techniques. These algorithms either under-sample/over-sample the data objects [9, 34] or generate new data objects for the minority classes [10, 19]. [28, 31] proved that generalization error for both linear and non-linear models with hinge losses is bounded. Recently, Cao *et al.* [7] showed that the error bound could be found on imbalanced datasets.

Graph representation learning. The goal is to learn node representations in a low-dimensional space using random-walk paths or factorized features [15, 16, 37, 44]. These algorithms are transductive as they directly train node embeddings for individual nodes and require retraining or additional training to generate embeddings for new nodes. DeepWalk [37] and Node2Vec [16] learned node embeddings by performing word embedding models Word2Vec on "corpus" of nodes generated by random walk. BiNE [15] extended DeepWalk and optimized for bipartite graphs. DeepFD [47] learned the node embeddings with an auto-encoder that's designed for fraud detection. AdONE [6] was an unsupervised auto-encoder that learns outlier resistant embeddings.

Graph neural networks. GNNs are deep learning architectures for graph structured data. The core idea is to learn node representations through local neighborhoods. Kipf et al. [29, 30] proposed graph convolutional network (GCN) for semi-supervised graph representation learning. GCN is a transductive model that requires the calculation of whole graph Laplacian during training. Many inductive GNNs [12, 35, 45, 48, 49, 52] that follow a neighborhood aggregation scheme are proposed in recent years. In these models, the representation of a node is computed by recursively aggregating representations of its neighbors. More recently, GNN-based models were also proposed for tasks in varies fields of research such as natural language processing [50] and behavior modeling [46].

Graph outlier detection. The goal is to find outlier nodes in large graphs [2, 3]. Traditional density-based clustering methods [8, 18] regarded nodes in sparse regions as outliers. Similar approaches have been developed for bipartite graphs [43]. SpotLight [13] detected outliers in streaming graphs. BirdNest [21] was a Bayesian inference model for outlier detection on rating networks. Rev2 [32] was an iterative algorithm that calculated reviewer fairness scores. TwoFace [27] uncovered review manipulators users with structurally similar roles. Dominant [11] was an attributed graph auto-encoder that detects anomalous nodes. REPEN [36] learned representations for the distance-based outlier detection methods.

Unexpected density detection. The goal is to find suspicious nodes by locating dense blocks in the graph's adjacency matrix. SpokEn [38] found the "spokes" pattern on pairs of eigenvectors of graphs. LockInfer [26] identified pattern of communities based on singular vectors of graphs. fBox [41] located mini-scale attacks missed by spectral techniques. CatchSync [25] and CrossSpot [24] found the lockstep behaviors made by fraudulent users. Several methods [40, 42, 53] utilized dense subgraph detection algorithms to find the suspicious dense blocks. Hooi *et al.* [22] and Shin *et al.* [42] showed that the edge density-based suspiciousness of subgraph or subtensor can be maximized with approximation guarantee.

3 PROPOSED METHOD

3.1 Problem Definition

The goal of our approach is to learn low-dimensional representations of user nodes on a bipartite graph for detecting anomalous users. Suppose that $\mathcal{U}$ is the set of users, $\mathcal{V}$ is the set of items (e.g., products, hashtags), and $\mathcal{R} = \{r_{u,v} | u \in \mathcal{U}, v \in \mathcal{V}\}$, where $r_{u,v}$ denotes the weight of the edge between node u and node v. $r_{u,v} = 0$ indicates that u and v are not connected. If the graph is unweighted, $r_{u,v} = 1$ when the two nodes are connected. So, the problem is defined as follows:

Given a bipartite graph $G(\mathcal{U}, \mathcal{V}, \mathcal{R})$ and a set of user's node feature vectors $\{\mathbf{x}_u \in \mathbb{R}^{d_x}, \forall u \in \mathcal{U}\}$ (where d_x is the dimension of raw features), **find** a mapping function of the representations of user nodes $f : u \in \mathcal{U} \rightarrow \mathbf{z}_u \in \mathbb{R}^d$ where d is the number of latent dimensions in user embeddings. We expect the user representations are optimized for the task of anomaly detection by preserving both user's node attribute information and proper global properties.

Currently most GNNs are designed for homogeneous graphs by default. GNNs generate embeddings of a node by aggregating the embeddings from nodes in its local neighborhood. The assumption is that neighboring nodes have related information and/or similar characteristics. However, on a bipartite graph, this assumption does not hold as neighbors are different types of nodes. Hence in order to have GNNs better perform on bipartite graphs, rather than aggregating embeddings from immediate neighbors, we aggregate from 2-hop neighbors which are the same type as the target node.

3.2 Distribution-aware Anomaly Margin Loss

Traditionally, in order to train the network model in an unsupervised manner, RW-based loss functions are often applied to learn the output representations, $\mathbf{z}_u, \forall u \in \mathcal{U}$, and to tune the weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$ via stochastic gradient descent. The RW-based loss encourages nearby nodes to have similar representations as well as enforcing the representations of disparate nodes to be distinct, which can be formatted as [49]:

$$\mathcal{L}_{\text{RW}}(u) = \mathbb{E}_{u_+ \sim \mathcal{U}_{u+}, u_- \sim \mathcal{U}_{u-}} \max\{0, \mathbf{z}_u^T \mathbf{z}_{u_-} - \mathbf{z}_u^T \mathbf{z}_{u_+} + \Delta\}, \quad (1)$$

where Δ denotes a fixed margin hyper-parameter, $\mathcal{U}_{u+}$ denotes the set of user nodes that are reachable with a fixed length random-walk starting from u, and $\mathcal{U}_{u-}$ denotes $\mathcal{U} \setminus \mathcal{U}_+$. However, as we mentioned before, it is neither proper nor effective when the task is to detect anomalies on graphs. Task-specific loss functions are desired. Fortunately, researchers have proposed algorithms about learning with imbalanced data [7, 28, 31] and mining anomalies from graph data [3, 22, 25, 32]. Hence we propose a task-specific class-distribution-aware graph anomaly loss that is able to utilize the results of these unsupervised algorithms. Here we first introduce the class-distribution-aware margin loss function.

Let y_u denote the the label of user node u (note that user nodes in different dense blocks should have different labels). We assume that the class-conditional distribution $\mathcal{P}(u|y_u)$ is the same at training and testing. Then let $\mathcal{P}_j$ denote the class-conditional distribution, that is, $\mathcal{P}_j = (u|y_u = j)$, For our graph neural network model $f : \mathcal{U} \rightarrow \mathbb{R}^d$, we use function $g : \mathcal{U} \times \mathcal{U} \rightarrow \mathbb{R}$ to denote the similarity of the representations of any two user nodes u and u':

$$g(u, u') = f(u)^T \cdot f(u'), \quad (2)$$

and $L_{bal}[g]$ to denote the standard 0-1 test error on the balanced data distribution:

$$L_{bal}[g] = \Pr_{(u,j)\sim\mathcal{P}_{bal}}[\min_{y_{u_+}=j} g(u, u_+) < \max_{y_{u_-}\neq j} g(u, u_-)]. \quad (3)$$

The error L_j for class j is then defined similarly as:

$$L_j[g] = \Pr_{u\sim\mathcal{P}_j}[\min_{y_{u_+}=j} g(u, u_+) < \max_{y_{u_-}\neq j} g(u, u_-)]. \quad (4)$$

Let n_j be the number of user nodes in class j and $S_j = u : y_u = j$ denote the set of user nodes with label j. Define the training margin for class j as:

$$\gamma_j = \min_{u\in S_j} \left(\min_{y_{u_+}=j} g(u, u_+) - \max_{y_{u_-}\neq j} g(u, u_-) \right), \quad (5)$$

where $\gamma_{min} = \min\{\gamma_1, \ldots, \gamma_j\}$ is the widely used training margin in previous studies [31]. Then we let $L_{\gamma,j}$ denote the margin loss for class j when training:

$$L_{\gamma,j}[g] = \Pr_{u\sim\mathcal{P}_j}[\min_{y_{u_+}=j} g(u, u_+) < \max_{y_{u_-}\neq j} g(u, v') + \gamma], \quad (6)$$

and let $\hat{L}_{\gamma,j}$ denote its empirical variant. For a hypothesis class $\mathcal{G}$, we use $\hat{\mathfrak{R}}(\mathcal{G})$ to denote the empirical Rademacher complexity of margin for class j:

$$\hat{\mathfrak{R}}_j(\mathcal{G}) = \frac{1}{n_j} \mathbb{E}_\sigma \left[\sup_{g\in\mathcal{G}} \sum_{u\in S_j} \sigma_u [\min_{y_{u_+}=j} g(u, u_+) - \max_{y_{u_-}\neq j} g(u, u_-)] \right], \quad (7)$$

where σ is a vector of i.i.d. uniform $\{-1, +1\}$ bits. Here we consider the bound below for balanced test distribution by considering the margin of each class, which allows us to design distribution-aware margin loss function that is suitable for the imbalanced data.

Theorem 3.1. *[7] With probability $1-\delta$ over the randomness of the training data, for all choices of class-dependent margins $\gamma_1, \gamma_2, \ldots, \gamma_k > 0$, all hypotheses $g \in \mathcal{G}$ will have balanced-class generalization bounded by:*

$$L_{bal}[g] \leq \frac{1}{k} \left(\sum_{j=1}^{k} \hat{L}_{\gamma_j,j}[g] + \frac{4}{\gamma_j} \hat{\mathfrak{R}}_j(\mathcal{G}) + \varepsilon_j(\gamma_j) \right), \quad (8)$$

where $\varepsilon_j(\gamma) \triangleq \sqrt{\frac{\log\log_2(\frac{2\max_{u,v\in\mathcal{U},g\in\mathcal{G}}|g(u,v)|}{\gamma}) + \log\frac{2e}{\delta}}{n_j}}$ is typically a low-order term in n_j. Concretely, the Rademacher complexity $\hat{\mathfrak{R}}_j(\mathcal{G})$ will typically scale as $\sqrt{\frac{C(\mathcal{G})}{n_j}}$ for some complexity measure $C(\mathcal{G})$, in which case:

$$L_{bal}[g] \leq \frac{1}{k} \left(\sum_{j=1}^{k} \hat{L}_{\gamma_j,j}[g] + \frac{4}{\gamma_j} \sqrt{\frac{C(\mathcal{G})}{n_j}} + \varepsilon_j(\gamma_j) \right). \quad (9)$$

Note that although the losses and empirical Rademacher complexity of margins are defined different from those in Theorem 2 in [7], the above inequality still holds. For the coherence of reading, the proof of Theorem 3.1 is provided in the next subsection.

The balanced generalization error bound (Eq. 9) suggests that in order to improve the generalization of minority classes, we should

enforce larger margins for them. However, manually assigning larger margins for minority classes may lead to sub-optimal margin for the frequent class and hence hurt the model's performance. Thus here we take the binary classification problem as an example of showing how to obtain the optimal trade-off.

When $k = 2$, we aim to optimize the balanced generalization error bound in Eq. 9, which can be simplified to (after removing constant factors, common factor $C(\mathcal{G})$ and low order term $\varepsilon_j(\gamma_j)$) [7]

$$\frac{1}{\gamma_1\sqrt{n_1}} + \frac{1}{\gamma_2\sqrt{n_2}}. \tag{10}$$

Although the it is hard to get the optimal margins with the above equation as they are complicate functions of the parameters in $g(\cdot)$, we can figure out the relative scales between the two margins. Suppose we have $\gamma_1^*, \gamma_2^* > 0$ that minimize the equation above, we observe that any $\gamma_1' = \gamma_1^* - \delta$ and $\gamma_2' = \gamma_2^* + \delta$ (where $-2\gamma_2^* < \delta < \gamma_1^*$) can be realized by the same parameters with a shifted bias term. Therefore, for γ_1^*, γ_2^* to be optimal, the following inequality must be satisfied [7],

$$\frac{1}{\gamma_1^*\sqrt{n_1}} + \frac{1}{\gamma_2^*\sqrt{n_2}} \le \frac{1}{(\gamma_1^* - \delta)\sqrt{n_1}} + \frac{1}{(\gamma_2^* + \delta)\sqrt{n_2}}, \tag{11}$$

which implies that

$$\gamma_1^* \propto n_1^{-1/4}, \text{ and } \gamma_2^* \propto n_2^{-1/4}. \tag{12}$$

Given the trade-off above, we can define our margin loss as

$$\begin{aligned} \mathcal{L}(u) &= \max\{0, \max_{y_{v'} \neq y_u} g(u, v') - \min_{y_v = y_u} g(u, v) + \Delta_{y_u}\}, \\ &\text{where } \Delta_{y_u} = \frac{C}{n_{y_u}^{1/4}}. \end{aligned} \tag{13}$$

Here C is a constant hyper-parameter. When applying the above loss function on real-world graphs, it is impossible to enumerate all node pairs when calculating the minimum and maximum distances. Hence we use positive and negative sampling to approximate the distances. That is, we propose the following margin loss function in our GAL,

$$\begin{aligned} \mathcal{L}(u) =& \mathbb{E}_{u_+ \sim \mathcal{U}_{u+}, u_- \sim \mathcal{U}_{u-}} \max\{0, g(u, u_-) - g(u, u_+) + \Delta_{y_u}\}, \\ &\text{where } \Delta_{y_u} = \frac{C}{n_{y_u}^{1/4}}. \end{aligned} \tag{14}$$

Here $\mathcal{U}_{u+}$ denotes the set of user nodes that has the same label as u, $\mathcal{U}_{u-}$ denotes $\mathcal{U} \setminus \mathcal{U}_{u+}$, and $n_{y_u} = |\mathcal{U}_{u+}|$.

3.3 Proofs

In order to prove Theorem 3.1, we first need to get familiar with the following Theorem for standard margin-based generalization bounded (Theorem 5 in [28]) in our notation.

Theorem 3.2. *[28, 31] Consider and arbitrary function class $\mathcal{G}$ such that $\forall g \in \mathcal{G}$ we have $\sup_{x \in \mathcal{X}} \le C$. Then, with probability at least $1 - \delta$ over the sample, for all margins $\gamma > 0$ and all $g \in \mathcal{G}$ we have,*

$$L[g] \le K_\gamma[g] + \frac{4}{\gamma_j}\mathfrak{R}_n(\mathcal{G}) + \sqrt{\frac{\log\log_2 \frac{4C}{\gamma}}{n}} + \sqrt{\frac{\log\frac{1}{\delta}}{2n}}, \tag{15}$$

where $K_\gamma[f]$ is the fraction of the data that have γ-margin mistakes.

Proof. [28] Let $l_\gamma(t)$ be defined as

$$l_\gamma(t) = \begin{cases} 1 & t \le 0, \\ 1 - \frac{t}{\gamma} & 0 < t < \gamma, \\ 0 & t \ge \gamma. \end{cases} \tag{16}$$

For $j = 0, 1, \ldots$, set $\gamma_j = C/2^j$ and $\delta_j = \delta/(i+1)^2$. Applying Theorem 3 in [28] to the loss function l_{γ_j}, which has Lipschitz constant $1/\gamma_j$, we get the following inequality with probability at least $1 - \delta_j$ for all $g \in \mathcal{G}$,

$$L_{\gamma_j}[g] \le \hat{L}_{\gamma_j}[g] + \frac{2}{\gamma_j}\mathfrak{R}_n(\mathcal{G}) + \sqrt{\frac{\log\frac{1}{\delta}}{2n}}. \tag{17}$$

Note that for any $\gamma > 0$ and any $g \in \mathcal{G}$, $L[g] \le J_\gamma[j]$ and $\hat{L}_\gamma[f] \le K_\gamma[f]$. Hence with probability at least $1 - \delta_i$, for all $g \in \mathcal{G}$, we have

$$L[g] \le K_{\gamma_j}[g] + \frac{2}{\gamma_j}\mathfrak{R}_n(\mathcal{G}) + \sqrt{\frac{\log\frac{1}{\delta}}{2n}}. \tag{18}$$

Taking union bound over all j, we get that with probability at least $1 - \pi^2\delta/6 \ge 1 - 2\delta$, for all j and all $g \in \mathcal{G}$, we still have

$$L[g] \le K_{\gamma_j}[g] + \frac{2}{\gamma_j}\mathfrak{R}_n(\mathcal{G}) + \sqrt{\frac{\log\frac{1}{\delta}}{2n}}. \tag{19}$$

Note that since $|g(x)| \le C$, $\hat{L}_C[g] = 1$ and so that bound claimed in the Theorem holds trivially for $\gamma \ge C$. For $\gamma < C$ find the j such that $\gamma_j \le \gamma < \gamma_{j-1}$. Note that this means $j \le \log_2(C/\gamma) + 1$. Since $K_{\gamma_j}[g] \le K_\gamma[g]$, $1/\gamma_j \le 2/\gamma$ and $\log(1/\delta_j) \le \log(1/\delta) + 2\log\log_2(4C/\gamma)$, we have

$$L[g] \le K_\gamma[g] + \frac{4}{\gamma_j}\mathfrak{R}_n(\mathcal{G}) + \sqrt{\frac{2\log\log_2\frac{4C}{\gamma} + \log(1/\delta)}{2n}}. \tag{20}$$

□

Now we lead to the proof of Theorem 3.1:

Proof. We first prove the generalization seperately for each class j. Let $L_j[g]$ denote the 0-1 test error of classifier g on examples drawn from $\mathcal{P}_j$. Since all examples of class j is a set of n_j i.i.d. draws from the conditional distribution $\mathcal{P}_j$, we can apply the standard margin-based generalization bound (Theorem 3.2 [28, 31]) and obtain the following bound with probability $1 - \delta/c$, for all $\gamma_j > 0$ and $g \in \mathcal{G}$,

$$\begin{aligned} L_j[g] \le& \hat{L}_{\gamma_j, j}[g] + \frac{4}{\gamma_k}\hat{\mathfrak{R}}_j(\mathcal{G}) \\ &+ \sqrt{\frac{\log\log_2(\frac{2\max_{u,v\in\mathcal{U},g\in\mathcal{G}}|g(u,v)|}{\gamma})}{n_j}} + \sqrt{\frac{\log\frac{2c}{\delta}}{n_j}}. \end{aligned} \tag{21}$$

Since $L_{bal} = \frac{1}{k}\sum_{j=1}^k L_j$, we can union the above bound over all classes and average Eq. 21 to get the generalized bound. □

3.4 Estimating the Parameters of GAL

Recall that our task is unsupervised. When applying the distribution-aware anomaly margin loss (Eq. 14), we are not aware of the label for each user node $u \in \mathcal{U}$ during the training process. Therefore, we utilize the results of existing unsupervised graph-based *outlier detection* [3, 32] and *dense blocks detection* [22, 25] algorithms to estimate the user node sets $\mathcal{U}_{u+}$ and $\mathcal{U}_{u-}$ for each user node u. We call GAL utilizing the results of these algorithms respectively as GAL with *graph outlier loss* and GAL with *dense block loss*.

3.4.1 Graph outlier detection algorithms.
Given a graph, they assign binary labels to nodes in an unsupervised way. We use $\mathcal{U}_o$ and $\mathcal{U}_n$ to denote the set of outlier nodes and the set of normal nodes, respectively. Here are the three categories of the algorithms:

• *Feature-based graph outlier detection.* These methods define the outlierness score of node u based on a pair of its particular features a_u and b_u [2, 25]:

$$outlierness(u) = |b_u - \hat{b}_u| \text{ or } \frac{\max(b_u, \hat{b}_u)}{\min(b_u, \hat{b}_u)} \cdot \log(|b_u - \hat{b}_u| + 1), \quad (22)$$

where $\hat{b}_u$ is the predicted feature value based on the observed a_u. Intuitively, the measure is the "distance to fitting line (a, b)." Akoglu *et al.* [2] adopted four basic features such as number of neighbors, number of edges, total weight, and principal eigenvalue of the weighted adjacency matrix. Power laws were observed between the features with a large population of nodes (i.e., $b_u \propto a_u{}^{\gamma}$, γ is a constant). Big distance to the power-law fitting line indicates the role of graph outlier. Jiang *et al.* [25] proposed two high-order features: one is called synchronicity which describes how similar a node's neighbors are with each other in the space of basic features (e.g., degree, PageRank); the other is called normality that describes how similar the neighbor nodes are with every node in the space. They found the synchronicity had a parabolic lower limit of the normality (i.e., $sync_u \propto \alpha \cdot norm_u{}^2 + \beta$, α and β are constants) and designed an outlierness scoring function to catch the suspicious nodes. Big synchronicity and small normality indicate suspiciousness.

• *Structure-based graph outlier detection.* These methods define the outlierness score using the graph structure. They assume that the majority of users have low outlierness score. Then users with high outlierness scores can be reported as outliers [32]:

$$outlierness(u) = 1 - \frac{\sum_{v \in \mathcal{N}_{\mathcal{V}}(u)} \hat{R}(u,v) + \alpha_1 \cdot \mu_f + \alpha_2 \cdot \prod_U(u)}{|\mathcal{N}_{\mathcal{V}}(u)| + \alpha_1 + \alpha_2}, \quad (23)$$

where $\hat{R}(u, v)$ is the normality score of the rating from u to item v, μ_f is the prior belief of u's normality score given by BirdNest [21]. $\prod_U(u)$ is u's behavior normality score. α_1 and α_2 are constants.

• *Model-based graph outlier detection.* The idea behind these methods is that the majority of the graph, or say, the structural dependency can be learned by a specific graph model (e.g., compression model, generative model) and the outliers deviate significantly from the model. Chakrabarti [8] used a compression scheme based on the Minimum Description Length (MDL) philosophy. Therefore, the removal of outliers led to the maximum reduction in compression cost. Gao *et al.* [14] used the hidden Markov random fields (HMRF) model to characterize normal communities and assumed that the outliers follow a uniform distribution.

GAL with graph outlier loss. The graph outlier loss function encourages the pairs of positive nodes (outliers) to have similar representations and encourages the pairs of negative nodes (i.e., normal nodes) to have similar representations. It enforces that the representations of pairs of positive and negative nodes are highly distinct. Therefore, when sampling for each user node $u \in \mathcal{U}$ in the loss function (Eq 14), we let

$$\mathcal{U}_{u+} = \begin{cases} \mathcal{U}_n & \text{, if } u \in \mathcal{U}_n \\ \mathcal{U}_o & \text{, if } u \in \mathcal{U}_o \end{cases}, \mathcal{U}_{u-} = \begin{cases} \mathcal{U}_o & \text{, if } u \in \mathcal{U}_n \\ \mathcal{U}_n & \text{, if } u \in \mathcal{U}_o \end{cases}. \quad (24)$$

3.4.2 Density-based graph anomaly detection algorithms.
These algorithms defined a measurement on how suspicious a subgraph is with respect to the size and high density in a large graph, then employed an efficient algorithm scheme (e.g., greedy search) to detect the subgraphs of high suspiciousness. Suppose the detection algorithm finds B dense subgraphs (i.e., "blocks") $\{\mathcal{B}_i = (\mathcal{U}_{b,i} \subseteq \mathcal{U}, \mathcal{V}_i \subseteq \mathcal{V})\}_{i=1}^{B}$, where $\mathcal{U}_{b,i}$ and $\mathcal{V}_i$ denote the set of user nodes and item nodes in block $\mathcal{B}_i$, respectively. $\mathcal{U}_n = \mathcal{U} \setminus \cup_{i=1}^{B} \mathcal{U}_{b,i}$ denotes the set of normal nodes that have never participated in dense blocks. For block $\mathcal{B}_i$, we denote the size by $n_i = |\mathcal{U}_{b,i}|$ and $m_i = |\mathcal{V}_i|$; we denote the number of ratings in the block by $c_i = |\{r_{u,v} > 0 | u \in \mathcal{U}_{b,i}, v \in \mathcal{V}_i\}|$. The suspiciousness score is defined in different ways in different approaches:

- *Average Degree (AD)* [4, 5]: $susp^{AD}(\mathcal{B}_i) = \frac{c_i}{n_i}$;
- *Singular Value (SV)* [38, 41]:$susp^{SV}(\mathcal{B}_i) = \frac{c_i}{\sqrt{n_i \cdot m_i}}$;
- *Kullback–Leibler divergence of Density (KL)* [24]:

$$susp^{KL}(\mathcal{B}_i) = n_i \cdot m_i \cdot D_{KL}(\rho_i \parallel p); \quad (25)$$

where $\rho_i = \frac{c_i}{n_i \cdot m_i}$ is the block density, $p = \frac{|\{r_{u,v} > 0 | u \in \mathcal{U}, v \in \mathcal{V}\}|}{|\mathcal{U}| \cdot |\mathcal{V}|}$ is the data density, and $D_{KL}(\rho_i \parallel p) = p - \rho + \rho \log \frac{\rho}{p}$ is the KL divergence between ρ_i and p.

GAL with dense block loss. Suspicious user nodes may perform in multiple blocks. So the graph density loss should be designed based on the following assumptions: (i) user node pairs in the same dense block have similar representations; (ii) pairs of normal nodes have similar representations; (iii) for each suspicious user u, the representations of normal nodes and the nodes in blocks that do not include u are dissimilar with u's representation.

Therefore, considering the possibility of one user occurring in multiple dense blocks, when sampling for each user node $u \in \mathcal{U}$ in the loss function (Eq 14), we have

$$\mathcal{U}_{u+} = \begin{cases} \mathcal{U}_n & \text{, if } u \in \mathcal{U}_n \\ \bigcup_{i=1, u \in \mathcal{U}_{b,i}}^{B} \mathcal{U}_{b,i} & \text{, if } u \notin \mathcal{U}_n \end{cases}, \quad (26)$$

$$\mathcal{U}_{u-} = \begin{cases} \bigcup_{i=1}^{B} \mathcal{U}_{b,i} & \text{, if } u \in \mathcal{U}_n \\ \mathcal{U} \setminus \bigcup_{i=1, u \in \mathcal{U}_{b,i}}^{B} \mathcal{U}_{b,i} & \text{, if } u \notin \mathcal{U}_n \end{cases}. \quad (27)$$

4 EXPERIMENTS

In this section, we evaluate the proposed GAL for suspicious behavior detection on two real-world datasets. We first introduce experimental settings, then present the results on the two datasets,

following with ablative study and sensitivity analysis of GAL. Our code package and datasets can be found on GitHub[1].

4.1 Experimental Settings

4.1.1 Datasets.

Bitcoin-Alpha is a trust network of Bitcoin trading users on the Alpha platform [33], where each edge indicates a rating from one user to another with a rating score. The network has 3,275 user nodes and 3,742 items nodes. Kumar et al. [32] labeled 214 users: 83 and 131 are labeled as suspicious users and benign users, respectively. The raw feature vector is concatenated by three parts: (i) a one-hot vector indicating the degree of the user node; (ii) the summation of one-hot vectors that each represents a rating score given by this user; (iii) the summation of one-hot vectors where each hot represents a time interval between two consecutive ratings.

Tencent-Weibo is a user-posts-hashtag graph from a Twitter-like platform. It has 8,405 users and 61,964 hashtags [23]. The weight of user-hashtag edge is the number of the particular hashtag the user posted. Temporal information was used to label the users. The algorithm in [23] assumed that posting two messages within a specific number of seconds such as 10, 15, 30, 45, and 60 is a suspicious event. If a user made at least 5 suspicious events, he/she is labeled as a suspicious user; if made no suspicious event, he/she is a benign user. So we have 868 suspicious users and 7,537 benign users. Since the ground truth was generated using time information, we do not use timestamps to create raw user features. Therefore, the raw feature vector has two parts: (i) For each user, we first sum up one-hot vectors. Each hot represents the location where a micro-blog post was made. Then we reduce #dimensions of the location features to 100 using SVD. (ii) For each user, we reduce #dimensions of bag-of-words features to 300.

4.1.2 Baseline methods and GAL variants.

Unsupervised dense block detection methods.

• Fraudar [22]: it catches suspicious dense subgraphs with theoretical bounded densities for camouflage;
• CatchSync [25]: it captures the synchronized behavior patterns in rating networks and social networks;
• LockInfer [26]: it uses singular vectors of adjacency matrix to find anomalous groups of users in spectral subspaces.

Unsupervised graph outlier detection methods.

• Fraudar_r: it is the reverse of Fraudar. We use Fraudar to detect first several dense blocks until the remaining density is very low and report the remaining users as outliers;
• LockInfer_r: it is the reverse of LockInfer. We use LockInfer to detect all user groups in spectral subspace and report users that are not contained in any group as outliers;
• FraudEagle [1]: it uses a belief propagation-based algorithm to give a fraud score to each user. Outlying users who behaves more different than the majority are given higher fraud scores;
• BirdNest [21]: it ranks users by a Bayesian model with the users' rating timestamps and rating score distribution;
• REV2 [32]: it uses an iterative algorithm to find unfair users whose behaviors can be considered outlier compared with the majority.

Unsupervised graph embedding methods.

• Node2Vec [16]: it uses biased random walks to capture the homophily and local structure information of the network. Hyperparameters $p, q \in \{0.25, 0.5, 1, 2, 4\}$ control the search bias;
• LINE [44]: it uses 1st order and 2nd order proximity to capture the local and 2-hop structure of the network via edge sampling;
• BiNE [15]: it learns the representations of vertices in a bipartite network. Biased random walks are conducted to preserve the long-tail distribution of vertices.

Unsupervised GNN methods.

• GCN [29]: it is a spectral-based GNN that learns node embeddings via a localized first-order approximation of spectral graph convolutions. It uses a unsupervised RW-based loss function;
• GraphSAGE [17]: it is a GNN that enables specifying different weights to different nodes in a neighborhood. It uses a unsupervised RW-based loss function;
• GAT [45]: it is a GNN that learns node embeddings inductively from its own feature and the aggregated features of its neighbors. It uses a unsupervised RW-based loss function;
• DeepFD [47]: it is a encoder-decoder structured deep neural network model for fraud detection. It learns user embeddings by minimizing the difference of pairwise distance between user embeddings and a modified Jaccard similarity;
• Dominant [11]: it is a graph auto-encoder based deep neural network model for graph anomaly detection. It reconstructs the graph structure and node attributes to find the anomaly nodes.

In summary, we will compare among **16** methods, including 3 dense block detection methods, 5 graph outlier detection methods, 3 graph embedding methods, and 5 unsupervised GNN methods. We also compare GAL itself among **10** variants: 3 with density losses (+Fraudar, +CatchSync, +LockInfer), 5 with outlier losses (+Fraudar_r, +LockInfer_r, +FraudEagle, +BirdNest, +REV2), and 2 on other GNN methods (GCN+GAL, GAT+GAL).

4.1.3 Implementation details.

For all graph learning methods including ours, the embedding size is 128 and the total sampling size is 100 times of the number of user nodes. For all GNN methods, number of layers are set as 2. For GraphSAGE [17], we use the Mean aggregator. For GAT [45], due to its high computational complexity, the number of heads for hidden layers' self attention is set as 1 in order to have hidden size of 128. For Node2Vec, we use grid search to find the optimal hyperparameters p, q. For Fraudar [22], we use log weighted average degree as the suspiciousness metric. Parameters for other methods are set to typical values as used in previous studies. When training GNNs with GAL, a lightly weighted RW-loss is used as a regularization term to train the model together with GAL.

4.1.4 Evaluation settings.

We use a 3-layer fully-connected feedforward neural network (MLP) as the binary classification model to evaluate the learned embeddings. For Dominant [11], the output anomalous ranking score is used as the input of MLP classifier. For each dataset, the labeled users are divided into training set, validation set and testing set. The MLP model is trained on the training set, and the evaluation results on testing set are reported when the model achieves highest F1 score on validation set. Due to the limited number of labeled

[1] https://github.com/zhao-tong/Graph-Anomaly-Loss.

Table 1: GNNs with graph outlier losses perform the best on Bitcoin-Alpha data.

	Precision	Recall	F1	AUC
Graph outlier detection algorithms				
FRAUDAR_R	0.5714	0.8193	0.6733	0.7659
LOCKINFER_R	0.5512	0.8434	0.6667	0.7593
FRAUDEAGLE [1]	0.4205	0.8916	0.5714	0.5149
BIRDNEST [21]	0.3971	1.0000	0.5685	0.3756
REV2 [32]	0.6471	0.6627	0.6548	0.7094
Raw node features				
MLP	0.8667	0.5652	0.6842	0.7239
Graph embedding methods				
NODE2VEC [16]	0.6522	0.7500	0.6977	0.7606
LINE [44]	0.4722	0.8500	0.6071	0.6455
BINE [15]	0.5143	0.9000	0.6545	0.7409
Graph neural network methods (GNNs)				
GCN [29]	0.5526	0.9545	0.7000	0.7463
GRAPHSAGE [17]	0.6471	0.6111	0.6286	0.7238
GAT [45]	0.6071	0.7727	0.6800	0.7199
DEEPFD [47]	0.3774	1.0000	0.5479	0.4652
DOMINANT [11]	0.4231	1.0000	0.5946	0.3966
GRAPHSAGE + dense block losses (GAL)				
+FRAUDAR	0.7000	0.7368	0.7179	0.8282
+CATCHSYNC	0.7895	0.6818	0.7317	0.7742
+LOCKINFER	0.7143	0.7500	0.7317	0.7970
GRAPHSAGE + graph outlier losses (GAL)				
+FRAUDAR_R	0.7368	0.7778	**0.7568**	**0.8556**
+LOCKINFER_R	0.6800	0.7727	0.7234	0.7874
+FRAUDEAGLE	0.9231	0.6000	0.7273	0.8303
+BIRDNEST	0.7368	0.6364	0.6829	0.7683
+REV2	0.7500	0.7500	0.7500	0.8030
Other GNNs + FRAUDAR_R (GAL)				
GCN+GAL	0.7143	0.7895	0.7500	0.8498
GAT+GAL	0.6333	0.8636	0.7308	0.7273

Table 2: GNNs with dense block losses perform the best on Tencent-Weibo data.

	Precision	Recall	F1	AUC
Dense block detection algorithms				
FRAUDAR [22]	0.9624	0.6198	0.7540	0.9079
CATCHSYNC [25]	0.3200	0.8111	0.4589	0.7752
LOCKINFER [26]	0.9318	0.4562	0.6125	0.8674
Raw node features				
MLP	0.5105	0.7711	0.6143	0.8907
Graph embedding methods				
NODE2VEC [16]	0.9111	0.7218	0.8055	0.9642
LINE [44]	0.8675	0.7606	0.8105	0.9432
BINE [15]	0.3678	0.7007	0.4824	0.8774
Graph neural network mehods (GNNs)				
GCN [29]	0.8373	0.8697	0.8532	0.9690
GRAPHSAGE [17]	0.8200	0.8662	0.8425	0.9780
GAT [45]	0.8586	0.8768	0.8676	0.9706
DEEPFD [47]	0.1080	0.9562	0.1941	0.4980
DOMINANT [11]	0.3006	0.8627	0.4524	0.8167
GRAPHSAGE + dense block losses (GAL)				
+FRAUDAR	0.8669	0.8944	0.8804	0.9777
+CATCHSYNC	0.9067	0.8556	0.8804	0.9685
+LOCKINFER	0.9294	0.8803	**0.9042**	0.9843
GRAPHSAGE + graph outlier losses (GAL)				
+FRAUDAR_R	0.8547	0.8697	0.8621	0.9697
+LOCKINFER_R	0.8591	0.8803	0.8696	0.9757
+FRAUDEAGLE	0.8581	0.8732	0.8656	0.9730
+BIRDNEST	n/a	n/a	n/a	n/a
+REV2	0.8759	0.8944	0.8850	0.9677
Other GNNs + LOCKINFER (GAL)				
GCN+GAL	0.9459	0.8627	0.9024	**0.9853**
GAT+GAL	0.9097	0.8873	0.8984	0.9838

users in the Bitcoin-Alpha dataset, we divide the labeled users into the aforementioned sets differently for 5 times and report the evaluation result with the median F1 score.

For all methods, we report evaluation metrics of Precision, Recall, F1 score, area under ROC curve (AUC), and precision-recall curve for selective methods.

4.2 Results on Bitcoin-Alpha

Table 1 presents the performance of our GAL with different anomaly losses and all baselines for the task of anomaly detection on the Bitcoin-Alpha dataset. Figure 3 (a,c,e) present the precision-recall curves of applying GAL to three basic graph neural algorithms (i.e., GCN, GAT, GRAPHSAGE) with two types of graph outlier losses.

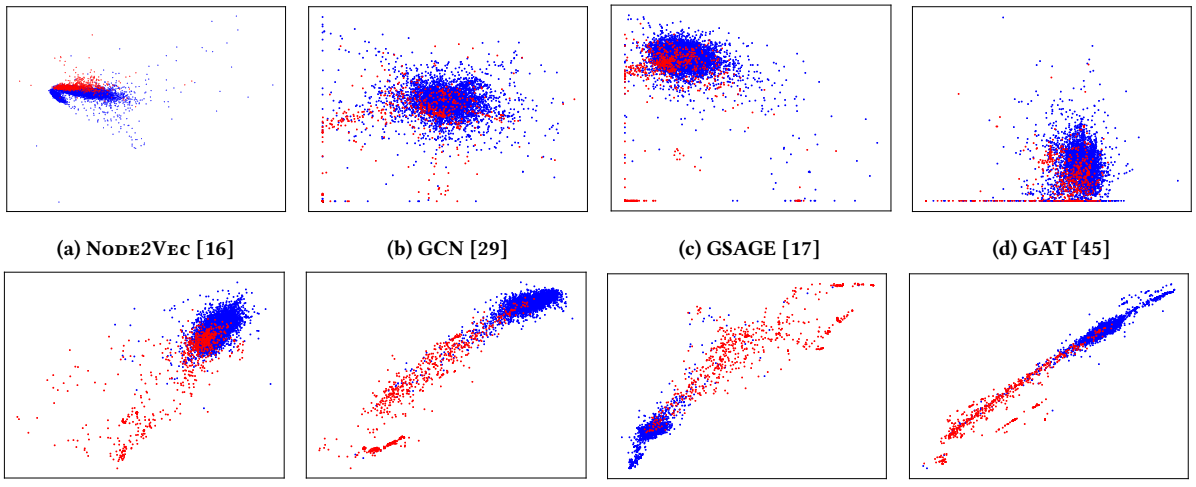
• *The suspicious behaviors on Bitcoin-Alpha are more likely to form outliers than dense subgraphs.* Most of the graph outlier detection methods perform better than dense subgraph detection methods. For example, FRAUDAR_R achieved an F1 of 0.6733, while CATCHSYNC had an F1 of 0.5616.

• *Node features are useful for predicting the labels.* The unsupervised anomaly detection algorithms use only the graph structural information *not* node features. Thus, the best of them, FRAUDAR_R, achieved an F1 of 0.6733, while an MLP classifier with raw features had an F1 of 0.6842.

• *Graph representation learning models, including GNNs, combine both node feature and graph structural information, but the improvement by those with RW-based loss is not significant.* For example, NODE2VEC achieved an F1 of 0.6977 and FRAUDAR_R achieved an F1 of 0.6733. The reason is that the outlier's local neighbors may not be outliers. DEEPFD only obtained an F1 of 0.5479, because it relied heavily on the Jaccard similarity to optimize node embeddings, but the Jaccard similarity only used neighbor's information.

• *GNNs trained by our GAL, graph outlier losses, outperform all baseline methods.* GAL-FRAUDAR_R achieved the best performance: an F1 of 0.7568, an AP of 0.8221, and an AUC of 0.8556. It outperformed the best graph representation learning method GCN relatively by **+8.1%** and **+14.6%** on the two metrics. And it outperforms the best anomaly detection method FRAUDAR_R relatively by **+12.4%** and **+11.7%**. GAL with graph outlier losses can train the GNN models more effectively.

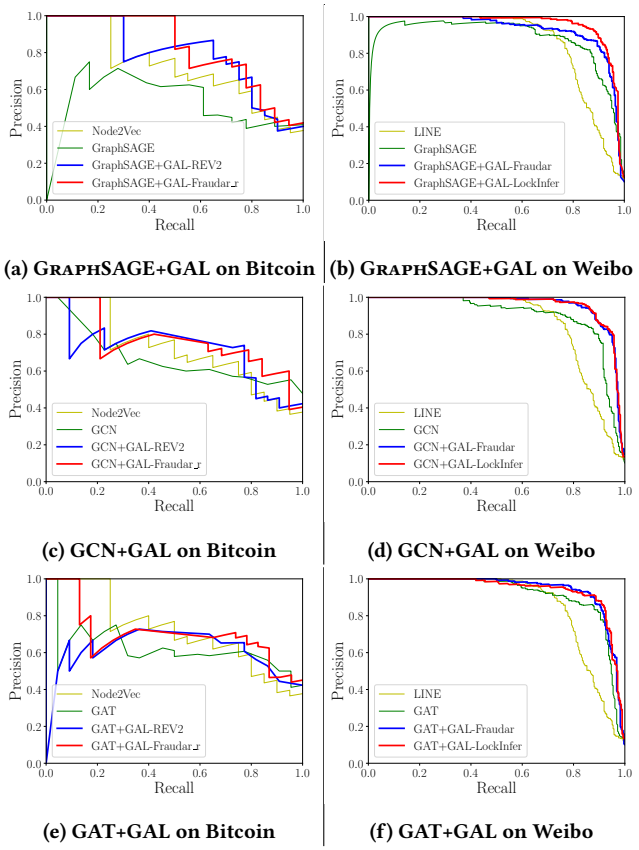
Figure 3 (a,c,e) compare the precision-recall curves of NODE2VEC, a GNN algorithm, and the algorithm with graph outlier losses such as REV2 and FRAUDAR_R. We observe that GRAPHSAGE performs



(a) NODE2VEC [16] (b) GCN [29] (c) GSAGE [17] (d) GAT [45]

(e) GSAGE+FRAUDAR (GAL) (f) GSAGE+LOCKINFER (GAL) (g) GCN + LOCKINFER (GAL) (h) GAT + LOCKINFER (GAL)

Figure 2: Visualizing user embeddings in Tencent-Weibo data. Blue dots represent benign users, and red dots represent anomalous users. Graph anomaly losses (e–h) are better than random-walk loss (b–d).



(a) GRAPHSAGE+GAL on Bitcoin (b) GRAPHSAGE+GAL on Weibo

(c) GCN+GAL on Bitcoin (d) GCN+GAL on Weibo

(e) GAT+GAL on Bitcoin (f) GAT+GAL on Weibo

Figure 3: Precision-recall curves are presented to compare the best graph embedding baseline, a graph neural algorithm, and that equipped with two best types of GALs.

the best among the three GNN algorithms, and the algorithms with graph outlier losses perform better than those without the loss.

4.3 Results on Tencent-Weibo

Table 2 presents the performances on Tencent-Weibo dataset. Note that since the labels are seriously biased, F1 is more representative than AUC. BIRDNEST could not work due to the absence of timestamps. Figure 3 (b,d,f) show the precision-recall curves of the three best baselines and GAL with two types of graph density losses.

• *The suspicious behaviors on Tencent-Weibo are more likely to form dense subgraphs than outliers.* Dense subgraph detection methods perform much better than outlier detection methods. FRAUDAR achieved an F1 of 0.7540, while FRAUDEAGLE only made an F1 of 0.4102. The reason is that fraudsters had to post a large number of messages in group to inflate popularity of hashtag.

• *Local neighborhood is more informative than pure global structure, as graph embedding models perform better than the dense subgraph detection algorithms.* For example, LINE achieved an F1 of 0.8105, while FRAUDAR achieved an F1 of 0.7540. The models that use local structures for embedding aggregation can preserve the user similarity of being in the same blocks.

• *GNNs trained by our GAL, dense block losses, outperform all baseline methods.* GAL-LOCKINFER performed the best: an F1 of 0.9042 and an AUC of 0.9843. It outperformed the best graph embedding method LINE relatively by **+11.6%** and **+4.4%** on the two metrics. And it outperformed GRAPHSAGE relatively by **+7.3%** and **+0.6%**. GAL with dense block losses can train the GNN models more effectively w.r.t. the task of detecting grouping behaviors, than traditional RW-based loss.

Figure 3 (b,d,f) compare the precision-recall curves of LINE, a GNN algorithm, and the algorithm with dense block losses such as FRAUDAR and LOCKINFER. We observe that GRAPHSAGE performs the best among the three GNN algorithms, and the algorithms with dense block losses perform better than those without the loss.

Table 3: Ablation study results on Tencent-Weibo data.

Backbone	Setting	F1	AUC
GraphSAGE	original	0.8425	0.9780
	+GAL-Random	0.8448	0.9685
	+GAL-SingleBlock	0.8754	0.9757
	+GAL-Fraudar	0.8804	0.9777
	+GAL-LockInfer	**0.9042**	**0.9843**
GCN	original	0.8532	0.9690
	+GAL-random	0.8735	0.9757
	+GAL-SingleBlock	0.8909	0.9775
	+GAL-Fraudar	0.8982	0.9814
	+GAL-LockInfer	**0.9024**	**0.9853**
GAT	original	0.8676	0.9706
	+GAL-random	0.8709	0.9749
	+GAL-SingleBlock	0.8722	0.9754
	+GAL-Fraudar	0.8822	**0.9921**
	+GAL-LockInfer	**0.8984**	0.9838

Embedding visualization. In Figure 2, we show the embeddings of users in the Tencent-Weibo dataset given by four baselines of graph embedding (a–d) with RW-based losses and our GAL with different GNNs (e–h). The 128-dim embeddings are projected to 2-dim space. From the plots, we find that the embeddings of GAL with dense block losses can better separate the *blue* benign users and *red* suspicious users. Moreover, GAL-LockInfer on all GNN frameworks show the best embedding spaces of a clear separation between the two classes, consistent with the results in Table 2.

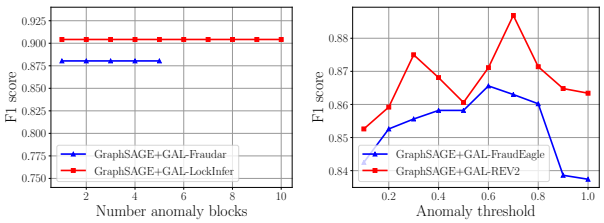
4.4 Ablation Study

In addition to analyze the effectiveness of GAL, we conduct an ablation study to examine the contribution of different components by having different settings of each GNN backbone:

- Original: original GNNs under the bipartite setting trained by the RW-based loss function (Eq. 1).
- +GAL-Random: instead of estimating the parameters of GAL with the result of existing unsupervised methods as described in Section 3.4, we randomly generate the anomaly labels.
- +GAL-SingleBlock: when training by GAL with dense block loss, instead of separating all anomaly dense blocks, treat all anomal dense block as one single user block.
- +GAL-Fraudar/LockInfer: GNNs trained by different dense block losses: GAL-Fraudar and GAL-LockInfer.

Table 3 summarizes the results of the ablative study, from which we have the following observations:

- *The GNNs trained by GAL outperform original GNNs.* LockInfer is the best option among the dense block detection algorithms for generating block labels in GAL.
- *Labels from existing unsupervised methods give useful information.* When the results of existing unsupervised graph anomaly detection methods are substituted by random pseudo labels (+GAL-Random), performances of GNNs train by GAL decreases, indicating that the random labels do not give the model useful information.
- *Separating different dense blocks helps.* The performance of GNNs trained by +GAL-SingleBlock are not comparable with the performances of GNNs trained by GAL with dense block losses, which indicates that the design of separating the embeddings of anomaly users from different dense blocks (Section 3.4.2) is able to learn embeddings that are more separatable.



(a) Number of anomaly blocks **(b) Anomaly threshold**

Figure 4: Parameter sensitivity of GAL on GNNs.

4.5 Sensitivity Analysis

When utilizing outlier scores or dense blocks to estimate parameters of GAL as described in Section 3.4, a classification threshold is needed to determine which users are labelled as potential anomalies. In dense block detection algorithms, different blocks have their unique labels instead of a too general positive label. With the multi-class dense block loss, the node embeddings would not change significantly when the threshold changed (and the number of blocks changed), as shown in Figure 4a. In outlier detection algorithms, the node outlierness scores are usually continuous and do not show obvious gap. We need parameter search to determine the best threshold. Figure 4b shows that the performance of GraphSAGE+GAL is not sensitive to this parameter.

5 CONCLUSIONS

In this work, we presented a novel Graph Anomaly Losses (GAL) that is able to unsupervisedly train GNNs for anomaly-detectable node representations. GAL has a bounded test error and GAL with *graph outlier losses* and *dense block losses* evaluates node similarity based on the global properties discovered by graph mining algorithms. Experiments on two real-world datasets demonstrated that (i) GNNs with GAL significantly outperformed 17 baseline methods and (ii) GAL trained the models more effectively than traditional RW-based loss on various state-of-the-art GNN frameworks. The limitation of the proposed loss is that factual anomalous labels may have little or zero correlation with outlier or dense block patterns. Fortunately, it is rare in our observations.

ACKNOWLEDGMENTS

This work is supported by National Science Foundation IIS-1849816.

REFERENCES

[1] Leman Akoglu, Rishi Chandy, and Christos Faloutsos. 2013. Opinion fraud detection in online reviews by network effects. In *Seventh international AAAI conference on weblogs and social media*.
[2] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. 2010. Oddball: Spotting anomalies in weighted graphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 410–421.
[3] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery* 29, 3 (2015), 626–688.
[4] Reid Andersen. 2010. A local algorithm for finding dense subgraphs. *ACM Transactions on Algorithms (TALG)* 6, 4 (2010), 60.
[5] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. 2000. Greedily finding a dense subgraph. *Journal of Algorithms* 34, 2 (2000), 203–221.

[6] Sambaran Bandyopadhyay, Saley Vishal Vivek, and MN Murty. 2020. Outlier Resistant Unsupervised Deep Architectures for Attributed Network Embedding. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 25–33.

[7] Kaidi Cao, Colin Wei, Adrien Gaidon, Nikos Arechiga, and Tengyu Ma. 2019. Learning Imbalanced Datasets with Label-Distribution-Aware Margin Loss. *arXiv preprint arXiv:1906.07413* (2019).

[8] Deepayan Chakrabarti. 2004. Autopart: Parameter-free graph partitioning and outlier detection. In *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 112–124.

[9] Nitesh V Chawla. 2003. C4. 5 and imbalanced data sets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure. In *Proceedings of the ICML*, Vol. 3. 66.

[10] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[11] Kaize Ding, Jundong Li, Rohit Bhanushali, and Huan Liu. 2019. Deep anomaly detection on attributed networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, 594–602.

[12] Kaize Ding, Yichuan Li, Jundong Li, Chenghao Liu, and Huan Liu. 2019. Graph Neural Networks with High-order Feature Interactions. *arXiv preprint arXiv:1908.07110* (2019).

[13] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. 2018. Spotlight: Detecting anomalies in streaming graphs. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1378–1386.

[14] Jing Gao, Feng Liang, Wei Fan, Chi Wang, Yizhou Sun, and Jiawei Han. 2010. On community outliers and their efficient detection in information networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 813–822.

[15] Ming Gao, Leihui Chen, Xiangnan He, and Aoying Zhou. 2018. BiNE: Bipartite Network Embedding.. In *SIGIR*. 715–724.

[16] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[17] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[18] Ville Hautamaki, Ismo Karkkainen, and Pasi Franti. 2004. Outlier detection using k-nearest neighbour graph. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, Vol. 3. IEEE, 430–433.

[19] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. 2008. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. IEEE, 1322–1328.

[20] Haibo He and Edwardo A Garcia. 2009. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering* 21, 9 (2009), 1263–1284.

[21] Bryan Hooi, Neil Shah, Alex Beutel, Stephan Günnemann, Leman Akoglu, Mohit Kumar, Disha Makhija, and Christos Faloutsos. 2016. Birdnest: Bayesian inference for ratings-fraud detection. In *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 495–503.

[22] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. Fraudar: Bounding graph fraud in the face of camouflage. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 895–904.

[23] Meng Jiang. 2016. Catching Social Media Advertisers with Strategy Analysis. In *Proceedings of the First International Workshop on Computational Methods for CyberSafety*. ACM, 5–10.

[24] Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos. 2016. Spotting suspicious behaviors in multimodal data: A general metric and algorithms. *IEEE Transactions on Knowledge and Data Engineering* 28, 8 (2016), 2187–2200.

[25] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2014. Catchsync: catching synchronized behavior in large directed graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 941–950.

[26] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2016. Inferring lockstep behavior from connectivity pattern in large graphs. *Knowledge and Information Systems* 48, 2 (2016), 399–428.

[27] Parisa Kaghazgaran, James Caverlee, and Anna Squicciarini. 2018. Combating crowdsourced review manipulators: A neighborhood-based approach. In *Proceedings of the 11th International Conference on Web Search and Data Mining*. 306–314.

[28] Sham M Kakade, Karthik Sridharan, and Ambuj Tewari. 2009. On the complexity of linear prediction: Risk bounds, margin bounds, and regularization. In *Advances in neural information processing systems*. 793–800.

[29] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[30] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[31] Vladimir Koltchinskii, Dmitry Panchenko, et al. 2002. Empirical margin distributions and bounding the generalization error of combined classifiers. *The Annals of Statistics* 30, 1 (2002), 1–50.

[32] Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian. 2018. Rev2: Fraudulent user prediction in rating platforms. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 333–341.

[33] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. 2016. Edge weight prediction in weighted signed networks. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 221–230.

[34] David Mease, Abraham J Wyner, and Andreas Buja. 2007. Boosted classification trees and class probability/quantile estimation. *Journal of Machine Learning Research* 8, Mar (2007), 409–439.

[35] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *International conference on machine learning*. 2014–2023.

[36] Guansong Pang, Longbing Cao, Ling Chen, and Huan Liu. 2018. Learning representations of ultrahigh-dimensional data for random distance-based outlier detection. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2041–2050.

[37] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[38] B Aditya Prakash, Ashwin Sridharan, Mukund Seshadri, Sridhar Machiraju, and Christos Faloutsos. 2010. Eigenspokes: Surprising patterns and scalable community chipping in large graphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 435–448.

[39] Shebuti Rayana and Leman Akoglu. 2015. Collective opinion spam detection: Bridging review networks and metadata. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 985–994.

[40] Neil Shah. 2017. FLOCK: Combating astroturfing on livestreaming platforms. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1083–1091.

[41] Neil Shah, Alex Beutel, Brian Gallagher, and Christos Faloutsos. 2014. Spotting suspicious link behavior with fbox: An adversarial perspective. In *2014 IEEE International Conference on Data Mining*. IEEE, 959–964.

[42] Kijung Shin, Bryan Hooi, and Christos Faloutsos. 2016. M-zoom: Fast dense-block detection in tensors with quality guarantees. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 264–280.

[43] Jimeng Sun, Huiming Qu, Deepayan Chakrabarti, and Christos Faloutsos. 2005. Neighborhood formation and anomaly detection in bipartite graphs. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*. IEEE, 8–pp.

[44] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 1067–1077.

[45] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[46] Daheng Wang, Meng Jiang, Munira Syed, Oliver Conway, Vishal Juneja, Sriram Subramanian, and Nitesh V Chawla. 2020. Calendar Graph Neural Networks for Modeling Time Structures in Spatiotemporal User Behaviors. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[47] Haibo Wang, Chuan Zhou, Jia Wu, Weizhen Dang, Xingquan Zhu, and Jilong Wang. 2018. Deep structure learning for fraud detection. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 567–576.

[48] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks? *arXiv preprint arXiv:1810.00826* (2018).

[49] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 974–983.

[50] Wenhao Yu, Mengxia Yu, Tong Zhao, and Meng Jiang. 2020. Identifying referential intention with heterogeneous contexts. In *Proceedings of The Web Conference 2020*. 962–972.

[51] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 793–803.

[52] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. 2020. Data Augmentation for Graph Neural Networks. *arXiv preprint arXiv:2006.06830* (2020).

[53] Tong Zhao, Matthew Malir, and Meng Jiang. 2018. Actionable objective optimization for suspicious behavior detection on large bipartite graphs. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 1248–1257.