IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 12, 2022, accepted May 9, 2022, date of publication May 16, 2022, date of current version May 23, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3175581

A Graph Representation Learning Algorithm for Approximate Local Symmetry Feature Extraction to Enhance Malicious Device Detection Preprocessing

YIRAN HAO[1,2], QUAN LU[2], AND XINMING CHEN[2]
[1]Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
[2]Alibaba Group, Beijing 100016, China

Corresponding author: Yiran Hao (aurora.hyr@alibaba-inc.com)

ABSTRACT Existing malicious device detection preprocessing ignores the topological similarity of node neighborhood structures in the network. According to the structural equivalence hypothesis, devices with approximately local symmetry in the device-account graph have similar topological embeddings after preprocessing. In order to improve the performance of malicious device detection, we propose the Graph Structural-topic Similar Subgraph Merging, abbreviated GraphSTSGM, to extract topological similarity between nodes. GraphSTSGM extracts approximate local symmetry features by adding local neighborhood structural patterns merge to Graph Structural-topic Neural Network (GraphSTONE). In this algorithm, we build a device-account relationship graph G with devices and accounts as nodes, build an edge between associated devices and accounts, and then calculate the approximate local symmetry of each device via the merge-similar-substructures-based anonymous walk in G. Then, the approximate local symmetry features and device features of the nodes are aggregated through Graph Convolutional Network (GCN). We use the above algorithm as a preprocessing method to enhance the ability of malicious device detection by accurately characterizing the approximate local symmetry features of nodes. Finally, the obtained aggregated features are used as the input of the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) model for malicious device detection. Experiments based on Alibaba Cloud Security data show that the proposal outperforms the state-of-the-art algorithms by 3.6% with respect to the AUC of malicious device detection. In addition, experiments based on the graph dataset Cora show that the proposal outperforms the state-of-the-art algorithms by 2.6% with respect to the AUC of node classification.

INDEX TERMS Approximate local symmetry features, anonymous walk, automatic preprocessing, graph representation, graph structural topic neural network, local neighborhood structural patterns, malicious device detection.

I. INTRODUCTION

With the development of the Internet, many businesses will attract users by issuing coupons and other marketing activities on terminal devices, such as: Android Phone and iOS Phone [1], [2]. However, these marketing activities also attract a lot of malicious users, who usually profit from malicious behaviors such as registered spam accounts and account theft. However, this kind of malicious behavior will cause the merchants to invest a lot of money in marketing activities without really attracting users and causing a lot of losses. Therefore, how to accurately detect malicious users' devices has become an important issue.

The malicious device detection algorithm mainly includes the rule-based malicious device detection algorithm, machine learning-based malicious device detection algorithms and the graph-based malicious device detection algorithm [3]. Rule-based malicious device detection algorithms have high detection efficiency for known malicious. Machine learning-based malicious device detection algorithms are able to detect unknown attacks. However, a common disadvantage of the above algorithms is that they cannot detect malicious devices that are not in the extracted features. Users of malicious devices invent new malicious patterns and continuously

The associate editor coordinating the review of this manuscript and approving it for publication was Nazar Zaki.

change their strategies to avoid detection. In order to detect new malicious attacks, it is necessary to continuously add features extracted by expert experience. The effect of continuously increasing individual feature engineering is a logarithmic curve, but the cost is an exponential curve. Therefore, how to automatically detect malicious devices that are not covered by features becomes a problem. The graph-based malicious device detection algorithm is a good solution to the above problem. The attacker's resources such as devices and accounts are limited, and there are usually close associations between several devices and some accounts [3]. The association between the attacker's several devices and accounts is different from that of normal users. Therefore, we need to detect devices that have similar device and account relationships to known attackers in the device-account graph. Structural equivalence [4] is a good way to solve the above problem.

Graph-based malicious device detection algorithms are able to extract the structural equivalence to detect devices with similar functions in the graph. Structural equivalence [4] means that the substructures that reconstruct the local neighborhood of two nodes are the same or similar. Two nodes with structural equivalence are considered to have local symmetry. Local symmetry includes complete local symmetry and approximate local symmetry. In detail, complete local symmetry means the substructure of the reconstructed local neighborhood of two nodes is exactly the same. Approximate local symmetry means the substructures that reconstruct the local neighborhood of two nodes are similar. According to the structural equivalence hypothesis [4], devices with approximately local symmetry in the device-account graph have similar topological embeddings after preprocessing. The more similar the topological structures within the node neighborhood, the more similar the functions the two nodes have. As shown in Figure 1, complete local symmetry cannot characterize the similarity of local neighborhoods, and approximate local symmetry can characterize the similarity of local neighborhoods. Existing graph-based malicious device detection can extract completely local symmetric features, but cannot extract approximate local symmetric features. The device preprocessing methods suffer from low performance due to the lack of approximate local symmetry features in the extracted features [5]. Therefore, we introduce approximate local symmetry to characterize the similarity of the neighborhood structure between devices.

In order to improve the performance of malicious device detection, we propose the Graph Structural-topic Similar Subgraph Merging, abbreviated GraphSTSGM, to extract topological similarity between nodes. GraphSTSGM extracts approximate local symmetry features by adding local neighborhood structural patterns merge to Graph Structural-topic Neural Network (GraphSTONE).

The motivation of introducing a graph representation algorithm for device preprocessing to obtain approximately locally symmetric features of nodes is as follows. An interesting analogy is that in every company there is the role of the boss, and in each company the boss has many employees, and each employee has only one boss. The interpersonal topology of each boss is similar, and the interpersonal topology of each employee is also similar. As shown in Figure 2, the neighborhood structure of the boss relationship in most companies is shown in (a), and the neighborhood structure of the boss relationship in a few companies is shown in (b). The v_1 node in Figure 2(a) and the v_4 node in Figure 2(b) have complete local symmetry. The v_1 node in Figure 2(a) and the v_2 node in Figure 2(b) have approximate local symmetry. The v_3 and the v_5 have approximate local symmetry. Existing graph representation learning algorithms have the ability to find nodes with complete local symmetry and consider nodes with the same neighborhood structure to have the same function, but cannot find nodes with approximate local symmetry. Therefore, existing graph representation learning algorithms cannot distinguish whether v_3 and v_5 have the same function. In addition, existing methods are also unable to distinguish whether the red node is a boss or an employee.In fact, the neighborhood structure of the red node in (b) is similar to that of the v_3 node in (a). The two nodes are approximately locally symmetrical. Therefore, it is necessary for us to distinguish whether nodes have the same function by whether their local neighborhoods are approximately locally symmetric. The stronger the approximate local symmetry of a node's neighborhood structure topology is, the more similar the node's embeddings are [6]. It can be concluded that analyzing the similarity of node neighborhood topology has the ability to detect malicious devices more flexibly and accurately. Moreover, effective preprocessing is the basis for improving malicious device detection performance, because the preprocessing directly affect the final performance of malicious device detection. Therefore, this paper proposes the GraphSTSGM learning algorithm to preprocess the device neighborhood structure. The algorithm preprocesses the neighborhood structure of the device to obtain an approximately locally symmetric embedding.

The main contributions of this paper are as follows.

1) In order to improve the performance of malicious device detection, we propose the GraphSTSGM algorithm to extract topological similarity between nodes. GraphSTSGM extracts approximate local symmetry features by adding local neighborhood structural patterns merge to GraphSTONE. We use feature embeddings as input to Density-Based Spatial Clustering of Applications with Noise (DBSCAN) for malicious device detection. The system is named by GraphSTSGM-DBSCAN.
2) In this algorithm, we use anonymous walks to automatically extract the substructures that constitute the neighborhood structure of each device, and then merge similar substructures to obtain a set of similar substructures. In order to get the topic substructures of the node neighborhood more accurately, the device neighborhood structure is reconstructed using the topic substructures to obtain a structure embedding for each

device. Finally, we use cluster detection to obtain malicious devices with similar structural embeddings to known malicious devices. In addition, abnormal nodes are identified by finding that the node neighborhood structure cannot be reconstructed with the topic substructures. This method is superior to the existing methods that require manual experience to pre-define abnormal structures, and then obtain the abnormal structures through structure matching.

3) In this paper, the penalty-based similar substructure set merging is added on the basis of the existing graph representation algorithm GraphSTONE [6]. This method is able to merge substructures that appear infrequently and are similar to frequently occurring substructures. Therefore, this method has the ability to extract the approximate local symmetry features of the device more accurately, and thus more accurately describe the neighborhood structure similarity relationship between the devices.
4) The approximate local symmetry features of the device can be accurately extracted by GraphSTSGM. When the merge threshold η in GraphSTSGM is infinite, the GraphSTSGM algorithm is equivalent to GraphSTONE. Experiments show that the GraphSTSGM algorithm with the addition of similar substructure merging has better performance. In the best case, the AUC and the recall of GraphSTSGM-DBSCAN reached 89.2% and 87.1%, respectively. In the worst case, the AUC and the recall of GraphSTSGM-DBSCAN reached 86.5% and 86.6%, respectively. In the worst case, the proposed algorithm achieves the good performances regarding the AUC exceeding those of the other state-of-the-art algorithms by 0.9%.
5) An empirical formula, i.e., the benefits of combining substructure $h\left(s_i, s_j\right)$-penalty for merging substructure $g\left(s_i, s_j\right)>=$threshold η, is designed to calculate the whether the two substructures should be merged. The experimental results show that the calculation of this formula has the ability to obtain better malicious device detection performance.

Section 2 describes related work. Section 3 introduces the malicious device detection based on device preprocess using GraphSTSGM-DBSCAN. Section 4 introduces the experiment. Section 5 discusses the results. Section 6 concludes the paper.

II. RELATED WORK

A. MALICIOUS DEVICE DETECTION ALGORITHM

The malicious device detection algorithm mainly includes the rule-based malicious device detection algorithm, machine learning-based malicious device detection algorithms and the graph-based malicious device detection algorithm [3]. In detail, the rule-based malicious device detection algorithms are generally considered to be based on user profiling and feature engineering. The rule-based malicious device detection algorithm has high detection efficiency for known malicious. Machine-learning based malicious device detection algorithms are able to detect unknown attacks. In recent years, there have been some studies based on machine learning for malicious device detection. Machine-learning based malicious device detection algorithms mainly include Support Vector Machines (SVM) [7], [8], tree-based models [9], [10] and neural networks [11]-[13], etc. In 2015, Manjula [8] detected malicious devices by finding outliers and then using Support Vector Machines (SVM) to classify the outliers; In 2021, Sun *et al.* [10] use an optimized boosting model to detect malicious devices by combining prior knowledge and feature importance analysis. The method is interpretable; In 2021, Benchaji *et al.* [13] used the historical behavior information of accounts as features. First, the algorithm normalizes the features. After that, a Long Short-Term Memory (LSTM) model is trained using historical behavioral sequence data. Finally, the algorithm uses LSTM on the predicted data to detect malicious devices; A common disadvantage of the above algorithms is that they cannot detect malicious devices that are not in the extracted features. Malicious device users invent new malicious patterns and continuously change their strategies to avoid detection [13]. In order to detect new malicious attacks, it is necessary to continuously add features extracted by expert experience. The effect of continuously increasing individual feature engineering is a logarithmic curve, but the cost is an exponential curve. Therefore, malicious device detection algorithms need to transform from rule-based and machine learning-based feature engineering to global network engineering [3]. Malicious device detection in the network needs to obtain features that accurately describe the topology similarity of the device neighborhood structure. We usually detect malicious devices based on graph algorithms.

In recent years, there have been some graph-based malicious device detection algorithms. In 2021, Motschnig [14] *et al.* used the heuristic Louvain algorithm for community grouping. Louvain algorithm can obtain the division method with the largest community modularity, but this method only considers the structural association information, and does not consider the structural topological similarity of nodes in the network; In addition, the time complexity of this method is too high, and it cannot run in the case of hundreds of millions of points and billions of edges; In 2020, Santra *et al.* [15] proposed a community definition for Heterogeneous MultiLayer Networks (HeMLNs) that preserves semantics and structure. In 2017, Pandhre *et al.* [16] proposed a method that detects novel outlier graph nodes by taking into account the node and edge simultaneously to detect anomalies. This method can find outliers in heterogeneous graphs with multiple edge relationships, but the disadvantage of this method is that only abnormal nodes can be detected, and devices similar to known malicious devices cannot be detected.

The common disadvantage of the above methods is that the preprocessing only considers the tightness of the

connection between the current node and surrounding neighbor nodes, but ignores the local symmetric relationship between nodes [17], resulting in a low AUC for malicious device detection. Therefore, we introduce a graph representation learning algorithm to solve the above problem.

B. GRAPH REPRESENTATION LEARNING

Graph representation learning can automatically extract structural topology information between nodes [17], such as local symmetry, etc. This algorithm has been widely used in recommender systems, e-commence networks [16], etc. The existing graph representation learning algorithms are mainly the following, such as: struc2vec [17], ComE [18], Metapath [20]. In 2017, the ComE [18] method was proposed. This method uses structural information, but lacks approximate local symmetry features; The Struc2vec [17] algorithm distinguishes whether two nodes have similar structure only by the degree of node neighborhood structure. This method can roughly distinguish whether the neighborhood structures of nodes are similar, but ignores the approximate local symmetry of nodes, resulting in low AUC. Metapath [20] can extract the structural topology information between heterogeneous graph nodes, but this method requires manual experience to pre-define the random walk paths to obtain the pre-defined substructures, and cannot automatically capture abnormal substructures;

The common disadvantage of the above algorithms is that they do not have the ability to automatically distinguish the structural topology between malicious nodes, and require manual experience or feature engineering skills to process the structural information between nodes. GraphSTONE [6] has the ability to automatically distinguish the structural topology of malicious nodes. GraphSTONE uses anonymous random walks to obtain substructures to characterize the topological similarity of nodes. GraphSTONE is able to extract the complete local symmetry of nodes, but this method ignores the similarity between substructures, so it cannot extract approximate local symmetry of nodes. GraphSTONE ignores approximate neighborhood symmetry features that characterize the similarity between substructures. Therefore, GraphSTONE may cause nodes with normal neighborhood structure to be mistakenly considered abnormal nodes. Our proposed algorithm has the ability to automatically extract approximate local symmetry features that characterize the structural similarity of node neighborhoods. Finally, the approximate local symmetry features and the original information of nodes are obtained as the input of DBSCAN.

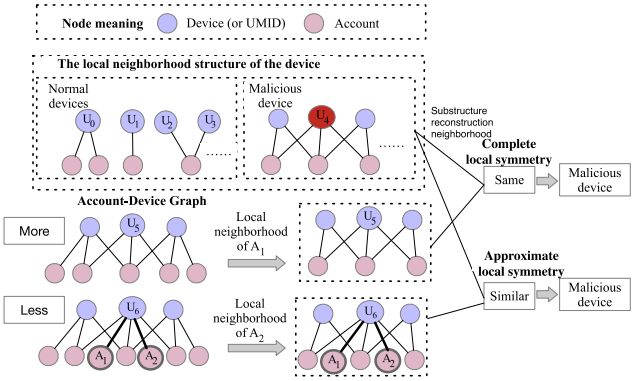


FIGURE 1. An example illustrating that approximate local symmetry has higher performance than complete local symmetry. The local neighborhood structure of U_5 in the account-device graph is the same as that of U_4, so U_4 and U_5 have complete local symmetry. The local neighborhood structure of U_6 in the account-device graph is similar to that of U_4, so U_4 and U_6 are approximate local symmetry. Existing methods can detect devices U_5 with complete local symmetry, but cannot detect devices with approximate local symmetry. Complete local symmetry is a special case of approximate local symmetry, and devices U_5 and U_6 can be detected using approximate local symmetry. Therefore, approximate local symmetry has higher performance than complete local symmetry.

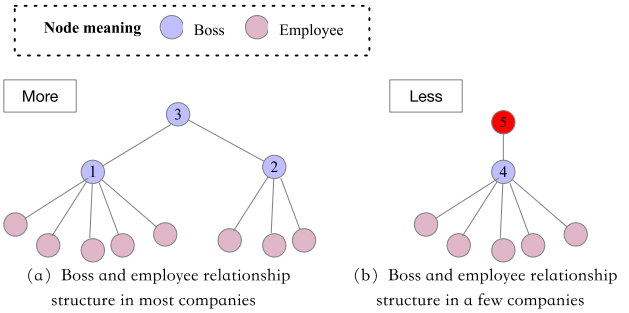


FIGURE 2. An example in social network of substructure similarity is the same functional node. The neighborhood structure of the boss relationship in most companies is shown in (a), and the neighborhood structure of the boss relationship in a few companies is shown in (b). Using the GraphSTONE algorithm, the neighborhood structure of the red node in (b) will be considered an abnormal substructure, and it is impossible to confirm whether the node is a boss or an employee. In fact, the neighborhood structure of the red node in (b) is similar to that in (a).

III. PROPOSED MALICIOUS DEVICE DETECTION ALGORITHM

A. DEFINITION

Unique mobile identity (alias name: Node/Device): It refers to establishing a globally unique device ID for each operating device to uniquely represent the device [19]. In the following, the unique mobile identification is called the device or node;

Device-account graph: It refers to taking the device and the account as nodes, and constructing an edge between the associated device and the account to obtain the device-account graph[21];

Local proximity(alias name: Local neighborhood): It refers to capturing relationships within two hops from a node, and it describes the neighborhood structure of a node. In Figure 3, nodes v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12} are the local neighborhoods of v_i [22].

Anonymous Walks [6]: It refers to sampling in the graph by random walk [4], the sequence obtained by sampling is "v_0-v_9-v_8-v_{11}-v_9", and the first node v_0 that appears from the left is marked as 0. We construct a dictionary S, store v_0 as the key and 0 as the value in S, S={v_0:0}. The second

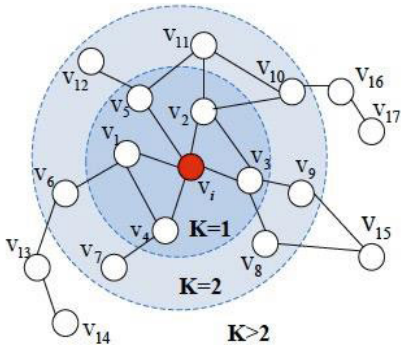


FIGURE 3. K-order neighborhood graph [22].

node v_9 appears, because v_9 is not in S, so v_9 is stored as key and 1 is stored in S as value. At this time, S={v_0:0, v_9:1}. And so on, until the sequence is traversed. Therefore, the sequence "v_0-v_9-v_8-v_{11}-v_9" can be obtained as "0-1-2-3-1" after anonymity. As shown in the Figure 4(a), it is the process of anonymous walk.

Local Neighborhood Structural Patterns (alias name: Substructure): It refers to the substructure obtained by anonymous walks in the local neighborhood of a node. For example, random walk can get sequence "v_0-v_9-v_8-v_{11}-v_9" and sequence "v_0-v_9-v_{18}-v_{19}-v_9"; In detail, the sequence "v_0-v_9-v_8-v_{11}-v_9" becomes "0-1-2-3-1" after anonymity, the sequence "v_0-v_9-v_{18}-v_{19}-v_9" is also "0-1-2-3-1" after anonymity. The "1-2-3-1" sequence represents a triangle, that is, a substructure of a triangle is extracted;

Topic substructure: It refers to selecting the top-N substructures from the substructures that can reconstruct the structure of most node neighborhoods. As shown in the Figure 4(a).

Neighborhood Structure Reconstruction: It refers to the use of topic substructure to reconstruct the neighborhood structure of the node, and has obtained the embedding of the neighborhood structure of the node.

Homophily hypothesis [4]: It means that the nodes in the graph are highly interconnected. The more common neighbors between two nodes, the stronger the correlation between the two nodes. As shown in Figure 5(b), the two red nodes in the figure are the nodes of homogeneous equivalence;

Structural equivalence [4](alias name: Local symmetry): It refers to that the substructures that reconstruct the local neighborhood of two nodes are the same or similar. Nodes of structural equivalence have the same function. Two nodes with structural equivalence are considered to have local symmetry. Local symmetry includes complete local symmetry and approximate local symmetry. In detail, complete local symmetry means the substructure of the reconstructed local neighborhood of two nodes is exactly the same. Approximate local symmetry means the substructures that reconstruct the local neighborhood of two nodes are similar. As shown in Figure 5(a), node u and node v are structurally equivalent. Importantly, unlike homophily, structural equivalence does not emphasize connectivity; nodes could be far apart in the network and still have the same structural role [4]. Figure 2 shows an example of a social network where nodes of structural equivalence have the same function. In every company there are bosses and employees, each boss usually has many employees, and each employee usually has only one boss. In Figure 2, all pink nodes have only one parent node, and the neighborhood structures of all pink nodes are similar. That is, all pink nodes are employees, and employees have the same function in every company. All pink nodes are structural equivalence; In Figure 2, all purple nodes have many child nodes, and the neighborhood structure of all purple nodes is similar. That is, all purple nodes are bosses, and bosses have the same function in every company. All purple nodes are structural equivalence.

Complete local symmetry: It means that the substructure of the reconstructed local neighborhood of two nodes is exactly the same. Complete local symmetry is one type of local symmetry. Nodes with complete local symmetry have the same function. The v_1 node in Figure 2(a) and the v_4 node in Figure 2(b) have complete local symmetry;

Approximate local symmetry: It means that the substructures that reconstruct the local neighborhood of two nodes are approximately the same. That is, approximate local symmetry can characterize the topological similarity between the neighborhood structures of two nodes. Approximate local symmetry is one type of local symmetry. Nodes with approximate local symmetry have the same function. The v_1 node in Figure 2(a) and the v_2 node in Figure 2(b) have approximate local symmetry;

Approximate local symmetry feature: It refers to extracting embeddings that can characterize the approximate local symmetric relationship between nodes.

Automatic preprocessing: It refers to the automatic extraction of features that can characterize nodes. This method does not require any manual experience and feature engineering skills;

Malicious Device Detection: It refers to the detection of malicious terminal devices or abnormal terminal devices. In detail, the malicious devices in this article refer to devices that are similar to known malicious devices and anomalous devices that differ from most normal devices. It is worth noting that in the device-account graph, we reconstruct the neighborhood structure of the device and account neighborhoods to obtain device embeddings, and cluster the embeddings to detect malicious devices.

B. FLOWCHART

Malicious device detection is mainly divided into three parts. The first part is to generate unique mobile identity for each traffic using the method in [19], each unique mobile identity represents a unique device. The second part is automatic

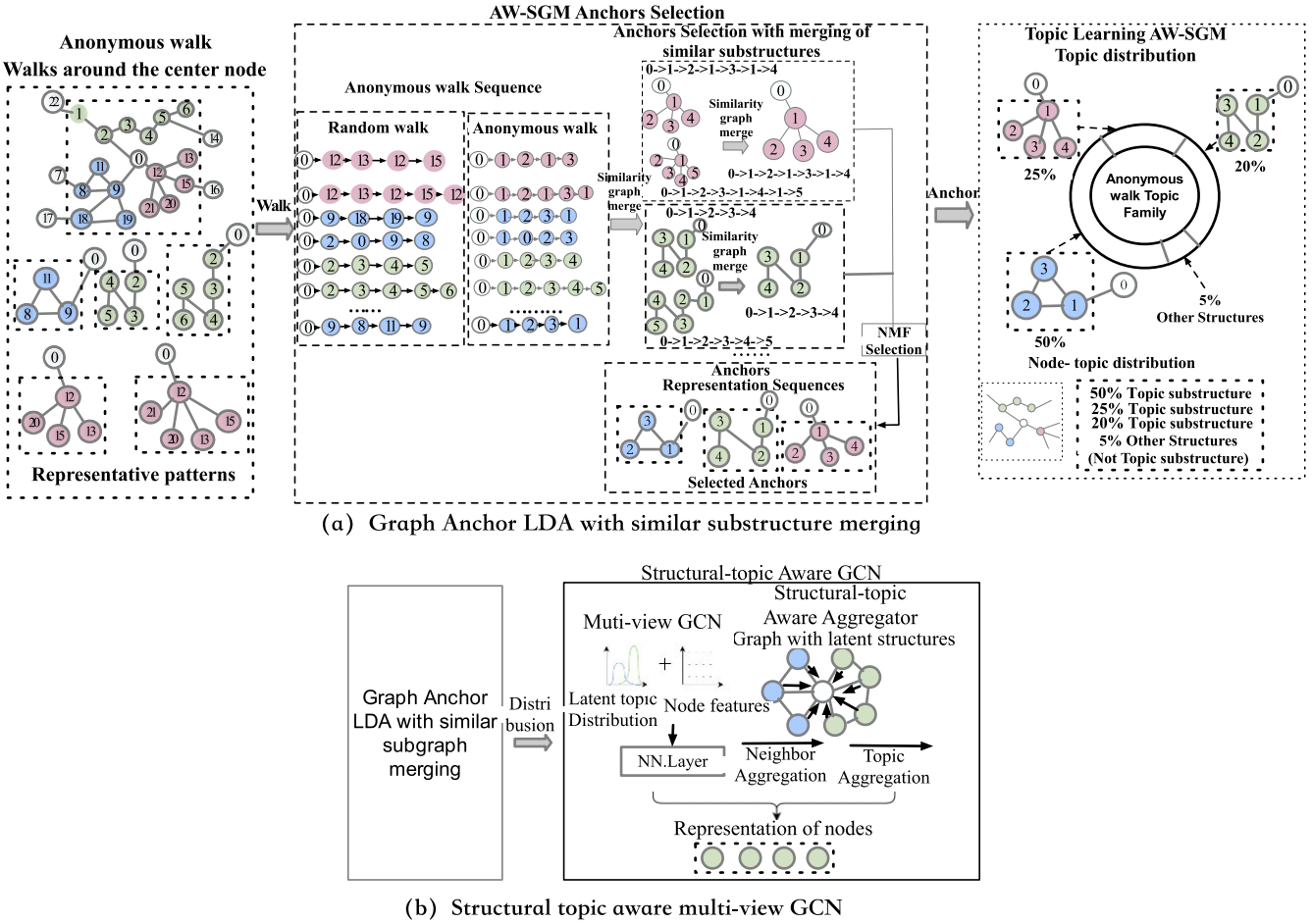


FIGURE 4. Schematic diagram of the GraphSTSGM algorithm. GraphSTSGM consists of two major components: (a) graph Anchor LDA with similar substructure merging, (b) structural topic aware multi-view GCN.

preprocessing to obtain the features of each device. In this part, we use the GraphSTSGM algorithm to preprocess to obtain the features of the device by combining the original features of the device and the approximate local symmetry features of the neighborhood structure. In the third part, the features obtained by the preprocessing are used as the input of the DBSCAN for malicious device detection.

Data preprocessing is the basis for improving the performance of the entire malicious device detection, because the device features obtained by data preprocessing directly affect the final performance of the malicious device detection. Existing data preprocessing for malicious device detection usually uses manually pre-defined random walk paths to obtain substructures [20]. Existing algorithms cannot directly obtain abnormal substructures, resulting in low AUC.

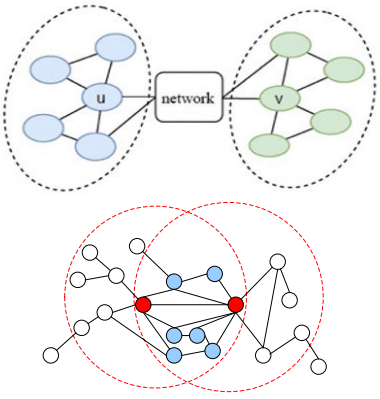


FIGURE 5. Graph of structural equivalence and homophily equivalence. (a) Node graph with structural equivalence neighborhood structure. (b) Node relation graph with homophily equivalence neighborhood structure.

C. EXTRACT APPROXIMATELY LOCALLY SYMMETRIC FEATURES BY USING THE GraphSTSGM ALGORITHM

1) PREPROCESSING OF THE GraphSTSGM ALGORITHM

Figure 6 is the process of generating a Unique Mobile Identity, abbreviated umid, for each traffic and building a device-account graph. First, we generate umid from network traffic following the algorithm in Reference 19. The calculation method of umid is detailed in Reference 19,

which will not be repeated in this article. Traffic flow refers to the information of the traffic generated by each operation of the device, including sip, account id, and information required by the device to generate umid [19]. Unique Mobile IDentity refers to establishing a globally unique device ID for each operating device to uniquely represent the device [19]. After calculating the umid using the algorithm of Reference 19, if the information of this device has appeared before, the previous umid will be returned. If the information for this device has never appeared, the newly generated umid is returned. We perform the above processing on each device traffic to obtain device traffic with umid. After that, a device-account bipartite graph G(V, E) is constructed with devices and accounts as nodes.

2) OVERVIEW OF GraphSTSGM

First, a device-account bipartite graph G(V, E) is constructed with devices and accounts as nodes. The set of nodes is $V = (\{\text{Vumid1}, \ldots, \text{Vumidi}\}, \{\text{Vaccountid1}, \ldots, \text{Vaccountidi}\})$. Edge is connected between the associated device and accountid, and the set of edges is $\mathrm{E} = \left[e_{ij}\right]^n$. Then, perform an anonymous random walk of length l for each $v_i \in V$ in G to obtain W_l. Finally, the sequence set $\boldsymbol{M} \in \mathbb{R}^{m*|w_l|}$ of the original random walk and the corresponding anonymous random walk sequence set $\bar{M} \in \mathbb{R}^{m*|w_l|}$ are obtained.

Each anonymous walk sequence represents a substructure, so we can get the set $sub - G \in \mathbb{R}^{m*|sub-G|}$ of the node-to-node relationships of m substructures corresponding to the random walk sequence set M. Therefore, the AW-SGM algorithm is used to merge the substructures in $\bar{M}$ whose similarity is greater than the threshold η to obtain a set $sub - G' \in \mathbb{R}^{n*|sub-G|}$ containing n substructures and the corresponding anonymous walk sequence $\widehat{M} \in \mathbb{R}^{n*|w_l|}$. The resulting $\widehat{M}$ is the set of substructures of various types that characterize the anonymous walk representation. This step will be detailed in Section 3.3.2.

Then, we formulate topic modeling on graphs. We select the k substructures with the most occurrences as the structural topics of the graph. The top-k substructures that are selected to be the node neighborhood graph are also called "anchors." A topic model on graphs aims to learn the following parameters.

(1)A node-topic matrix $R \in \mathbb{R}^{|V|\times K}$, where a row R_i corresponds to a distribution with R_{ik} denoting the probability of node v_i belonging to the $k-$th structural topic [6].

(2)A walk-topic matrix $U \in \mathbb{R}^{K\times|\mathcal{W}_l|}$ where a row U_k is a distribution over W_l and U_{kw} enotes the probability of $w \in W_l$ belonging to the k-th structural topic [6].

In addition, we define the set of anonymous walks starting from v_i as D_i, with $D_i = N$ as the number of walks to sample. we define the walk-walk co-occurrence matrix [6] $\widehat{M} \in \mathbb{R}^{|W_l|\times|W_l|}$,with $M_{i,j} = \sum_{v_k\in V} \mathbb{I}\left(w_i \in D_k, w_j \in D_k\right)$, and adopt non-negative matrix factorization (NMF) [23] to extract anchors. In detail, $\widehat{M}$ is the walk-walk co-occurrence matrix of anonymous walk sequences after merging similar substructures.

$$H, Z = \operatorname{argmin} \|\widehat{M} - HZ\|_F^2, \quad s.t.\ H, Z^T \in \mathbb{R}^{|\mathcal{W}_l|\times\alpha}, H, Z \geq 0, \tag{1}$$

Finally, we get anchor $\mathrm{A}_k = \arg\max\left(Z_k\right), k = 1, \ldots \alpha$. In detail, A is the matrix of anchor and Z_k is the first k rows of matrix Z. The anchor node obtained at this time is the potential topic. Based on the anchors we found, we proceeded to learn the walk-topic distribution U. We get $U \in \mathbb{R}^{K\times|\mathcal{W}_l|}$ through optimizing.

$$\arg\min_U D_{KL}\left(Q_i \| \sum_{k\in \mathrm{A}} U_{ik}\,\mathrm{diag}^{-1}(\overrightarrow{Q_1})Q_{\mathrm{A}_k}\right), \tag{2}$$

where Q is the re-arranged walk co-occurrence matrix with anchors A lying in the first α rows and columns, and Q_{Ak} is the row of Q for the k-th anchor [6].

In addition, we define the node-walk matrix $Y \in \mathbb{R}^{|V|\times|W_l|}$ with Y_{iw} denoting the occurrences of w in D_i. We then get the node-topic distribution R through $R = YU\dagger$, where $U\dagger$ denotes pseudo-inverse [6].

Structural-topic Aware Aggregator. We use neighborhood aggregation to illustrate structural equivalence, which means that nodes with similar neighborhood connections have similar effects on surrounding neighborhoods;

$$h_i^{(k)} = AGGREGATE\left(\left\{\frac{R_i^T R_j}{\sum_j R_i^T R_j} h_j^{(k-1)}, v_j \in N\left(v_i\right)\right\}\right), \tag{3}$$

In detail, where $h_i^{(k)}$ denotes the output vector for node v_i from the k-th layer. We aggregate neighborhood information with Graph Sample and aggregate (GraphSAGE)[24] for added flexibility.

We use two parallel Graph Convolutional Networks (GCNs), one focusing on structural topics and the other on node features correspondingly [6]. Where $h_{i,n}^{(k)}$ and $h_{i,s}^{(k)}$, $k = 1, \ldots, L$ denote the outputs of the two GCNs for node v_i at layer k. After that we use a nonlinear neural network layer on the two output vectors. As shown in Figure 4(b).

$$h_i^{(L)} = \left(\mathbf{W} \cdot \tanh\left(\left[h_{i,n}^{(L)} \otimes h_{i,s}^{(L)}\right]\right) + \mathbf{b}\right), \tag{4}$$

In detail, $h_i^{(L)}$ is the final output of muti-view GCN to node v_i, $\otimes$ is the concatenation operation.

The initialization $h_i^{(0)}$ consists of two parts, one is $h_i^{(0)}, n = X_i$ of node v_i, and the other is node $h_{i,s}^{(0)}$ This part is obtained by splicing the distribution of node v_i at anchor node A and the node-topic distribution R_i.

$$\mathcal{L} = -\log\left[\sigma\left(h_i^{(L)T} h_j^{(L)}\right)\right] - q \cdot \mathbb{E}_{v_n\sim P_n(v)} \log\left[\sigma\left(h_i^{(L)T} h_n^{(L)}\right)\right], \tag{5}$$

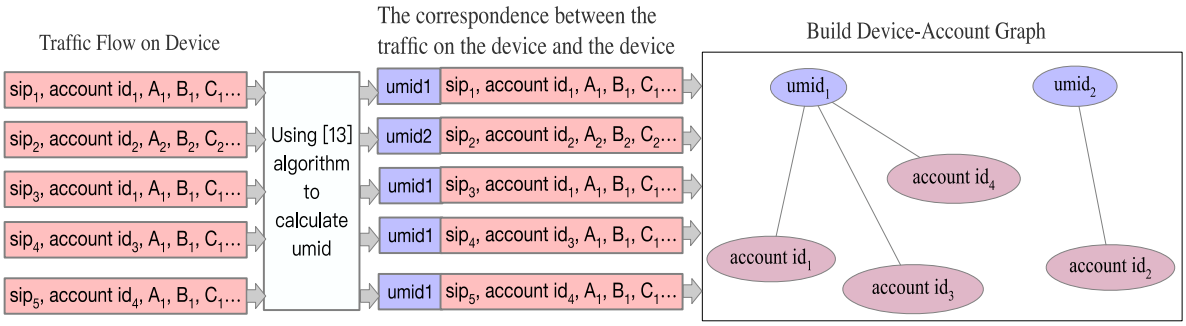


FIGURE 6. Preprocessing for building a device-account graph. This is the process of generating umid for each device traffic and building a device-account graph. First, we generate umid for each device traffic according to the algorithm in [19]. After that, build a device-account graph according to the relationship between accounts and umids in device traffic. It is worth noting that in the process of constructing the device-account graph, no duplicate edges are constructed between umid and account. That is to say, if there is multiple traffic between umidi and account idi, only one edge is retained in the device-account graph.

Algorithm 1 Algorithm of GraphSTSGM-DBSCAN

Require: network traffic flow f_i, f contains the traffic of q devices, each of which has a device;
Ensure: the type to which the q device belongs;

Step 1: Build a multidimensional relationship graph between devices
A device-account relationship graph is constructed with the device and account id as nodes, and an edge is constructed between the associated device and account id to obtain graph G;
Step 2: Extract topological relation representations using GraphSTSGM
$M \leftarrow Walkco - occourences(G)$ $(M = M_1, M_2, \ldots, M_m)$;
Form $\bar{M} = \{\bar{M}_1, \bar{M}_2, \ldots, \bar{M}_m\}$, Anonymous walk of M;
$\widehat{M} \leftarrow \text{AW} - \text{SGM}(\bar{M}_i, \bar{M}_j)$, $\bar{M}_i$ and $\bar{M}_j$ belongs to $\bar{M}$, $\widehat{M} = \{\widehat{M}_1, \widehat{M}_2, \ldots, \widehat{M}_n\}$. (Eq. 9, 10 merging similar graphs);
$A \leftarrow \text{LDA}(\widehat{M}, \text{K})$;
$U, R \leftarrow RecoverLatentTopic(M, A)(Eq.2)$
$\Phi \leftarrow$ Structure-topic aware GCN(G, U, R); (Eq. 3, 4, and 5)
Get a vector representation of the device's neighborhood structure Vec consists of q devices of network traffic, each device vector $(vec_{i1}, vec_{i2},,, vec_{ir})$;
Step 3:Clustering to get topologically similar sets of nodes
Use DBSCAN clustering to obtain a set of topologically similar nodes to obtain a cluster of q devices after clustering;
Cluster(q) $\leftarrow DBSCAN(Vec(q))$;
return Cluster(q) after clustering of q devices;

where $\sigma(x)$ is the sigmoid function, v_j co-occurs with v_i in random walks, and Pn(v) is the noise distribution for negative sampling.

Algorithm 1 is the pseudocode of the GraphSTSGM algorithm. Figure 7 shows the algorithm flow of GraphSTSGM based on graph representation with similar substructure aggregation.

3) AW-SGM: ANONYMOUS WALK BASED ON SIMILAR SUBSTRUCTURE MERGING

The existing method uses the device-account graph obtained by anonymous walk to represent the neighborhood structure of the device M as the input of the Graph Anchor Latent Dirichlet Allocation (LDA) [25] to select the topic substructure. Anonymous random walk is a method to obtain the representation of the neighborhood structure, which has the ability to obtain all the neighborhood structure representations in the graph. The neighborhood structure representations of nodes with similar functions are similar. Therefore, we hope that similar structures in the process of preprocessing to get the neighborhood structure representation M are grouped into same category, as shown in Figure 3(a). However, existing graph representation learning algorithms do not have the ability to identify whether substructures are similar, which leads to the use of Graph Anchor LDA for topic substructure selection to obtain anchor A, some substructures are not selected into anchor set A. These substructures appear infrequently, but are similar to frequently occurring substructures. When the substructure reconstructs the node neighborhood structure, the nodes whose neighborhood is reconstructed by the substructure with few occurrences will be mistakenly regarded as abnormal nodes. Therefore, existing algorithms using anchor set A have no ability to accurately detect nodes with abnormal

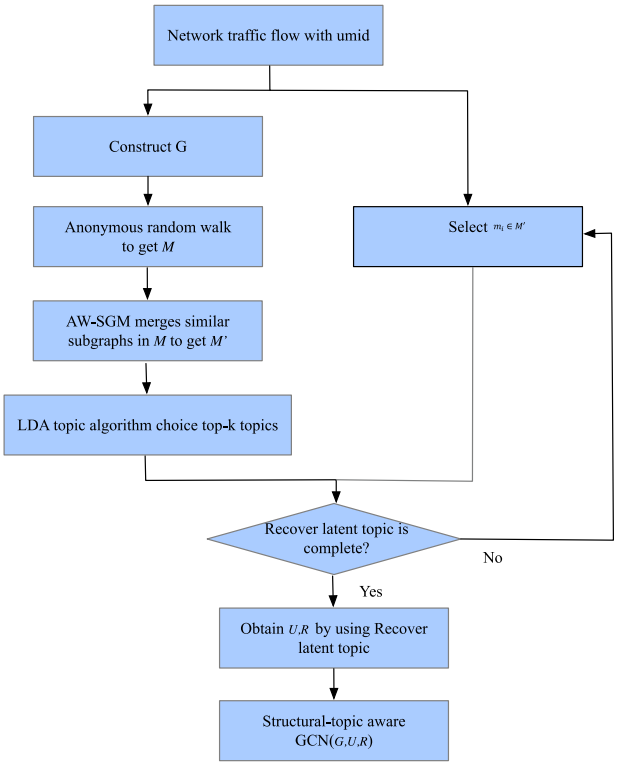


FIGURE 7. GraphSTSGM algorithm flow chart based on graph representation with similar substructure aggregation.

substructures. In detail, the neighborhood structure representation M obtained by preprocessing is the basis for obtaining accurate substructure nodes with abnormality. However, the neighborhood structure representation M obtained by existing algorithms has no ability to merge similar substructures. In order to merge similar substructures through the neighborhood structure representation M, we introduce an anonymous walk algorithm based on A^* SGM similar substructure merging, which is called AW-SGM. This algorithm is equivalent to merging anonymous walk sequences with high similarity in M. Anonymous walk sequences are merged to obtain n sets of similar substructures. An anchor node is selected from each set of the n substructure sets and is stored in $\bar{M} \in \mathbb{R}^{n*|w_l|}$ on behalf of the set. In detail, the substructures in the same set represent the same type. Then the resulting $\bar{M}$ is the set of all the different substructure types that characterize the anonymous walk representation. That is to say, we use the AW-SGM algorithm to combine the substructures of the same type in the substructure set into one representation. Two basic steps of the AW-SGM learning algorithm are as follows.

First, the A^* SGM based on anonymous walk(AW-SGM) is used to merge similar substructure s to obtain the similar substructure set $S = \{s_1, \ldots, s_n\}$. The specific process is as follows. (1) The substructure obtained by anonymous walk is $\bar{M} = \{\bar{M}_1, \bar{M}_2, \ldots, \bar{M}_m\}$. Store each substructure in $\bar{M}$ into a similar substructure set to get $S = \{s_1, \ldots, s_m\}$, that is, the only element in the similar substructure set s_i is the substructure $\bar{M}_i$. Then, the similar substructure merging algorithm Merge(s) based on A^* is used to merge the substructure s; (2) In the process of merging similar substructure s in each step, the A^*-based similar substructure merging algorithm Merge(s) is used to find substructure s in $s_i \in S$, i.e. satisfy (current merged gain - current merged loss)$\leq\eta$ as candidate substructure, that is, to find the substructure of $\boldsymbol{h}(s_i, s_j) \leq \eta$; The algorithm flow is detailed in Section 3.3.3. (3) We select a substructure set to be merged in the next step from the candidate substructure, which is equivalent to simulating the Alias sampling with time complexity O(1) according to the $\boldsymbol{h}(s_i, s_j)$. We merge the substructure s_j selected in the candidate substructure with the current substructure s_i to obtain the merged substructure set $s_i = \{s_i \cup s_j\}$; (4) The above steps (2) and (3) are repeated until no substructure s can be merged; Finally, we get the set of similar substructure $S = \{s_1, \ldots, s_n\}$. In detail, s_i contains one or more substructure s, and the similarity between multiple substructure s in each s_i is high.

We select the central node $\bar{M}_i$ representing the similar substructure set $s_i \in S$ from each similar substructure set s_i to obtain $\widehat{M} = \{\widehat{M_1}, \widehat{M_2}, \ldots, \widehat{M_n}\}$. The specific process is as follows. (1) We calculate the sum of edit distances [26] d_{now} of each substructure $\bar{M}_{now} \in s_i$ from other substructure s in the set in $s_i \in S$ the calculation method is as shown in (6); (2) We repeat the above step (1). We respectively calculate the sum of edit distances [26] between substructure s in s_i and other substructure s, and save them to the set $d_{si} = d_{now1}, \ldots, d_{nowp}$. The center $\bar{M}_{center}$ of the similar substructure set s_i is the substructure corresponding to the node with the smallest graph edit distance from other substructure s in d_{si}, the calculation method is as shown in (7); (3) We repeat the above steps (1) (2), and calculate the center $\bar{M}_{center}$ of each $s_i \in S$ in $S = \{s_1, \ldots, s_m\}$, and save the center $\bar{M}_{center}$ corresponding to each s_i into $\widehat{M}$.

$$d_{\text{now}} = \sum_{i=0}^{\bar{M}_i \in s_i} \boldsymbol{D}\left|\bar{M}_i, \bar{M}_{\text{now}}\right|, \tag{6}$$

$$\bar{M}_{\text{center}}\, ofs_i = \min\left(\mathrm{d}_{si}\right), \tag{7}$$

It is worth noting that the central substructure of each similar substructure set s_i is considered to be the graph representing the set of substructure s.

4) A^*: PENALTY-BASED SIMILAR SUBSTRUCTURE SET MERGING ALGORITHM

Existing graph representation learning algorithms do not have the ability to combine similar substructures set, resulting in some substructures that are not selected into anchor set A when using Graph Anchor LDA to select topic substructures to obtain anchor A. Therefore, we propose a similar substructure merging algorithm of A^* to merge sets of substructures. The specific process of the A^*'s similar substructure set merging method is as follows. First, we use

formula (9), as shown at the bottom of the next page, to calculate the gain $\boldsymbol{h}(s_i, s_j)$ for merging between similar substructure sets s_i and s_j. Then, formula (10), as shown at the bottom of the next page, calculates the loss $\boldsymbol{g}(s_i, s_j)$ of merging between sets of similar substructure s. Finally, when formula (11) is satisfied, the substructure sets s_i and s_j can be merged. Several key operations involved in the A^*'s similar substructure set merging methods are described below.

Search Bias $\boldsymbol{h}(s_i, s_j)$*:* $\boldsymbol{h}(s_i, s_j)$ is used to compute the similarity between sets of substructures s. At the same time, $\boldsymbol{h}(s_i, s_j)$ is considered to be the yield of the merge between similar substructure sets s_i and s_j. In detail, $\boldsymbol{h}(s_i, s_j)$ is equivalent to using dynamic programming to calculate the similarity between substructure sets s_i and s_j. When $\bar{M}_m = s_i$ and $\bar{M}_n = s_j$, it is equivalent to calculating the similarity between substructure $\bar{M}_m$ and substructure $\bar{M}_n$. Essentially, it calculates the sum of the number of nodes and edges in substructure $\bar{M}_m$ and the number of nodes and edges in substructure $\bar{M}_n$, and then subtracts the graph edit distance [26] $\boldsymbol{D}\left|\bar{M}_m, \bar{M}_n\right|$ between $\bar{M}_m$ and $\bar{M}_n$; $\boldsymbol{h}(s_i, \bar{M}_n)$ refers to calculating the similarity between the similar substructure set s_i and the substructure $\bar{M}_n$, that is, calculating the sum of the similarity between each substructure in the similar substructure set s_i and $\bar{M}_n$. Similarly, $\boldsymbol{h}(\bar{M}_m, s_j)$ refers to calculating the similarity between the substructure $\bar{M}_m$ and the set of similar substructure s_j. $\boldsymbol{h}((s_i - \bar{M}_m), s_j)$ refers to calculating the similarity between the set of substructure s after removing $\bar{M}_m$ in s_i and s_j. $\boldsymbol{h}(s_i, (s_j - \bar{M}_n))$ refers to the similarity between the set of substructure s formed by removing $\bar{M}_n$ from s_j and s_i. In detail, the more similar s_i and s_j are, the higher the value of $\boldsymbol{h}(s_i, s_j)$.

The calculation method of search bias $\boldsymbol{h}(s_i, s_j)$ is shown in (9), where num $(\bar{M}_m)$ refers to the sum of the number of nodes and edges in the substructure $\bar{M}_m$. $\boldsymbol{D}\left|\bar{M}_m, \bar{M}_n\right|$ refers to the graph edit distance between $\bar{M}_m$ and $\bar{M}_n$.

Penalty $\boldsymbol{g}(s_i, s_j)$*:* $\boldsymbol{g}(s_i, s_j)$ is the penalty term for merging between similar substructure sets, also known as the loss of merging between similar substructure sets.

The calculation method of penalty $\boldsymbol{g}(s_i, s_j)$ is shown in (10), where n is the number of substructure s involved, and w_i is the total number of substructure structures in the i^{th} substructure;

Penalty-Based Weight $\boldsymbol{n}(s_i, s_j)$*):* $\boldsymbol{n}(s_i, s_j)$ is a penalty-based weight, also known as the probability of merging sets of similar substructure s. The penalty based weight $\boldsymbol{n}(s_i, s_j)$ is calculated in (8), as shown at the bottom of the next page. The penalty-based weight $\boldsymbol{n}(s_i, s_j)$ is the penalty $\boldsymbol{g}(s_i, s_j)$ minus the biased weight $\boldsymbol{h}(s_i, s_j)$. The substructure sets s_i and s_j can be merged when (current merged gain - current merged loss) is greater than a threshold η. That is, when formula (11) is satisfied, the substructure sets s_i and s_j can be merged; If the threshold $\eta = \max(\text{num}(\bar{M}_i))$, and $\bar{M}_i \epsilon \bar{M}$, the GraphSTSGM algorithm is the GraphSTONE [6] algorithm; That is to say, when the selected threshold is the largest number of nodes and edges of the substructure in $\bar{M}$, no substructure can be merged in the graph. At this time, the GraphSTSGM algorithm is the GraphSTONE [6] algorithm.

TABLE 1. Dataset statistics and properties.

Dataset	Type	|V|	|E|
Cora	Citation	2708	5429
Alibaba Cloud's data	Risk detection	36725	286591

D. DBSCAN FOR MALICIOUS DEVICE DETECTION

We use the vector obtained by GraphSTSGM as the input of DBSCAN to get the clustering result. We consider that there are malicious devices in a cluster, then other devices in the cluster are also malicious devices. At the same time, the abnormal device in the reconstruction of the neighborhood structure is also determined as a malicious device. In detail, the abnormal devices during neighborhood structure reconstruction are outliers after DBSCAN clustering.

IV. EXPERIMENTS

In this section, we perform experiments to evaluate the model GraphSTSGM on Alibaba Cloud data and the public dataset Cora. First, we introduce the experimental setup. Specifically, our experiments include:

(1) Comparing the model effect when preprocessing with GraphSTSGM and GraphSTONE;

(2) Parameter evaluation, including model parameter analysis and so on;

(3) We verified the effect of the model on Alibaba Cloud data, and proved the generality of the algorithm on public dataset Cora;

A. DATASET

The proposed algorithm and existing algorithms are compared in the following two datasets.

1) CORA DATASET

The dataset [27] has a total of 2708 samples, each sample is a scientific paper, and all samples are divided into 8 types. Each paper cites at least one other paper, or is cited by other papers. That is, there are connections between the samples and no isolated nodes. If we treat the samples as nodes in the graph, then this is a connected graph and there are no isolated nodes.

2) ALIBABA CLOUD DATASET

Alibaba Cloud dataset is the correlation data between devices and accounts, and it is used to detect malicious devices.

B. EVALUATION METRICS

We use the widely adopted metric to measure the performance of malicious device detection, namely AUC [28] and Recall. The formulas are as follows. The meanings of True Positive (TP), False Positive (FP), False Negative (FN), and True Negative (TN) [22] are shown in Table 4.

Algorithm 2 AW-SGM Learning Algorithm

Require: $\bar{M} = \{\bar{M}_1, \bar{M}_2, \ldots, \bar{M}_m\}$, the normalized rows of M, the original structure G of each substructure in M, and the nodes V and edges E in G, Threshold η;

Ensure: De-duplicate the substructure s with high similarity in the substructure set in $\bar{M}$ to get $\bar{M}' = \{\bar{M}_1, \bar{M}_2, \ldots, \bar{M}_m\}$; Similar substructure set $S = \{s_1, \ldots, s_n\}$. $\bar{s_1} = \{\bar{M}_1, \bar{M}_2, \ldots, \bar{M}_t\}$,...,$\bar{s_n} = \{\bar{M}_i, \ldots, \bar{M}_r\}$, in detail $\bar{M} = s_1 \cup \ldots \cup s_n$;

Initialize s =Deepcopy($\bar{M}$);
for each graph $\overline{m_i} \in \bar{M}$ **do**
  is_graph_merge = $\boldsymbol{h}(s_i, s_j) - \boldsymbol{g}(s_i, s_j)$
  if is_graph_merge $\leq \eta$ **then**
    $s_i = \{s_i \cup s_j\}$;
  end if
end for
Obtain Similar substructure set $S = \{s_1, \ldots, s_n\}$.
Initialize M' to Empty;
for each $s_i \in s$ **do**
  for each $\bar{M}_{\text{now\_}} \in s_i$ **do**
    $\mathrm{d}_{\text{now}} = \sum_{i=0}^{\bar{M}i \in s_i} \boldsymbol{D}\left|\bar{M}_i, \bar{M}_{\text{now}}\right|$
  end for
  Obtain s_i in $d_{\text{si}} = \{d_{\text{now1}}, \ldots, d_{\text{nownn}}\}$
  $\bar{M}_{\text{center}}$ of $s_i = \min(d_{\text{si}})$
  Append $\bar{M}_{\text{center}}$ of s_i into M'

end for
return M',s;

TABLE 2. Confusion matrix.

	Actual Class: True	Actual Class: False
Predicted Class: True	TP	FP
Predicted Class: False	FN	TN

$$\text{DetectionRate}(DR) = \frac{TP}{TP + FN},$$
$$\text{Precision}(P) = \frac{TP}{TP + FP}, \tag{12}$$

C. TWO WAYS TO PREPROCESS MALICIOUS DEVICE DETECTION

There are two ways to preprocess network packets. In method (a), we extract approximate local symmetry features of devices using the GraphSTSGM algorithm, which adds a penalty-based similar substructure set merging algorithm for preprocessing devices. In method (b), the local neighborhood structural features of the device are extracted using the GraphSTSGM algorithm without similar substructure set merging. The approximate local symmetry feature of the device can characterize the neighborhood topology. This feature can also characterize whether there is an abnormal structure or similarity to a known malicious device. In detail, extracting local neighborhood structure features of devices without similar substructure merging, i.e., the threshold $\eta = \max(\text{num}(\bar{M}_i))$, and $\bar{M}_i \epsilon \bar{M}$, in (11), as shown at the bottom of the page, is equivalent to extracting neighborhood structure features using GraphSTONE [6]. From Table 3, when the length of anonymous walk increases, the AUC of both methods (a) and (b) decreases, but the AUC of method (a) is still as high as 85.9%. From Table 3, we can conclude that the AUC of method (a) is 2.7% higher than the method (b). A possible explanation is that the lack of merging of similar substructures when extracting topic

$$\boldsymbol{n}(s_i, s_j) = \boldsymbol{h}(s_i, s_j) - \boldsymbol{g}(s_i, s_j), \tag{8}$$

$$\boldsymbol{h}(s_i, s_j) = \begin{cases} \boldsymbol{h}\left(\left(s_i - \bar{M}_m\right), s_j\right) + \boldsymbol{h}\left(\bar{M}_m, s_j\right), & \bar{M}_m \in s_i \\ \boldsymbol{h}\left(s_i, \left(s_j - \bar{M}_n\right)\right) + \boldsymbol{h}\left(s_i, \bar{M}_n\right), & \bar{M}_n \in s_j \\ \frac{\text{num}(\bar{M}_m) + \text{num}(\bar{M}_n) - 2 * \boldsymbol{D}\left|\bar{M}_m, \bar{M}_n\right|}{2}, & \text{if } \bar{M}_m = s_i \text{ and } \bar{M}_n = s_j \end{cases} \tag{9}$$

$$\boldsymbol{g}(s_i, s_j) = \frac{\sum_{i=0}^{p} \sum_{j=0}^{q} w_i * w_j * D\left|\bar{M}_i, \bar{M}_j\right|}{\sum_{i=0}^{p} w_i + \sum_{j=0}^{m} w_j}, \tag{10}$$

$$\boldsymbol{h}(s_i, s_j) - \boldsymbol{g}(s_i, s_j) >= \eta, \tag{11}$$

TABLE 3. Compares the performance of GraphSTSGM and GraphSTONE preprocessed under different random walk lengths in malicious device detection (%).

length	Preprocessing Algorithm			
	(a) Preprocessed with GraphSTSGM		(b) Preprocessed with GraphSTONE	
	AUC	*Recall*	AUC	*Recall*
10	**89.2**	**87.1**	**86.5**	**86.6**
20	86.7	80.9	80.5	76.3
80	85.9	60.3	79.1	58.5

TABLE 4. Compare the impact of different threshold values on the performance of GraphSTSGM(%).

Threshold	AUC	Recall
100	86.5	86.6
50	87.9	83.7
20	**89.2**	**87.1**
0	85.3	83.9
-10	63.2	50.9

substructures in method (b) will lead to random walks to obtain substructures that appear infrequently and are similar to frequently occurring substructures. This method is easy to cause the nodes with normal neighborhood structure to be mistakenly regarded as abnormal nodes. Eventually, the AUC of the method (b) is not high, and the approximate local symmetry features obtained by preprocessing with this method cannot accurately describe the neighborhood topology of the device. Therefore, in the following we use method (a) for data preprocessing.

D. THE EFFECTIVESS OF SIMILAR SUBSTRUCTURE MERGING IN GraphSTSGM PREPROCESSING BY DBSCAN

This section discusses the impact of similar substructure merging threshold values on the performance of malicious device detection in GraphSTSGM. Different thresholds for merging similar substructures set will affect the selection of topic substructures, which in turn affects the accuracy of describing the structure embedding of each device's neighborhood. Therefore, it is necessary to evaluate the impact of different thresholds of similar substructure merging on the performance of GraphSTSGM-DBSCAN. We selected the thresholds for merging similar substructures to be 100, 50, 20, 0, and -10 for evaluation. Table 4 shows the effect of different thresholds on the performance of the GraphSTSGM algorithm. It can be concluded from Table 4 that the GraphSTSGM-DBSCAN preprocessing algorithm achieves optimal performance when the threshold for merging similar substructures is 20. When the threshold is -10, the performance of malicious device detection is low. A possible explanation is that the threshold for merging similar substructures determines the substructures included in the topic substructure, which in turn affects the composition of the device neighborhood structure reconstruction. This threshold cannot obtain an approximate local symmetry feature that accurately characterizes the neighborhood structure of nodes. When the threshold is 100, the threshold selection in formula (11) is too high, so that similar substructures are hardly merged. This threshold cannot obtain an approximate local symmetry feature that accurately characterizes the neighborhood structure of nodes. Therefore, the effect of the algorithm is similar to that of GraphSTONE. One possible explanation is that GraphSTONE has the following two shortcomings. The first shortcoming is that an infrequent substructure in the set of substructures obtained by an anonymous random walk is similar to a frequently occurring substructure. When using the substructure to reconstruct the neighborhood structure of a node, the node is mistakenly regarded as an abnormal node. The second shortcoming is that this method can only find nodes with the same neighborhood structure, but cannot find nodes with similar neighborhood structures. Therefore, potential malicious nodes with similar neighborhood structures to the determined malicious nodes are not divided into the same cluster, which leads to a low AUC of malicious device detection; When the threshold is -10, the threshold selection in Eq. (11) is too low to cause dissimilar substructures to be merged into a set of substructure types. As a result, when the device neighborhood structure is reconstructed, devices with dissimilar neighborhood structures will be misclassified into one class. Therefore, the obtained local neighborhood structure vector does not have the ability to accurately describe the similarity of the device neighborhood topology;

Different thresholds for merging similar substructures set affect the performance of GraphSTSGM-DBSCAN malicious device detection. Therefore, other parameters also affect the detection performance. Table 5 shows a list of parameters for AW-SGM in GraphSTSGM.

TABLE 5. List of parameters for AW-SGM in GraphSTSGM.

Hyper-Parameters	Value
threshold	20
path length	10
window size	6
learning rate	0.002
num neighbor	20
hidden dim	100
embedding dims	64

E. COMPARISON WITH THE LATEST TECHNIQUES

Researchers have proposed graph-based or cluster-based detection algorithms, such as: Graph Neural Networks with Adaptive Receptive Paths (Geneipath) [28], Label Propagation Algorithm (LPA) [29], DBSCAN [30], Struc2vec [17], Graph Attention Networks (GATs) [31], GraphSTONE [6], etc. The experiment compares the performance of the proposed algorithm with the existing algorithms. Table 6 and Table 7 compare the effects of the latest existing methods and current algorithms when using Alibaba Cloud data and Cora dataset, respectively. It can be seen from Table 6, the AUC of the proposed GraphSTSGM-DBSCAN algorithm is 3.6% higher than that of other existing state-of-the-art algorithms. We consider that the GraphSTONE-DBSCAN algorithm is a GraphSTSGM-DBSCAN without similar substructure merging.GraphSTONE-DBSCAN can extract complete local symmetry features, but cannot extract approximate local symmetry features. Therefore, GraphSTONE-DBSCAN is

TABLE 6. Effects on Alibaba cloud data (%).

Algorithm	AUC	Recall
Struc2vec[17]	53.6	43.2
DBSCAN	52.9	50.3
GAT[31]	79.6	60.3
GraphSage[32]	84.9	85.9
GraphSTONE[6]	85.6	85.2
GraphSTONE-DBSCAN(proposal)	**86.5**	**86.6**
GraphSTSGM-DBSCAN(proposal)	**89.2**	**87.1**

TABLE 7. Effects on Cora dataset (%).

Algorithm	AUC	Recall
Struc2vec[17]	54.2(No Feature)	54.3
DBSCAN	52.6	51.9
GAT[31]	94.6	94.7
GraphSage[32]	95.3	95.4
GraphSTONE[6]	96.3	**96.7**
GraphSTONE-DBSCAN(proposal)	**97.6**	95.9
GraphSTSGM-DBSCAN(proposal)	**98.9**	**97.8**

considered to be the worst case for the GraphSTSGM-DBSCAN algorithm. In the worst case, the proposed algorithm achieves the good performances regarding the AUC exceeding those of the other state-of-the-art algorithms by 0.9%.

It can be seen from Table 6 and Table 7 that the AUC obtained by the GraphSTONE algorithm preprocessing is higher than the AUC obtained by the Struc2vec [17], GAT [31] and other algorithms. Therefore, we add similar substructure merging to the GraphSTONE algorithm, namely the GraphSTSGM algorithm, to obtain better preprocessing performance. GraphSTSGM can extract approximate local symmetry features. The AUC of GraphSTSGM-DBSCAN algorithm is higher than DBSCAN, so we conclude that preprocessing with GraphSTSGM algorithm has better malicious device detection performance than no preprocessing. As can be seen from Table 6 and Table 7, the Struc2vec algorithm has the lowest AUC and recall on Alibaba Cloud data and Cora dataset. There is little difference between GraphSage and GraphSTONE in Alibaba Cloud data and Cora dataset. The performance of the GraphSTSGM algorithm is the best. A possible explanation is that the Struc2vec algorithm only considers the topological relationship of the node neighborhood and does not consider the original information of the node, thus resulting in poor performance. The Struc2vec algorithm judges whether two nodes have similar structures through the same degree of node neighborhood structure. This method can roughly judge the similarity of the neighborhood structure of nodes, and this method describes the topological relationship of node neighborhoods more roughly than GraphSTSGM. Struc2vec ignores neighborhood topological similarity between devices when preprocessing device extraction features. The algorithm ignores the analyzed information resulting in low AUC. The GAT and GraphSage algorithms consider both the node neighborhood topological relationship and the original information of the node, so the performance is better than the Struc2vec algorithm. GraphSTONE can only extract completely local symmetric features, but cannot extract approximate local symmetric features, so the performance is lower than the GraphSTSGM algorithm. The AUC of the GraphSTSGM-DBSCAN algorithm in Table 6 and Table 7 are all better than other existing state-of-the-art algorithms. Therefore, it shows that the algorithm is superior to the existing state-of-the-art algorithms in general graph scenarios.

For training and testing time, all of my experiments were able to run in 1h. On the Cora data, we use the GraphSTSGM algorithm to preprocess and obtain the local neighborhood structure features of the device, which can be completed within 0.5h. After that, it takes 0.5h to use the obtained local neighborhood structure features as the input of DBSCAN for malicious device detection. In detail, our GraphSTSGM algorithm uses GraphLearn for graph distributed computing. Therefore, we could not find enough literature on training, testing time, and memory size, and we were not able to evaluate it.

V. DISCUSSION

In this paper, the GraphSTSGM learning algorithm is used to obtain the approximate local symmetry features that accurately describe the structural similarity relationship of devices. Next, the features extracted by the GraphSTSGM learning algorithm as input of the DBSCAN for malicious device detection. This article focuses on the impact of structural similarity relationships between devices and accounts on malicious device detection performance. Future work will further explore the features of device obtained by preprocessing. Device will be analyzed from multiple dimensions such as device-ip, device-androidid.

The proposed algorithm achieves better results than state-of-the-art algorithms on datasets such as Cora. We can conclude that the algorithm can be generalized in scenarios such as graph detection-based community discovery.

In this paper, the neighborhood structure of nodes is reconstructed by topic substructure for malicious device detection. This approach is highly interpretable.

In order to highlight the novelty and contribution of this paper, we compare the similarities and differences between our algorithm and existing algorithms in Table 8 and Table 9. Table 8 compares the differences between existing graph-based malicious device detection algorithms and our proposed algorithm. Our proposed algorithm is equivalent to enhancing the existing features of nodes by adopting the topological similarity of the node neighborhood structure. From Table 8, we can conclude that the performance of the existing Louvain and LPA algorithms only depends on the connection strength of nodes in the graph, and the performance of the Metapath algorithm depends on the features extracted by manually experience. Our proposed algorithm uses graph representation learning for automatic preprocessing. At the same time, our proposed algorithm can distinguish the neighborhood topology relationship of nodes in the graph,

TABLE 8. Comparison with other existing graph-based malicious device detection algorithms.

Algorithm	Raw Traffic Adoptation	Combine Approximate Local Symmetry Features in Preprocessing	Automatic Feature Extraction	Manual Experience to Extract Features	Automatic Malicious Device Detection
Louvain[14]	✓				
LPA[29]	✓				
DBSCAN[30]	✓			✓	
Geneipath[28]	✓		✓		
Metapath[20]				✓	
GraphSTONE-DBSCAN(proposal)	✓		✓		✓
GraphSTSGM-DBSCAN(proposal)	✓	✓	✓		✓

TABLE 9. Comparison with other published graph representation learning algorithms.

Algorithm	Local Symmetry Feature	Complete Local Symmetry Feature	Approximate Local Symmetry Feature	Obtain Substructure	Usability for Malicious Device Detection
Geneipath[28]					✓
Metapath[20]	✓				✓
GraphSTONE-DBSCAN(proposal)	✓	✓		✓	✓
GraphSTSGM-DBSCAN(proposal)	✓	✓	✓	✓	✓

which not only includes the connection strength information of the node, but also includes the neighborhood topology structure that can describe the function of the node. Therefore, our proposed algorithm is more suitable for malicious device detection. Table 9 compares the differences between existing graph representation learning algorithms and our proposed algorithm. Our proposed algorithm is capable of automatically extracting approximate local symmetry features that characterize the structural similarity of node neighborhoods.

In this paper, the graph is constructed and updated in days, and the algorithm will use near-line computing in the future.

Future research work mainly considers two aspects, including analysis from multi-dimensional graphs and near-line computation. Multi-dimensional graph analysis is to analyze the neighborhood structure of devices for more dimensions to detection malicious devices. The goal of near-line computing is to detect malicious devices in quasi-real time and improve computing and usage efficiency. We will continue to research the application of graph representation learning in the malicious device detection field with the hope of further improving the malicious device detection performance.

VI. CONCLUSION

In this paper, the GraphSTSGM algorithm is used to preprocess the device-account graph to obtain the approximate local symmetry features that describe the topological similarity of the device neighborhood. We use the features obtained by GraphSTSGM as the input of DBSCAN for malicious device detection. The experiment proves that our proposed algorithms achieve higher AUC than three of the state-of-the-art algorithms. In addition, it can be concluded from the experiment that the formula, i.e., the gain of combining substructure, $\boldsymbol{h}\left(s_i, s_j\right) - \boldsymbol{g}\left(s_i, s_j\right) >= \eta$, is designed to calculate the whether the two substructures should be combined. Experimental results show that merging similar substructures by this formula has the ability to obtain more accurate malicious device detection performance. In the best case, the AUC of GraphSTSGM-DBSCAN reached 89.2%. In the worst case, the AUC of GraphSTSGM-DBSCAN reached 86.5% Experimental results show that in the worst case, the AUC of the proposed algorithm is still higher than the existing state-of-the-art algorithms.

We verify the proposed algorithm on multiple datasets, and experiments show that the algorithm outperforms the existing state-of-the-art algorithms. The algorithm achieves an AUC of 89.2% on the Alibaba Cloud dataset. The algorithm achieves an AUC of 98.9% on the Cora[27] dataset.

The proposed algorithm can be applied in other contexts, such as citation networks, mining in academic social networks, and Protein-Protein Interaction (PPI) networks. Citation networks means that we can build a citation network based on the citation relationship between papers, and detect the same type of paper according to the citation relationship. This paper uses the citation network dataset Core for experiments. Experiments based on the graph dataset Cora show that the proposal outperforms the state-of-the-art algorithms by 2.6% with respect to the AUC of node classification. This shows that our proposed algorithm is general in other graph-based contexts. The proposed algorithm can be used in the mining of domain experts in academic social networks. We use paper information, paper citation, author information and author collaboration to build an academic social network, and mine experts of the same domain from the academic social network. Furthermore, the proposed algorithm can be used for the node classification task of Protein-Protein Interaction networks.

ACKNOWLEDGMENT

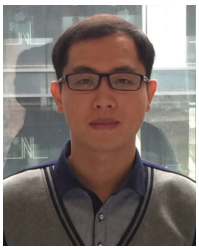
The authors would like to thank their family, friends, the editor, and anonymous reviewers for their insightful comments.

REFERENCES

[1] Q. Wu, X. Zhu, and B. Liu, "A survey of Android malware static detection technology based on machine learning," *Mobile Inf. Syst.*, vol. 2021, pp. 1–18, Mar. 2021.
[2] T.-M. Gronli, J. Hansen, G. Ghinea, and M. Younas, "Mobile application platform heterogeneity: Android vs Windows phone vs iOS vs Firefox OS," in *Proc. IEEE 28th Int. Conf. Adv. Inf. Netw. Appl.*, May 2014, pp. 635–641.
[3] Z. Liu, C. Chen, and X. Yang, "Heterogeneous graph neural networks for malicious account detection," in *Proc. 27th ACM Int. Conf. Inf. Knowl. Manage.*, Turin, Italy, 2020, pp. 2077–2085.
[4] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 855–864.
[5] D. Wang, Y. Qi, J. Lin, P. Cui, Q. Jia, Z. Wang, Y. Fang, Q. Yu, J. Zhou, and S. Yang, "A semi-supervised graph attentive network for financial fraud detection," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2019, pp. 598–607.
[6] Q. Long, Y. Jin, G. Song, Y. Li, and W. Lin, "Graph structural-topic neural network," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 23–27.
[7] S. K. S. Hussain, E. S. C. Reddy, K. G. Akshay, and T. Akanksha, "Fraud detection in credit card transactions using SVM and random forest algorithms," in *Proc. 5th Int. Conf. I-SMAC (IoT Social, Mobile, Anal. Cloud) (I-SMAC)*, Nov. 2021, pp. 1013–1017.
[8] D. Manjula, "A novel approach for behavior based charge card fraud detection using support vector machines," *Int. J. Sci. Res. Develop.*, vol. 3, no. 6, pp. 1094–1098, 2015.
[9] O. Raiter, "Applying supervised machine learning algorithms for fraud detection in anti-money laundering," *J. Modern Issues Bus. Res.*, vol. 1, no. 1, pp. 14–26, 2021.
[10] Q. Sun, T. Tang, H. Chai, J. Wu, and Y. Chen, "Boosting fraud detection in mobile payment with prior knowledge," *Appl. Sci.*, vol. 11, no. 10, p. 4347, May 2021.
[11] J.-W. Chang, N. Yen, and J. C. Hung, "Design of a NLP-empowered finance fraud awareness model: The anti-fraud chatbot for fraud detection and fraud classification as an instance," *J. Ambient Intell. Humanized Comput.*, pp. 1–17, Mar. 2022.
[12] W. Hilal, S. A. Gadsden, and J. Yawney, "Financial fraud: A review of anomaly detection techniques and recent advances," *Expert Syst. Appl.*, vol. 193, May 2022, Art. no. 116429.
[13] I. Benchaji, S. Douzi, and B. E. Ouahidi, "Credit card fraud detection model based on LSTM recurrent neural networks," *J. Adv. Inf. Technol.*, vol. 12, no. 2, pp. 113–118, 2021.
[14] N. Motschnig, A. Ramharter, O. Schweiger, P. Zabka, and K.-T. Foerster, "On comparing and enhancing common approaches to network community detection," 2021, *arXiv:2108.13482*.
[15] A. Santra, K. S. Komar, S. Bhowmick, and S. Chakravarthy, "A new community definition for multilayer networks and a novel approach for its efficient computation," 2020, *arXiv:2004.09625*.
[16] P. Supriya and V. N. Balasubramanian, "Community-based outlier detection for edge-attributed graphs," 2016, *arXiv:1612.09435*.
[17] L. F. R. Ribeiro, P. H. P. Saverese, and D. R. Figueiredo, "*struc2vec*: Learning node representations from structural identity," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 385–394.
[18] V. W. Zheng, S. Cavallari, H. Cai, K. C.-C. Chang, and E. Cambria, "From node embedding to community embedding," in *Proc. KDD WISDOM*, 2016, pp. 1–8.
[19] D. Lu, X. Li, and H. Wang, "Research and implementation of secure access method for IoT devices based on device fingerprints," (In Chinese), *Electron. Des. Eng.*, vol. 11, no. 2, pp. 136–140, Apr. 2019.
[20] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2017, pp. 135–144.
[21] B. Hu, Z. Zhang, C. Shi, J. Zhou, X. Li, and Y. Qi, "Cash-out user detection based on attributed heterogeneous information network with a hierarchical attention mechanism," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, pp. 946–953, Jul. 2019.
[22] Y. Hao, Y. Sheng, and J. Wang, "A graph representation learning algorithm for low-order proximity feature extraction to enhance unsupervised IDS preprocessing," *Appl. Sci.*, vol. 9, no. 20, p. 4473, Oct. 2019.
[23] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, Oct. 1999.
[24] L. H. William, Y. Rex, and L. Jure, "Inductive representation learning on large graphs," in *Proc. 31st Conf. Neural Inf. Process. Syst.*, Long Beach, CA, USA, 2017, pp. 1–11.
[25] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, Jan. 2003, pp. 993–1022.
[26] Z. Xu, "Graph edit distance overview," (In Chinese), *Comput. Sci.*, vol. 45, no. 4, pp. 11–18, 2018.
[27] (Jan. 19, 2022). *Download Dataset of Cora*. [Online]. Available: https://linqs-data.soe.ucsc.edu/public/lbc/cora.tgz
[28] Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, L. Song, and Y. Qi, "GeniePath: Graph neural networks with adaptive receptive paths," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, pp. 4424–4431, Jul. 2019.
[29] Z. Wang, F. Zou, B. Pei, W. He, L. Pan, Z. Mao, and L. Li, "Detecting malicious server based on server-to-server realation graph," in *Proc. IEEE 1st Int. Conf. Data Sci. Cyberspace (DSC)*, Jun. 2016, pp. 698–702.
[30] Y. He, H. Tan, W. Luo, S. Feng, and J. Fan, "MR-DBSCAN: A scalable MapReduce-based DBSCAN algorithm for heavily skewed data," *Frontiers Comput. Sci.*, vol. 8, no. 1, pp. 83–99, Feb. 2014.
[31] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. ICLR*, 2018, pp. 1–12.
[32] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1025–1035.
[33] (Apr. 11, 2022). *Extraction and Mining of Academic Social Networks*. [Online]. Available: https://www.aminer.cn/aminernetwork

YIRAN HAO received the Ph.D. degree from the Chinese Academy of Sciences, Beijing, China, in 2020, where she is currently pursuing the postdoctoral degree with the Institute of Automation. At the same time, she works as an Algorithm Engineer with Alibaba. Her current research interests include malicious device detection, graph representation learning, networks security situation, and big data.

QUAN LU received the Ph.D. degree from the University of Southern California. He is the Chief Algorithm Expert of Alibaba Group. Before joining Alibaba, he led Yahoo's Applied Research and Experian teams. He has 15 years of industry experience successfully developing large-scale machine learning, data mining, and deep learning systems.



XINMING CHEN is an Algorithm Expert of Alibaba Group. His research interests include big data and security algorithms.