Structural Temporal Graph Neural Networks for Anomaly Detection in Dynamic Graphs

Lei Cai*
Washington State University
Pullman, Washington, USA

Zhengzhang Chen†
NEC Laboratories America, Inc.
Princeton, New Jersey, USA

Chen Luo
Amazon, Inc.
Palo Alto, California, USA

Jiaping Gui
Stellar Cyber, Inc.
Santa Clara, California, USA

Jingchao Ni
NEC Laboratories America, Inc.
Princeton, New Jersey, USA

Ding Li
Peking University
Beijing, China

Haifeng Chen
NEC Laboratories America, Inc.
Princeton, New Jersey, USA

ABSTRACT

Detecting anomalies in dynamic graphs is a vital task, with numerous practical applications in areas such as security, finance, and social media. Existing network embedding based methods have mostly focused on learning good node representations, whereas largely ignoring the subgraph structural changes related to the target nodes in a given time window. In this paper, we propose **StrGNN**, an end-to-end structural temporal Graph Neural Network model for detecting anomalous edges in dynamic graphs. In particular, we first extract the h-hop enclosing subgraph centered on the target edge and propose a node labeling function to identify the role of each node in the subgraph. Then, we leverage the graph convolution operation and Sortpooling layer to extract the fixed-size feature from each snapshot/timestamp. Based on the extracted features, we utilize the Gated Recurrent Units to capture the temporal information for anomaly detection. We fully implement **StrGNN** and deploy it into a real enterprise security system, and it greatly helps detect advanced threats and optimize the incident response. Extensive experiments on six benchmark datasets also demonstrate the effectiveness of **StrGNN**.

CCS CONCEPTS

• **Theory of computation** → **Dynamic graph algorithms**; • **Computing methodologies** → **Anomaly detection**; **Neural networks**; **Learning latent representations**; • **Information systems** → *Data mining*; • **Security and privacy** → *Intrusion detection systems.*

KEYWORDS

anomaly detection; intrusion detection; dynamic graphs; graph embedding; graph neural network; graph structural feature extraction

ACM Reference Format:
Lei Cai, Zhengzhang Chen, Chen Luo, Jiaping Gui, Jingchao Ni, Ding Li, and Haifeng Chen. 2021. Structural Temporal Graph Neural Networks for Anomaly Detection in Dynamic Graphs. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3481955

1 INTRODUCTION

Recent studies of dynamic graphs/networks have witnessed a growing interest [9, 11, 13, 52]. Such dynamic graphs model a variety of systems including societies, ecosystems, the Internet, and others [10, 12, 44]. For example, in enterprise dynamic network [8, 33], the node represents a system entity (such as process, file, and Internet sockets) and an edge indicates the corresponding interaction between two system entities. These dynamic networks, unlike static networks, are constantly changing. Possible changes include graph structure change or modification of node attributes.

A fundamental task on dynamic graph analysis is anomaly detection–identifying objects, relationships, or subgraphs, whose "behaviors" significantly deviate from underlying majority of the network [1, 14, 17, 18, 36, 41, 48]. In this work, we focus on the anomalous edge detection in dynamic graphs. Detecting anomalous edges can help understand the system status and diagnose system fault [2, 36, 41]. For example, in an enterprise dynamic network, some system entity pairs, such as a user software and system-specific internet socket ports (*e.g.*, port number $\leq$ 1024), never form an edge (interaction/connection) in-between in normal system environments. Once occurring, these suspicious interactions/activities may indicate some serious cyber-attack happened and could significantly damage the enterprise system [18, 49].

To detect such anomalous interactions in dynamic graphs, a typical approach is to build a two-stage model [41]. In the first stage, data-specific features or low-dimensional representations (such as scan statistics [39], Eigen equation compression [25], or graph embedding [52]) are generated by finding the best mapping from dynamic graphs to a vector of real numbers. Then, in the second step,

*Work done during an internship at NEC Laboratories America.
†Corresponding author: zchen@nec-labs.com.

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3481955

a traditional anomaly detector, such as the support vector machines and the local outlier factor algorithm, is applied to identify anomalies [41]. As can be seen, the key step of the two-stage approach is to learn effective low-dimensional representations/features from dynamic graphs.

Recently, graph embedding has shown to be a powerful tool in learning the low-dimensional representations in networks that can capture and preserve the graph structure. However, most existing graph embedding approaches are designed for static graphs, and thus may not be suitable for a dynamic environment in which the network representation has to be constantly updated. Only a few advanced embedding-based methods (such as NetWalk [52]) are suitable for updating the representation dynamically as the network evolves. These methods first learn the node embeddings to encode edges. Then, a clustering-based method (such as K-means clustering [30]) is used to flag anomalous edges. However, similar to all other two-stage approaches, they cannot be trained end-to-end because the parameters in the two stages are not learned jointly and the objective of embedding learning is not designed for the anomaly detection task. Thus, the learned embedding may not be distinguishable for detecting the anomalies in dynamic graphs. In addition, these embedding based methods learn the graph embedding in an incremental way, *i.e.*, using all vertices and edges until the current timestamp to learn the node embedding. Thus, it can not capture some temporal dynamics like vertex removal or edge removal, which often indicates important temporal relationship changes. For example, one host/machine, which was removed (*i.e.,* a vertex removal) from the enterprise network for a long time, suddenly starts to connect to some other active hosts. It may indicate the target host has been compromised and tried to spread Trojan or other malware.

More importantly, these methods neglect a notable characteristic of the dynamic networks—the subgraph structural changes related to the target nodes. These structural temporal dynamics are key to understanding system behavior. For example, in Figure 1, the target edge at timestamp t is marked as a double red line, and the 1-hop subgraph centered on the target edge is marked with gray. It can be seen from Figure 1 (A) that the interactions between nodes of the subgraph (*i.e.*, gray nodes) become more frequent. Therefore, the target edge in Figure 1 (A) is reasonable to be a normal edge. In contrast, in Figure 1 (B), there are no interactions between the neighbors of the subgraph from timestamp $t-3$ to $t-1$. Therefore, the target edge at timestamp t is more likely to be an anomalous edge. Thus, it is critical to model and detect the structural changes over time for the anomaly detection task.

To address the aforementioned issues, we propose **StrGNN**, a structural graph neural network to identify anomalous edges in dynamic graphs. **StrGNN** is designed to detect unusual subgraph structures centered on the target edge in a given time window while considering the temporal dependency. **StrGNN** consists of three sub-models: **ESG** (Enclosing Subgraph Generation), **GSFE** (Graph Structural Feature Extraction), and **TDN** (Temporal Detection Network). First, **ESG** extracts a h-hop enclosing subgraph centered on the target edge from each graph snapshot. Subgraphs extracted based on different edges can result in the same topology structure. Thus, a node labeling function is proposed to indicate

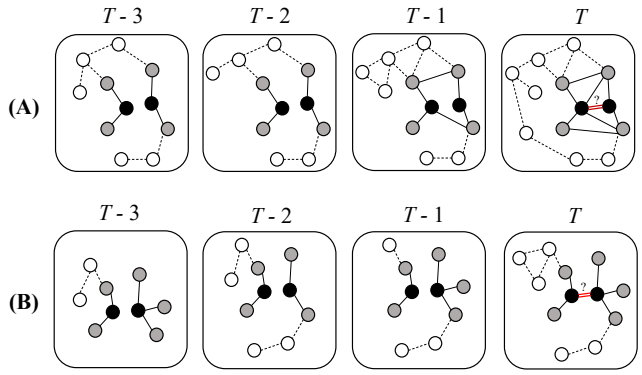

Figure 1: An example of structural changes in dynamic graphs.

the role of each node in the subgraph. Then, **GSFE** module leverages Graph Convolution Neural Network and pooling technologies to extract fixed-size feature from each subgraph. Based on the extracted features, **TDN** employs the Gated Recurrent Units (GRUs) to capture the temporal dependency for anomaly detection. Different from the previous embedding based methods, the whole process of **StrGNN** can be trained end-to-end, *i.e.*, **StrGNN** takes the test edges along with the original dynamic graphs as input and directly outputs the category (*i.e.*, anomaly or normal) for each test edge. Moreover, our proposed **StrGNN** framework focuses on mining the structural temporal patterns in a given time window and does not require to learn node embedding. Therefore, **StrGNN** is not sensitive to the edge and vertex changes (such as new nodes) in the dynamic graphs. We fully implement **StrGNN** and deploy it into a enterprise security system, which greatly helps detect advanced threats, and optimize the incident response. By using **StrGNN** in the enterprise security system for four weeks, we can reduce false positives of the state-of-the-art methods by at least 50% while keeping zero false negatives. We also conduct extensive experiments on six benchmark datasets to evaluate the performance of **StrGNN**. The results show that **StrGNN** significantly outperforms both network embedding based baselines and traditional graph anomaly detection based techniques.

The main contribution of the paper has been summarized as follows:

- We propose a novel structural Graph Neural Network **StrGNN** to identify anomalous edges in dynamic graphs.
- The proposed **StrGNN** focuses on detecting subgraph structural changes centered on the target edge in a given time window. To distinguish the role of each node in the subgraph, we propose a node labeling function to annotate the nodes in the subgraph with different labels. Different from the existing two-stage embedding based methods, our proposed method can be trained end-to-end. Therefore, the feature extraction module can be trained more efficient to capture the structural variation for anomaly detection.
- We fully develop the detection engine and deploy it into a real enterprise security system. We also conduct extensive

experiments on six benchmark datasets. Experimental results demonstrate the effectiveness of **StrGNN**.

The rest of the paper is organized as follows. Section 2 discusses our proposed structural temporal graph neural network in detail. Section 3 provides the experiment results. Section 4 discusses the related work. Finally, Section 5 concludes the paper.

2 METHOD

In this section, we introduce our method in detail. We start with the overall framework of our proposed Structural Temporal Graph Neural Networks for anomaly detection in dynamic graphs. The details of each component in our proposed method are introduced afterwards.

2.1 Overall Framework

Given a temporal network $\{G(t) = \{V(t), E(t)\}\}_{t=1}^{n}$, where $G(t)$ is the graph snapshot at timestamp t consisting of vertices $V(t)$ and edges $E(t)$. Our goal is to detect the anomalous edges at any timestamp t during the testing stage.

Compared with the anomaly detection in a static graph, dynamic graphs are more complex and challenging in two perspectives: (1) The anomalous edges cannot be determined by the graph from a single timestamp. The detection procedure must take the previous graphs into consideration; (2) Both the vertex and edge sets are changing over time. To tackle these challenges, we propose **StrGNN**, a structural temporal Graph Neural Network framework. The key idea of our proposed method is to capture structural changes centered on the target edge in a given time window and determine the category (*i.e.*, anomaly or normal) of the target edge based on the structural changes. Our proposed **StrGNN** framework consists of three key components: **ESG** (Enclosing Subgraph Generation), **GSFE** (Graph Structural Feature Extraction), and **TDN** (Temporal Detection Network), as illustrated in Figure 2.

2.2 ESG: Enclosing Subgraph Generation

For the first module, Enclosing Subgraph Generation, our goal is to generate enclosing subgraph structure related to the target edge so as to detect the anomalies more efficiently. Directly employing the whole graph for analysis can be highly computational expensive, especially considering the real-world networks with thousands or even millions of nodes and edges. Recent work [51] also proved that in Graph Neural Networks, each node is most influenced by its neighbors. Therefore, in anomalous edge detection tasks, the subgraph structure centered on the target edge (or enclosing subgraph) can be employed to detect anomaly more efficiently both in memory and computation aspects. We define the enclosing subgraph as follows:

Definition 1. (Enclosing subgraph in static graphs) For a static network $G = (V, E)$, given a target edge e with source node x and destination node y, the h–hop enclosing subgraph $G_{x,y}^{h}$ centered on edge e can be obtained by $\{i | d(i, x) \leq h \vee d(i, y) \leq h\}$, where $d(i, x)$ is the shortest path distance between node i and node x.

Definition 2. (Enclosing subgraph in dynamic graphs) For a temporal network $\{G(i) = \{V(i), E(i)\}\}_{i=t-w+1}^{t}$ with window size w, given a target edge e^t with source node x^t and destination node y^t, the h–hop enclosing subgraph G_{x^t,y^t}^{h} centered on edge e^t is a collection of all subgraph centered on e^t in the temporal network $\{G(i)_{x^t,y^t}^{h} | (t - w + 1) \leq i \leq t\}$.

For a target edge e^t, we extract the enclosing subgraph in dynamic graphs based on Definition 2. However, the extracted subgraph only contains topological information. Subgraphs extracted based on different edges can result in the same topological structure. To distinguish the role of each node in the subgraph, in this work, we propose to annotate the nodes in the subgraph with different labels.

A good node labeling function should convey the following information: 1) which edge is the target edge in the current subgraph, and 2) the contribution of each node in identifying the category of each edge. More specifically, given the edge e^t and the corresponding source and destination node x^t and y^t, we employ the following node labeling function to label each node i in the enclosing subgraph $G(i)_{x^t,y^t}^{h}$:

$$f(i, x^t, y^t) = 1 + \min(d(i, x^t), d(i, y^t)) + (d_{sum}/2)[(d_{sum}/2) + (d_{sum}\%2) - 1], \quad (1)$$

where $d(i, x^t)$ is the shortest path distance between node i and node x^t, and $d_{sum} = d(i, x^t) + d(i, y^t)$.

In addition, the two center nodes are labeled with 1. If a node i satisfies $d(i, x^t) = \infty$ or $d(i, y^t) = \infty$, it will be labeled as 0. The label will be converted into a one-hot vector as the attribute X for each node. By employing the node labeling function, we can generate the label for each node, which can represent structure information for the given subgraph. The category of the target edge e^t at timestamp t can be predicted by analyzing the labeled subgraph in the given time window.

2.3 GSFE: Graph Structural Feature Extraction

To analyze the structure of each enclosing subgraph from the given time period, the Graph Convolution Neural Network (GCN) [27] can be employed to project the subgraph into an embedding space. In GCN, the graph convolution layer was proposed to learn the embedding of each node in the graph and aggregate the embedding from its neighbors. The layer-wise forward operation of graph convolution layer can be described as follows:

$$G(X, A) = \sigma(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}XW), \quad (2)$$

where $\hat{A} = A + I$ is the summation of the adjacency matrix and identity matrix, $\sigma(\cdot)$ denotes an activation function, such as the $ReLU(\cdot) = max(0, \cdot)$, and W is the trainable weight matrix. By employing the graph convolution layer, each node can aggregate the embedding from its neighbors. By stacking the graph convolution layer in the neural network, each node can obtain more information from other nodes. For example, each node can obtain information from its 2-hop neighbors by stacking two graph convolution layers.

GCN can generate node embedding for detecting anomalous edges in a single graph [7]. However, in our dynamic graph setting, the anomalies should be determined in the context of $\{G(i)_{x^t,y^t}^{h} | t - w \leq i \leq t\}$. The number of nodes in different enclosing subgraphs is commonly different, thus results in different sizes of the feature

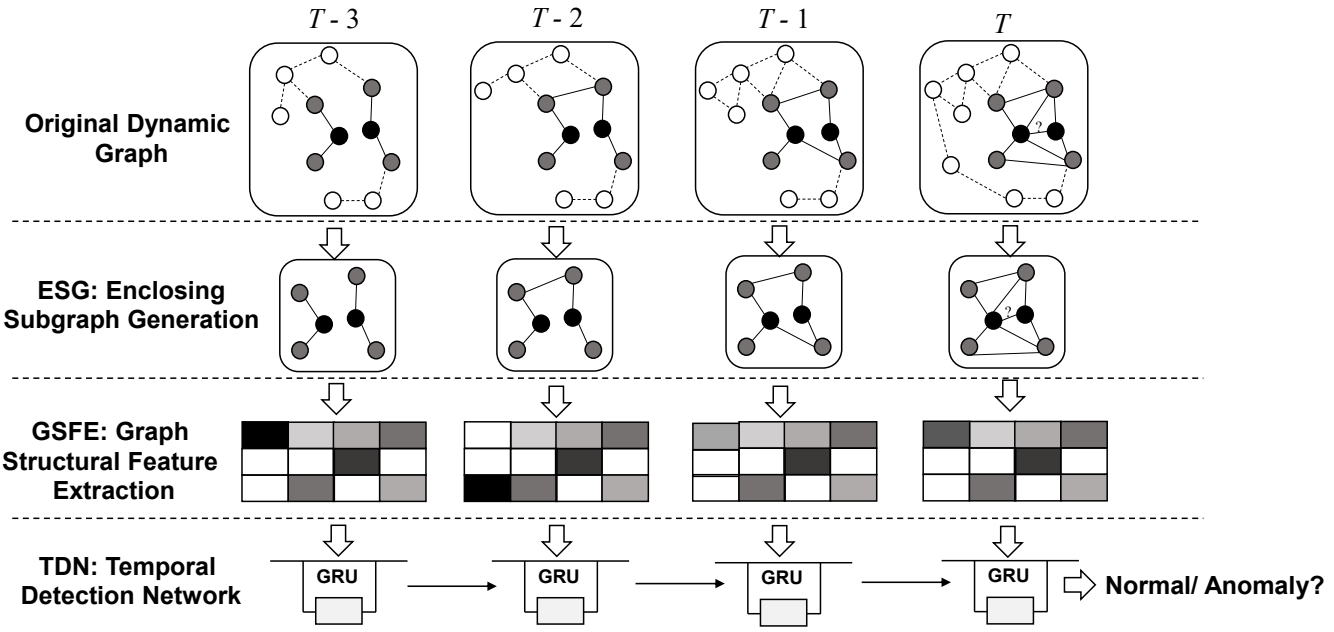

Figure 2: Illustration of our proposed StrGNN framework.

vector in different subgraphs. Therefore, it is challenging to analyze the dynamic graphs using Graph Neural Networks due to the various sizes of the input.

To tackle this problem, we leverage the graph pooling technology to extract the fixed-size feature for each enclosing subgraph. Any graph pooling method can be employed in our proposed **StrGNN** framework to extract the fixed-size feature for further analysis. In this work, we employ the Sortpooling layer proposed by [54], which can sort the nodes in the enclosing subgraph based on their importance and select the feature from the top K nodes.

Given the node embedding H_i corresponding to graph $G(i)^h_{x^t,y^t}$, the importance score for each node in the Sortpooling layer is defined as follows:

$$S(H_i, A) = \sigma(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}H_i W^1), \tag{3}$$

where A is the adjacency matrix of graph $G(i)^h_{x^t,y^t}$, and W^1 is the projection matrix with output channel 1. Each node can obtain the importance score by using Equation 3. All nodes in the enclosing subgraph will be sorted in order of the importance score. And only the top K nodes will be selected for further analysis. For the subgraphs that contain less than K nodes, the zero-padding will be employed to guarantee that each subgraph contains the same fixed-size feature.

2.4 TDN: Temporal Detection Network

The Graph Structural Feature Extraction module can generate low-dimensional features for anomaly detection. However, it does not consider the temporal information, which is of great importance for determining the category (*i.e.*, anomaly or normal) of an edge in the dynamic setting.

Given the extracted structural feature $\{\hat{H}_i\}^t_{i=t-w}$, $H_i \in R^{K\times d}$, where K is the number of selected nodes in each graph, and d is the dimension of feature for each node, in this work, we employ the Gated Recurrent Units (GRUs) [15], which can alleviate the vanishing and exploding gradient problems [20], to capture the temporal information as:

$$z_t = \sigma(W_z\hat{H}_t + U_z h_{t-1} + b_z) \tag{4}$$
$$r_t = \sigma(W_r\hat{H}_t + U_r h_{t-1} + b_r) \tag{5}$$
$$h_t^{'} = \tanh(W_h\hat{H}_t + U_h(r_t \circ h_{t-1}) + b_h) \tag{6}$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ h_t^{'}, \tag{7}$$

where $\circ$ represents the element-wise product operation, W, U, and b are parameters. The GRU network takes the feature at each timestamp as input, and feeds the output of current timestamp into the next timestamp. Therefore, the temporal information can be modeled by the GRU network. The output of last timestamp h_t is employed to analyze the category of the target edge e^t. The anomalous edge detection problem can be formulated as follows:

$$L = -(y^t \log(g(h_t)) + (1 - y^t)\log(1 - g(h_t))), \tag{8}$$

where $g(\cdot)$ is a fully connected network, and y^t is the category of edge e^t.

For the anomaly detection task, in many real-world cases, the dataset does not contain any anomalous samples or only contain a small number of anomalous samples.

One straightforward way of generating negative samples is to draw samples from a "context-independent" noise distribution (such as Random sampling or injected sampling [2]), where a negative sample is independent and does not depend on the observed samples. However, due to the large anomalous edge space, this noise distribution would be very different from the data distribution,

which would lead to poor model learning. Thus, in this work, we propose "context-dependent" negative sampling strategy.

The intuition behind our strategy is to generate negative samples from "context-dependent" noise distribution. Here, the "context-dependent" noise distribution for the sampled data E' is defined as: $P_{E'} \sim P(E) * (\frac{1}{N*|E|})$, where $P(E)$ denotes the observed data distribution, $|E|$ is the number of edges in the graph, and N is the number of nodes in the graph. Specifically, we first randomly sample a normal vertex pair $e = \{x_a, x_b\}$ in the graph. Then, we replace one of the nodes, say x_a with a randomly sampled node x' in the graph and form a new negative sample $e' = \{x', x_b\}$. If e' is not belongs to the normal graph, we retain the sample, otherwise, we delete it.

The overall training procedure is summarized as in Algorithm 1. The proposed **StrGNN** framework is quite flexible and easy to be customized. Any network that can capture the temporal information can be used in our proposed framework, such as Convolution Neural Network (CNN) and Vanilla Recurrent Neural Network (RNN).

Algorithm 1: StrGNN: Structural Graph Neural Networks for Anomaly Detection in Dynamic Graphs

Input: $\{G(t) = \{V(t), E(t)\}\}_{t=1}^{n}$; The number of hops h; Window size w.

1 Generating negative samples from "context-dependent" noise distribution if necessary;
2 Extract h-hop enclosing subgraphs based on Definition 2;
3 Generate the label for each node in the subgraph using Equation 1;
4 Extract the graph structure features;
5 Model the temporal information using GRU;
6 Train the detector using Equation 8.

3 EXPERIMENTS

In this section, we evaluate **StrGNN** on six benchmark datasets and a real enterprise network.

3.1 Datasets

We conduct experiments on six public datasets from different domains. We construct one graph per timestamp for each dataset. The statistics of the preprocessed datasets are shown in Table 1. The UCI Messages dataset [35] is collected from an online community platform of students at the University of California, Irvine. Each node in the constructed graph represents a user in the platform. And the edge indicates that there is a message interaction between two users. The Digg dataset [16] is collected from a news website digg.com. Each node represents a user of the website, and each edge represents a reply between two users. The Email dataset is a dump of emails of Democratic National Committee. Each node corresponds to a person. And the edge indicates an email communication between two persons. The Topology [53] dataset is the network connections between autonomous systems of the Internet. Nodes are autonomous systems, and edges are connections between autonomous systems. The Bitcoin-alpha and Bitcoin-otc [28, 29] datasets are collected from two Bitcoin platform named Alpha and OTC, respectively. Nodes represent users from the platform. If one user rates another user on the platform, there is an edge between them.

Table 1: Statistics of the datasets in dynamic graph setting.

Dataset	#Vertex	#Edge	#Timestamp
UCI Messages	1,899	13,838	190
Digg	30,360	85,155	16
Email	2,029	3,724	20
Topology	34,761	107,661	21
Bitcoin-alpha	3,783	14,124	63
Bitcoin-otc	5,881	21,492	63

3.2 Baselines

We compare **StrGNN** with two traditional graph anomaly detection based methods and four network embedding based baselines.

- **SedanSpot** [19]: SedanSpot is a principled randomized algorithm. It uses a holistic random walk based edge anomaly scoring function to compare an incoming edge with the whole (sampled) graph.
- **CM-Sketch** [40]: CM-Sketch is an anomaly detection model based on global and local structural properties of an edge stream. It utilizes Count-Min sketch for approximating these properties.
- **Node2Vec** [23]: Node2Vec combines breadth-first traversal and depth-first traversal in the random walks generation procedure. The embedding is learned using Skip-gram technology.
- **Spectral Clustering** [47]: To preserve the local connection relationship, the spectral embedding generates the node embedding by maximizing the similarity between nodes in the neighborhood.
- **DeepWalk** [38]: DeepWalk generates the random walks with given length starting from a node and learns the embedding using Skip-gram.
- **NetWalk** [52]: NetWalk generates several random walks for each vertex and learns a unified embedding for each node using auto-encoder technology. The embedding representation will be updated along the time dimension.

For the first three baselines, after representation learning, the same K-means clustering based method (as in NetWalk [52]) is used for anomaly detection.

3.3 Experiment Setup

The parameters of **StrGNN** can be tuned by 5-fold cross-validation on a rolling basis. Here, by default, we set the window size w to 5 and the number of hops h in enclosing subgraph to 1. We employ a Graph Neural Network with three graph convolution layers to extract graph features. The size of the output feature map is set to 32 for all three layers. The outputs of all three layers are concatenated as the embedding feature. The selected rate in the Sortpooling layer

Table 2: AUC comparison on six benchmark datasets.

Methods	UCI			Digg			Email		
	1%	5%	10%	1%	5%	10%	1%	5%	10%
SedanSpot	0.7342	0.7156	0.7061	0.6976	0.6784	0.6396	0.7427	0.7362	0.7235
CM-Sketch	0.7320	0.6968	0.6835	0.6884	0.6675	0.6358	0.7053	0.6946	0.6876
Node2Vec	0.7371	0.7433	0.6960	0.7364	0.7081	0.6508	0.7391	0.7284	0.7103
Spectral Clustering	0.6324	0.6104	0.5794	0.5949	0.5823	0.5591	0.8096	0.7857	0.7759
DeepWalk	0.7514	0.7391	0.6979	0.7080	0.6881	0.6396	0.7481	0.7303	0.7197
NetWalk	0.7758	0.7647	0.7226	0.7563	0.7176	0.6837	0.8105	0.8371	0.8305
StrGNN	**0.8179**	**0.8252**	**0.7959**	0.8162	0.8254	0.8272	**0.8775**	**0.9103**	**0.9080**
Methods	Bitcoin-Alpha			Bitcoin-otc			Topology		
	1%	5%	10%	1%	5%	10%	1%	5%	10%
SedanSpot	0.7380	0.7264	0.7085	0.7346	0.7284	0.7156	0.6873	0.6742	0.6672
CM-Sketch	0.7146	0.7015	0.6887	0.7412	0.7338	0.7242	0.6687	0.6605	0.6558
Node2Vec	0.6910	0.6802	0.6785	0.6951	0.6883	0.6745	0.6821	0.6752	0.6668
Spectral Clustering	0.7401	0.7275	0.7167	0.7624	0.7376	0.7047	0.6685	0.6563	0.6498
DeepWalk	0.6985	0.6874	0.6793	0.7423	0.7356	0.7287	0.6844	0.6793	0.6682
NetWalk	0.8385	0.8357	0.8350	0.7785	0.7694	0.7534	0.8018	0.8066	0.8058
StrGNN	**0.8574**	**0.8667**	**0.8627**	**0.9012**	**0.8775**	**0.8836**	**0.8553**	**0.8352**	**0.8271**

is set to 0.6. In terms of the temporal neural network, the hidden size of GRU is set to 256. We employ Adam method [26] to train the network. The learning rate of Adam is set to $1e-4$. We employ batch training in the experiments and the batch size is set to 32 for our proposed **StrGNN** method. **StrGNN** is end-to-end trained for 50 epochs. We use the first 50% edges as the training dataset, and the rest as the test dataset. Due to the challenges in collecting data with ground-truth anomalies, we use anomaly injection method to create the anomalous edges [2]. The metric used to compare the performance of different methods is AUC (the area under the ROC curve). The higher AUC value indicates the high quality of the method.

3.4 Results on Benchmark Datasets

We first compare **StrGNN** with the baseline methods on six benchmark datasets with different percentages (*i.e.*, 1%, 5%, and 10% as indicated in Table 2) of anomalous edges injected. The experimental results in Table 2 show that **StrGNN** outperforms all four embedding based baselines on all the benchmark datasets. In particular, on Bitcoin-otc data, **StrGNN** has gained more than 10% improvement over all four baseline methods. This is because our proposed **StrGNN** method can be trained end-to-end. Therefore, the feature extraction module can be trained more efficient to capture the structural variation for anomaly detection. In contrast, the embedding based baselines are two-stage methods. During the first stage, the embedding based method is used to generate embedding for each node in the graph. Thus, the training of embedding does not take the anomaly detection into consideration. This limits the performance of embedding based method. However, most of the network embedding-based approaches (*i.e.*, Node2Vec, DeepWalk, and NetWalk) still outperform the traditional models (*i.e.*, SedanSpot and CM-Sketch). The results also show that compared with the baseline methods, **StrGNN** is less sensitive to the number of injected anomalies. And even if 10% anomalies are injected, the performance of **StrGNN** is still acceptable.

Table 3: AUC results with different hops of enclosing subgraph on UCI Messages.

	1%	5%	10%
1-hop enclosing subgraph	0.8179	0.8252	0.7959
2-hop enclosing subgraph	0.8216	0.8274	0.7987
3-hop enclosing subgraph	0.8227	0.8294	0.8005

Table 4: AUC results with different sizes of time window on UCI Messages.

	1%	5%	10%
$w = 3$	0.7565	0.634	0.7478
$w = 4$	0.7987	0.8048	0.7646
$w = 5$	0.8179	0.8252	0.7959
$w = 6$	0.8186	0.8218	0.7937
$w = 7$	0.8148	0.8197	0.7924
$w = 10$	0.8086	0.8136	0.7879

3.4.1 Parameter Sensitivity Analysis. In the experiments, we evaluate the influence of each parameter: the number of hops h and the window size w, respectively. The AUC results of **StrGNN** with

different h on UCI Messages (with accumulated graph setting) are shown in Table 3. **StrGNN** with 2-hop or 3-hop subgraph achieves similar performance as 1-hop but requiring way more computational cost. Table 4 shows that **StrGNN** achieves a slightly better performance with the parameter $w = 5$, but the overall performance haven't been significantly affected by the window size when $5 \leq w \leq 10$.

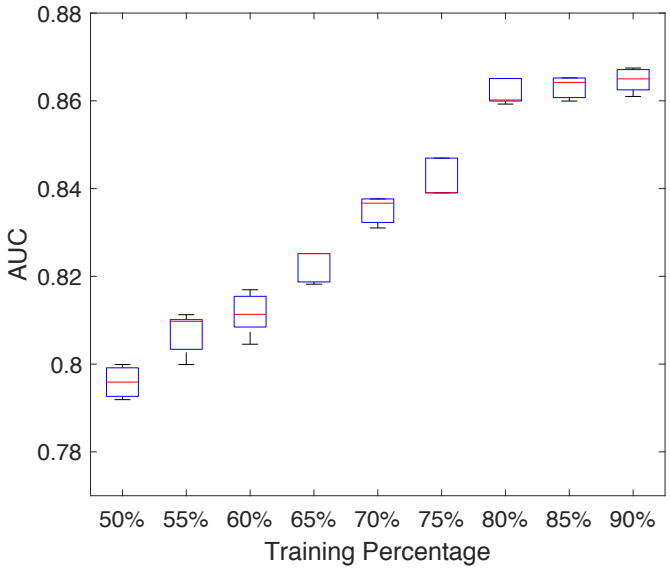

Figure 3: Stability over the training percentage of StrGNN on UCI Messages with 10% anomalies.

We also evaluate our proposed model using training data with different ratios. The AUC results on UCI Messages are shown in Figure 3. It can be seen from the results that the AUC increases with the percentage of training data ranging from 50% to 75%, and then the performance stays relatively stable.

3.5 Results on Intrusion Detection Application

To evaluate the effectiveness of **StrGNN** on practical applications with real anomalies, we apply it to detect malware attacks in the enterprise environment. We collect a 4-week period of data (with about ten thousand normal process-level and network-level event records) from a real enterprise network composed of 109 hosts (87 Windows hosts and 22 Linux hosts).

We also need to prepare the ground truth of "abnormal events" to evaluate the proposed approach. Unfortunately, in reality, compared to normal events the intrusion attacks rarely happen. In one single enterprise network, real attacks happen only several times in a year. Therefore, we want to make the evaluation as close to real scenarios as possible. We internally build the "attack testbed", and randomly pick a few hosts as targets on which different types of attacks are injected. In particular, we collaborate with an industrial company working on commercial enterprise security products. The attacks were performed by professional hackers hired by the company. We choose six typical attacks [5, 31] in the following:

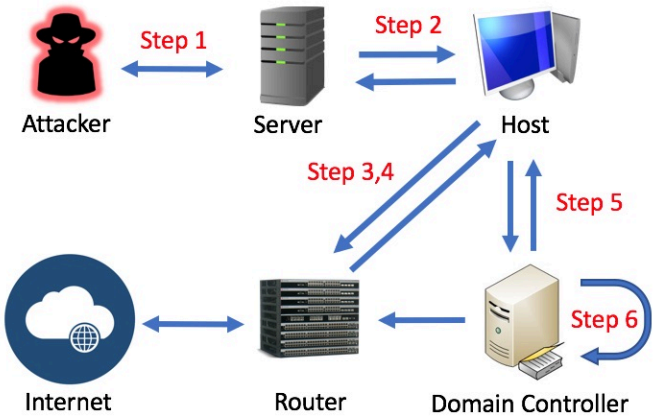

Figure 4: Attack testbed example related to the Diversifying Attack Vectors attack.

(1) *Diversifying Attack Vectors.* This intrusion scenario is a six-step attack chain [5], as shown in Figure 4. First, hackers create malicious php files, download malware binary (*trojan.exe*), and connect back to them. Then, they run the process *notepad.exe* to perform DLL injection. Next, they use *mimikatz* and *kiwi* to perform memory operation inside the *meterpreter* context. Finally, they copy and run *PwDump7.exe* and *wce.exe* on target hosts.

(2) *Emulating Enterprise Environment.* This intrusion includes seven steps. First, attackers generate telnet processes to create malware binary, open reverse connection, and download malware binary (*trojan.exe*). Then, *trojan.exe* is created to connect back to hackers, and DLL is injected through the running process *notepad.exe*. The hackers use *mimikatz* and *kiwi* for memory operation inside the *meterpreter* context. Finally, malware *PwDump7.exe* and *wce.exe* are copied and run on target hosts.

(3) *Domain Controller Penetration.* In this five-step attack chain, the hackers first send an email attaching a document that includes the malware *python32.exe*. This malware opens a connection back to hackers so that they can run *notepad.exe* and perform reflective DLL injection to obtain needed privileges. Then, they transfer password enumerator and run the process *gsecdump-v2b5.exe* to get all user credentials. Finally, they probe the SQL server address and dump the database into their own bases.

(4) *MLS Attack (MLS)*: This attack targets at the */selinux/mls* file, which defines the Multi-Level Security(MLS) classification of files within the host. In general, the */selinux/mls* file should be kept secret to all users except for security administrator, as it exposes security rules of a computer system and enables attackers to find potential vulnerabilities. By the intrusion attack, attackers first exploit the *ssh* process to access */selinux/mls* file. If the file access is successful, file content is sent to attacker's hosts.

(5) *Snowden Attack (SNO)*: This attack targets at the */etc/passwd* file, which stores the password digest of all users as well as user group information. First, the attacker tries to access */etc/passwd* file by *gvfs* process, which enables easy access from a remote host via FTP. Then the attacker tries to send the file via an *INETSocket*.
(6) *Botnet Attack (BOT)*: In this attack, the remote intruder employs the *bash* process to scan a sensitive file */var/log/apt/history.log*. This file stores detailed installation messages. Attackers are interested in it as they can exploit the vulnerabilities of installed softwares. The sensitive information is leaked via an *INETSocket*.

Table 5: Results on intrusion detection.

Method	AUC
SedanSpot	0.76
CM-Sketch	0.68
Node2Vec	0.71
Spectral Clustering	0.65
DeepWalk	0.76
Netwalk	0.90
StrGNN	**0.99**

We consider all events within the first three weeks as the "long-term normalities". Then, the malicious events by the hackers executed during different periods of the fourth week. In total, there are 82 attack records by executing different types of attacks including ATP attacks, Trojan attacks, and Puishing Email attacks. Based on the enterprise network event data, we construct an accumulated graph per day with nodes representing hosts and edges representing the network connection relationships. Based on the constructed graphs, we apply **StrGNN** and the baseline methods to detect the attacks.

The AUC results are shown in Table 5. We can see that **StrGNN** achieves an increase of 9% − 34% in AUC over the six baseline methods. Based on the experimental results, we also find that with the optimal hyperparameter setting, **StrGNN** can capture all 82 true alerts, while the baseline methods can only capture 72 true alerts at most. Meanwhile, **StrGNN** only generates 164 false positives while the baseline methods generate at least 335 false positives. The results demonstrate the effectiveness of **StrGNN** in solving real-world anomaly detection tasks.

4 RELATED WORK

In this section, we briefly introduce previous work on anomaly detection, especially on embedding based anomaly detection for graphs.

4.1 Anomaly Detection

Many traditional machine learning methods have been proposed to tackle anomaly detection tasks [6, 24]. One-class SVM [43] was proposed to learn the boundary of normal samples, and detect the anomaly based on the learned boundary. Due to the computation cost, it takes more time on the large scale dataset. Robust covariance [37] fits the data with a pre-defined distribution, and detect the anomaly by computing the distance between the sample and estimated distribution. Isolation forest [32] was proposed to isolate and detect the anomaly. Local outliers detection [4] estimates the density of a given dataset and detects the anomaly. When the feature space of a given dataset can represent the relationship between samples, the density-based method is efficient to detect the anomaly. Only recently, the deep neural network based one-class classifier (Deep SVDD) is proposed by Ruff *et al.* [42]. Inspired by SVDD [46], the authors designed a novel one-class classification objective that has very nice theoretical properties and can effectively train CNNs for image anomaly detection tasks. Unlike Deep SVDD and SVDD that mainly focus on dense data such as images, OC4Seq [50], a multi-scale one-class recurrent neural network, was proposed for detecting anomalies in discrete event sequences.

Anomaly detection is more challenging in graph setting due to the complexity of the data. In recent years, there have been increasing interests in learning the network embedding of the graph, which can further be employed to detect anomaly combing with the tradition anomaly detection methods.

4.2 Anomaly Detection on Static Graphs

Inspired by word embedding methods [3, 34] in natural language processing tasks, recent advances such as DeepWalk [38], LINE [45], and Node2Vec [23] have been proposed to learn node embedding via the **skip-gram** technology. The DeepWalk generates random walks for each vertex with a given length and picks the next step uniformly from the neighbors. The skip-gram is employed to learn the embedding from the node sequence. To preserve the local connection relationship, the spectral embedding generates the node embedding by maximizing the similarity between neighborhood nodes. Different from DeepWalk, the LINE [45] preserves not only the first-order (observed tie strength) relations but also the second-order proximities (shared neighborhood structures of the vertices). Node2Vec [23] uses two different sampling strategies (breadth-first sampling and depth-first sampling) for vertices that result in different feature representations. Through the network embedding technology, both anomalous node and edge detection tasks can be performed with traditional anomaly detection methods.

4.3 Anomaly Detection on Dynamic Graphs

Dynamic graphs are more complex due to the variation of the graph structure. That is, the vertices and edges are changing along the time dimension. Network embedding methods such as DeepWalk, LINE, NODE2VEC can only learn the embedding on a given graph. When dealing with a series of graphs, these methods cannot capture the dependency across different graphs/snapshots. To capture the dependency between different graphs along the time dimension, recently few network embedding based methods have been proposed [55]. Dyngem [22] employs the auto-encoder method to learn the embedding for each graph, and a constraint loss function is employed to minimize the difference between all graphs. Dyngraph2vec [21] uses the Recurrent Neural Network to capture the temporal information and learn the embedding using auto-encoder technology. Recently, NetWalk [52], one of the state-of-the-art methods for anomaly detection in dynamic networks, is proposed to learn the embedding while considering the temporal dependency

and detect the anomaly using the clustering-based method. The NetWalk generates several random walks for each vertex and learns a unified embedding for each node using auto-encoder technology. The embedding representation is updated along the time dimension.

5 CONCLUSION

In this paper, we investigated an important and challenging problem of anomaly detection in dynamic graphs. Different from network embedding based methods that focus on learning good node representations, we proposed **StrGNN**, a structural temporal Graph Neural Network to detect anomalous edges by mining the unusual temporal subgraph structures. **StrGNN** can be trained end-to-end and it is not sensitive to the percentage of anomalies. We implement and deploy our approach to a real enterprise security system, and evaluate the proposed algorithm for intrusion detection tasks. Our method achieved superior detection performance with zero false negatives. We also evaluated the proposed framework using extensive experiments on six benchmark datasets. The experimental results convince us of the effectiveness of our approach.

REFERENCES

[1] Charu C. Aggarwal, Yuchen Zhao, and Philip S. Yu. 2011. Outlier Detection in Graph Streams. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*. USA, 399–409.

[2] 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.

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research* 3, Feb (2003), 1137–1155.

[4] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: Identifying Density-based Local Outliers. In *SIGMOD*, Vol. 29. ACM, 93–104.

[5] Cheng Cao, Zhengzhang Chen, and *et al.* 2018. Behavior-based Community Detection: Application to Host Assessment In Enterprise Information Networks. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 1977–1985.

[6] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly Detection: A Survey. *ACM Comput. Surv.* 41, 3, Article 15 (July 2009), 58 pages. https://doi.org/10.1145/1541880.1541882

[7] Anshika Chaudhary, Himangi Mittal, and Anuja Arora. 2019. Anomaly Detection using Graph Neural Networks. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. 346–350.

[8] Ting Chen, L. Tang, Yizhou Sun, Zhengzhang Chen, Haifeng Chen, and G. Jiang. 2016. Integrating Community and Role Detection in Information Networks. In *SDM*.

[9] Zhengzhang Chen. 2012. *Discovery of Informative and Predictive Patterns in Dynamic Networks of Complex Systems*. Ph.D. Dissertation.

[10] Zhengzhang Chen, William Hendrix, Hang Guan, Isaac K. Tetteh, Alok N. Choudhary, Fredrick H. M. Semazzi, and Nagiza F. Samatova. 2013. Discovery of extreme events-related communities in contrasting groups of physical system networks. *Data Min. Knowl. Discov.* 27, 2 (2013), 225–258.

[11] Zhengzhang Chen, William Hendrix, and Nagiza F. Samatova. 2012. Community-Based Anomaly Detection in Evolutionary Networks. *J. Intell. Inf. Syst.* 39, 1 (2012), 59–85.

[12] Zhengzhang Chen, Kanchana Padmanabhan, Andrea M. Rocha, Yekaterina Shpanskaya, James Mihelcic, Kathleen Scott, and Nagiza F. Samatova. 2012. SPICE: Discovery of Phenotype-determining Component Interplays. *BMC Syst Biol* 6, 1 (2012), 40.

[13] Zhengzhang Chen, Kevin A. Wilson, Ye Jin, William Hendrix, and Nagiza F. Samatova. 2010. Detecting and Tracking Community Dynamics in Evolutionary Networks. In *2010 IEEE International Conference on Data Mining Workshops*. 318–327.

[14] Wei Cheng, Kai Zhang, Haifeng Chen, Guofei Jiang, Zhengzhang Chen, and Wei Wang. 2016. Ranking causal anomalies via temporal and dynamical analysis on vanishing correlations. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 805–814.

[15] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[16] Munmun De Choudhury, Hari Sundaram, Ajita John, and Dorée Duncan Seligmann. 2009. Social synchrony: Predicting mimicry of user actions in online social media. In *2009 International Conference on Computational Science and Engineering*, Vol. 4. IEEE, 151–158.

[17] Boxiang Dong, Zhengzhang Chen, Lu-An Tang, Haifeng Chen, Hui Wang, Kai Zhang, Ying Lin, and Zhichun Li. 2021. Anomalous Event Sequence Detection. *IEEE Intelligent Systems* 36, 3 (2021), 5–13. https://doi.org/10.1109/MIS.2020.3041174

[18] Boxiang Dong, Zhengzhang Chen, Hui (Wendy) Wang, Lu-An Tang, Kai Zhang, Ying Lin, Zhichun Li, and Haifeng Chen. 2017. Efficient Discovery of Abnormal Event Sequences in Enterprise Security Systems. In *CIKM*.

[19] Dhivya Eswaran and Christos Faloutsos. 2018. SedanSpot: Detecting Anomalies in Edge Streams. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*. 953–958.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[21] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. 2019. dyngraph2vec: Capturing Network Dynamics Using Dynamic Graph Representation Learning. *Knowledge-Based Systems* (2019).

[22] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. Dyngem: Deep Embedding Method for Dynamic Graphs. *arXiv preprint arXiv:1805.11273* (2018).

[23] 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.

[24] Manish Gupta, Jing Gao, Charu C. Aggarwal, and Jiawei Han. 2014. Outlier Detection for Temporal Data: A Survey. *IEEE Trans. Knowl. Data Eng.* 26, 9 (2014), 2250–2267. http://dblp.uni-trier.de/db/journals/tkde/tkde26.html#GuptaGAH14

[25] Shunsuke Hirose, Kenji Yamanishi, Takayuki Nakata, and Ryohei Fujimaki. 2009. Network Anomaly Detection Based on Eigen Equation Compression. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '09)*. 1185–1194.

[26] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

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

[28] 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.

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

[30] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. 2003. The global k-means clustering algorithm. *Pattern recognition* 36, 2 (2003), 451–461.

[31] Ying Lin, Zhengzhang Chen, Cheng Cao, Lu-An Tang, Kai Zhang, Wei Cheng, and Zhichun Li. 2018. Collaborative Alert Ranking for Anomaly Detection. In *CIKM*.

[32] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 413–422.

[33] Chen Luo, Zhengzhang Chen, Lu-An Tang, Anshumali Shrivastava, Zhichun Li, Haifeng Chen, and Jieping Ye. 2018. TINET: Learning invariant networks via knowledge transfer. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1890–1899.

[34] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*. 3111–3119.

[35] Tore Opsahl and Pietro Panzarasa. 2009. Clustering in Weighted Networks. *Social Networks* 31, 2 (2009), 155–163.

[36] Kanchana Padmanabhan, Zhengzhang Chen, Sriram Lakshminarasimhan, Siddarth Shankar Ramaswamy, and Bryan Thomas Richardson. 2013. Graph-based anomaly detection. *Practical Graph Mining with R (2013)* (2013).

[37] Daniel Peña and Francisco J Prieto. 2001. Multivariate Outlier Detection and Robust Covariance Matrix Estimation. *Technometrics* 43, 3 (2001), 286–310.

[38] 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.

[39] Carey E. Priebe, John M. Conroy, David J. Marchette, and Youngser Park. 2005. Scan Statistics on Enron Graphs. *Computational & Mathematical Organization Theory* 11, 3 (01 Oct 2005), 229–247.

[40] Stephen Ranshous, S. Harenberg, K. Sharma, and N. Samatova. 2016. A Scalable Approach for Outlier Detection in Edge Streams Using Sketch-based Approximations. In *SDM*.

[41] Stephen Ranshous, Shitian Shen, Danai Koutra, Steve Harenberg, Christos Faloutsos, and Nagiza F. Samatova. 2015. Anomaly Detection in Dynamic Networks: A Survey. *WIREs Comput. Stat.* 7, 3 (May 2015), 223–247.

[42] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. 2018. Deep one-class classification. In *International Conference on Machine Learning*. 4393–4402.

[43] Bernhard Schölkopf, Robert C Williamson, Alex J Smola, John Shawe-Taylor, and John C Platt. 2000. Support Vector Method for Novelty Detection. In *Advances in Neural Information Processing systems*. 582–588.

[44] Huseyin Sencan, Zhengzhang Chen, William Hendrix, Tatdow Pansombut, Fredrick H. M. Semazzi, Alok N. Choudhary, Vipin Kumar, Anatoli V. Melechko, and Nagiza F. Samatova. 2011. Classification of Emerging Extreme Event Tracks in Multivariate Spatio-Temporal Physical Systems Using Dynamic Network Structures: Application to Hurricane Track Prediction. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. 1478–1484.

[45] 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.

[46] David MJ Tax and Robert PW Duin. 2004. Support vector data description. *Machine Learning* 54, 1 (2004), 45–66.

[47] Ulrike von Luxburg. 2007. A Tutorial on Spectral Clustering. *CoRR* abs/0711.0189 (2007). http://arxiv.org/abs/0711.0189

[48] Shen Wang, Zhengzhang Chen, Ding Li, Zhichun Li, Lu-An Tang, Jingchao Ni, Junghwan Rhee, Haifeng Chen, and S. Philip Yu. 2019. Attentional Heterogeneous Graph Neural Network: Application to Program Reidentification. In *SDM*.

[49] Shen Wang, Zhengzhang Chen, Xiao Yu, Ding Li, Jingchao Ni, Lu-An Tang, Jiaping Gui, Zhichun Li, Haifeng Chen, and S. Philip Yu. 2019. Heterogeneous Graph Matching Networks for Unknown Malware Detection. In *IJCAI*.

[50] Zhiwei Wang, Zhengzhang Chen, Jingchao Ni, Hui Liu, Haifeng Chen, and Jiliang Tang. 2021. Multi-Scale One-Class Recurrent Neural Networks for Discrete Event Sequence Anomaly Detection. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (Virtual Event, Singapore) *(KDD '21)*. 3726–3734.

[51] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *International Conference on Machine Learning*. 5449–5458.

[52] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. 2018. Netwalk: A Flexible Deep Embedding Approach for Anomaly Detection in Dynamic Networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2672–2681.

[53] Beichuan Zhang, Raymond Liu, Daniel Massey, and Lixia Zhang. 2005. Collecting the Internet AS-level topology. *ACM SIGCOMM Computer Communication Review* 35, 1 (2005), 53–61.

[54] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[55] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic Network embedding by Modeling Triadic Closure Process. In *Thirty-Second AAAI Conference on Artificial Intelligence*.