# Anomaly Detection on Dynamic Graph

Sun, Peng

*Award date:*
2022

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science

# Anomaly Detection on Dynamic Graph

*Master Thesis*

Peng Sun
p.sun1@student.tue.nl

Supervisors:
dr. Yulong Pei
dr. Meng Fang
dr. Morteza Monemizadeh

Eindhoven, September 2022

# Abstract

Anomaly detection plays an important role in many applications, like social media and e-commerce systems. Recently, the anomaly detection in the graphs has attracted increasing attention, because of their robust expressiveness and ability to represent complex relationships. Initially, the research focused on anomaly detection in static graphs that do not change over time, yet real-world application scenarios are constantly evolving. Thus, investigating anomaly detection in dynamic graphs becomes preferable.

Among the numerous approaches to detecting anomalies in dynamic graphs, deep learning based methods are now highly popular. Most of them rely on GCN(Graph Convolutional Networks) and GRU(Gate Recurrent Unit) and have good performance, but these methods acquire structural features and temporal features separately. Such approaches may fail to capture the relation between structural and temporal features which may contain important information in identifying anomalies. In this thesis, we have an inspiration for detecting anomalous edges, using merged structural and temporal features. In order to obtain merger information, we design an edge encoding strategy to represent each edge's structural and temporal feature in graph streams. In this way, the input to the transformer model is already a combination of structural and temporal encoding. Finally, experimental results on four benchmark datasets demonstrate the promising performance of our method in anomaly detection.

# Preface

This master thesis signifies the end of my computer science and engineering studies at TU/e. I complete my graduation project within the Data Mining Research Group in the Department of Mathematics and Computer Science. I have gained a lot during my two years as a master, including knowledge and skills, as well as new friendships. And I am sure that what I have gained here will help me well in my future work.

During the process of doing my graduation project and writing my thesis, many people gave me assistance and encouragement, and I am grateful to them. First of all, I would like to thank my supervisor Yulong Pei, who has always given me guidance, always responded to my doubts promptly, provided valuable advice and inspiration at the weekly meetings. Then, I would like to thank Professors Meng Fang and Morteza Monemizadeh for accepting to serve in my committee. I would also like to thank all professors and educators at the university, as well as my fellow students, for inspiring me and helping me grow and learn during my student years.

Furthermore, I would like to thank my parents for encouraging me and supporting me to pursue my master's education in the Netherlands.

# Contents

**Contents** **vii**

**List of Figures** **ix**

**List of Tables** **xi**

**1 Introduction** **1**
1.1 Problem Definition . . . 3
1.2 Outline . . . 3

**2 Related Work** **5**
2.1 Static Graphs . . . 5
2.1.1 Anomaly node detection . . . 5
2.1.2 Anomaly edge detection . . . 6
2.1.3 Anomaly subgraph detection . . . 6
2.2 Dynamic Graphs . . . 6
2.2.1 Anomaly node detection . . . 6
2.2.2 Anomaly edge detection . . . 7
2.2.3 Anomaly subgraph detection . . . 8

**3 Preliminaries** **11**
3.1 Diffusion Technique . . . 11
3.2 Transformer . . . 11
3.2.1 Encoder . . . 12
3.2.2 Decoder . . . 13
3.2.3 Position Encoding . . . 13
3.3 Spectral Clustering . . . 14
3.4 Evaluation Criteria . . . 16

**4 Proposed Anomaly Detection Approach** **19**
4.1 Edge Related Nodes Sampling . . . 19
4.2 Structure and Temporal Encoding . . . 21
4.2.1 Diffusion-based Structure Encoding . . . 21
4.2.2 Importance Structure Encoding . . . 22
4.2.3 Normalized Relative Temporal Encoding . . . 22
4.2.4 Encoding Merge . . . 23
4.3 Dynamic Graph Transformer . . . 23
4.4 Anomaly Detector . . . 24
4.5 Differences . . . 25
4.6 Complexity Analysis . . . 26

5 Experiments 27
5.1 Datasets 27
5.2 Baselines 28
5.3 Experimental Setup 28
5.3.1 Experimental Design 28
5.3.2 Parameter Settings 29
5.3.3 Experimental Infrastructures 29
5.4 Results 29
5.4.1 Results on Graphs without Timestamps 29
5.4.2 Results on Dynamic Graphs 29
5.5 Parameter Sensitivity 31
5.5.1 Parameters of Edge-related Nodes Sampling 31
5.5.2 Parameters of Transformer 32
5.5.3 Training data proportion 34
5.6 Ablation Study 36

6 Conclusions 39
6.1 Summary 39
6.2 Limitations 39
6.3 Future Work 40

Bibliography 41

Appendix 45

A My First Appendix 45

# List of Figures

1.1 An example of graph stream. Original illustration by [17] . . . . . . . . . . . . . . . 2

2.1 AddGraph and StrGNN . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

3.1 Transformer Structure. Original illustration by [28] . . . . . . . . . . . . . . . . . . . 12
3.2 Self-Attention and Multi-Head Attention. Original illustration by [28] . . . . . . . 13
3.3 Cut . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
3.4 ROC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

4.1 Sampling Example. Original illustration by [17] . . . . . . . . . . . . . . . . . . . . 20
4.2 Edge Encoding . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
4.3 Anomaly Detector . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

5.1 Sensitive of k . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
5.2 Sensitive of $\tau$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
5.3 UCI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
5.4 Btc-Alpha . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
5.5 Btc-Otc . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
5.6 UCI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
5.7 Btc-Alpha . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
5.8 Btc-Otc . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
5.9 Experiment of varying the weight of each component . . . . . . . . . . . . . . . . . 36

# List of Tables

5.1 The statistics of the datasets . . . 28
5.2 AUC results for anomalous scores on graphs without timestamps . . . 29
5.3 AUC comparison on UCI and Digg . . . 30
5.4 AUC comparison on bitcoin-Alpha and Bitcoin-OTC . . . 30
5.5 AUC comparison on benchmark Email and Topology . . . 30
5.6 Ablation Study . . . 36

# Listings

# Chapter 1

# Introduction

Anomaly is something that deviates from the standard[10] and anomaly detection, also called outlier detection, is the identification of unexpected events, observations, or items that differ significantly from the norm. There are two basic characteristics of any type of anomaly. Firstly, anomalies in data occur only rarely; secondly, the features of data anomalies are significantly different from those of normal instances.

Anomaly detection in graphs has attracted a great deal of research attention and most of the existing studies have focused on static graphs [22, 25]. However, in fields such as social media, e-commerce systems and cybersecurity, it is impossible to investigate using static graphs, since edges and nodes represent real-world things and relationships, and the real world keeps evolving and changing rapidly over time. Taking social media networks as an example, fresh people join the community at any time, and the relationships between individuals change over time. Static graphs can only represent the structure of the graph at a single timestamp and cannot represent changes in the content of the graph, which indicates a lack of temporal features. Further, the anomaly detection algorithm for static graphs cannot effectively deal with dynamic read-world data.

Because of the limitations of static graphs in modeling real-world graph data, dynamic graphs become a more proper data structure for simulating the real-world networks. Among the various anomaly detection problems of dynamic graphs, detecting anomalous edges in the evolving graph stream is a critical task. For example, in the users network of social media scenario, anomalous users tend to spread spam messages widely, which contributes to millions of newly added edges into the graph. Also, users involved in anomalies sometimes perform anomalous operations while these users operate normally most of the time, which hides their long-term anomalous behavior and makes detecting anomalous nodes more difficult. Hence, anomaly edge detection in dynamic graphs is more suitable for this scenario, the detection of such spam messages is also significant for maintaining a clean social media environment. Generally, in the dynamic graph data, we assume that the edges of the dynamic network are given as streams, and this is why in dynamic graphs we use the update stream of dynamic graphs to represent dynamism.

However, detecting anomaly in dynamic graph is not a trivial task, since there are some challenges.

- The first challenge lies in the insufficient labeled data. The data is initially normal and identifiable, but in real-world applications, anomalous data will eventually mix with normal data over time. It would be a significant burden, or even infeasible, for us to manually check for abnormal data. Thus, it is hard to find the explicit labeled data and supervised models will not perform well.

- The second challenge is that anomalous edges cannot be identified by a single time-stamped graph. The model must take into account previous graphs. For example, in Fig.1.1, the green edge tends to be normal because of the structural connections between its end nodes' neighbors in the previous timestamps. Conversely, the red edge has a higher probability of being anomalous since the two end nodes of the red edge do not share neighbors, and even their neighbors do not have communication in the previous timestamp.

- The third challenge is to capture the structural and temporal features and then combine them in a reasonable way. Dynamic graph is represented in the form of graph streams, so the model needs not only the structural features of the graphs, but also temporal information. In addition, separate processing of these two features may result in the loss of feature information.

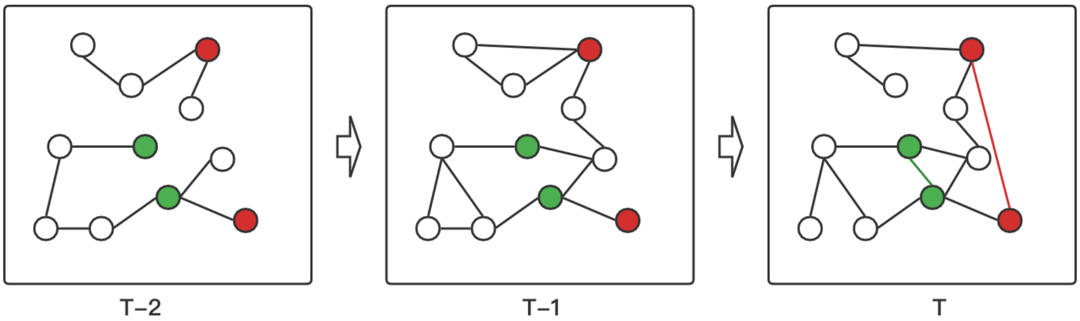


Figure 1.1: An example of graph stream. Original illustration by [17]

To detect the anomaly in dynamic graphs, several types of methods have been proposed. There have been works on graph embedding for anomaly detection [11, 33, 26]. These methods have achieved good results in detecting anomalies in dynamic graphs, whereas they cannot capture the long-term and short-term patterns of the nodes and ignore the changes in subgraph structure associated with the target nodes in dynamic graphs. In addition, deep learning-based methods have shown good results in anomaly detection, and there are some models that can be used to get the features in the graph. For example, it is common for GCN[14] to be used to obtain structural features in graphs. GCN can aggregate the information carried by each node and its neighboring nodes, then generate node embedding for detecting anomalous in a single graph. However, relying only on GCN cannot detect anomalies in dynamic graphs because GCN cannot capture the temporal features, which are essential factors in the dynamic graphs. To compensate for the limitations of GCN, many works have extended the original GCN model with GRU[6] to support temporal information.

However, many existing approaches get structure features and temporal features separately. For example, a module is responsible for generating structural embeddings and another module outputs the embeddings that contain temporal features. Then, processing these two types of features in isolation may miss the joint structure-temporal features and further lead to sub-optimal solutions.

The anomaly in the graph can be classified into three types, namely, anomalous nodes, anomalous edges and anomalous subgraphs. In this paper, we focus on detecting anomalous edges in dynamic graph and intend to process structural and temporal features together via transformer[28] model. Finally, the model outputs an anomaly score for each edge,then identify the category of the edges. Importantly, we optimize the representation formulas for structural and temporal features of dynamic graphs based on similar work[17]. Our proposed model uses the transformer as a backbone neural network because of its powerful representation capabilities. We apply the transformer to dynamic graphs, which is a more complex scenario where both structural and temporal features need to be considered.

## 1.1 Problem Definition

Before discussing anomaly detection in dynamic graph, it is necessary to define the notations we use for research. We describe a dynamic graph as a stream of graphs represented by a series of snapshots. The dynamic graph is defined as follows:

**Definition 1.** *Let $T$ be the maximum timestamp. A graph steam $\mathbb{G}$ takes the form of $\{\mathcal{G}^t\}_{t=1}^T$, where each $\mathcal{G}^t = (\mathcal{V}^t, \mathcal{E}^t)$ represents the entire snapshot at timestamp $t$, and $\mathcal{V}^t$ and $\mathcal{E}^t$ are the sets of nodes and edges, respectively. An edge $e^t = (i, j, w) \in \mathcal{E}^t$ means that the $i$-th node and the $j$-th node have a connection in the dynamic graph at the timestamp $t$ with weight $w$. For unweighted graphs, $w$ is always 1. For weighted graphs, $w \in \mathbb{R}^+$. We use $n^t = |V^t|$ and $m^t = |\mathcal{E}^t|$ to denote the number of nodes and edges at timestamp $t$. An adjacency matrix $\mathbf{A}^t \in \mathbb{R}^{m^t \times n^t}$ is to represent the edges in $\mathcal{E}^t$, where $\forall (i, j, w) \in \mathcal{E}^t, \mathbf{A}^t[i][j] = w$, otherwise $A^t[i][j] = 0$.*

Apart from that, there is no limit to the number of times an edge can appear between the same pair of nodes, and two nodes can be connected multiple times in the stream. However, we only pay attention to the neighbors associated with the two nodes of the target edge, rather than the number of edge between these two nodes. Hence, we do not use the weight option, that is, $w$ equals 1.

The goal of this thesis is to detect anomalous edges in $\mathcal{E}^t$. We formalize the anomaly detection problem as scoring problem. Given a dynamic graph $\mathbb{G} = \{\mathcal{G}^t\}_{t=1}^T$ where each $\mathcal{G}^t = (\mathcal{V}^t, \mathcal{E}^t)$, for each edge $e^t \in \mathcal{E}^t$, we produce the anomaly score $f(e^t)$ where $f(x)$ is a anomaly score function. The anomaly score indicates the anomalous probability of edge, where a larger score $f(e^t)$ means a higher anomalous probability of $e^t$. Due to the number of normal and abnormal edges is extremely imbalanced, in other words, the number of normal edges is much larger than the number of abnormal edges. After referring to some existing works[3, 33, 34] and there is not enough label data, we select unsupervised methods for anomaly detection in dynamic graphs. Specifically, in the training phase, we use the training set that contains only normal edges. In the testing phase, we use the testing set containing normal and anomalous edges, and then we can evaluate the performance of the model based on the labels and predicted values. As for the label details, the label $y_{e^t} = 1$ indicates that edge $e^t$ is an anomalous edge and $y_{e^t} = 0$ indicates that $e^t$ is a normal one.

After that, we define the research question:

*Given a stream of edges, how can we detect anomalous edges using the structural and temporal features of the graphs?*

To investigate this research question and to overcome the aforementioned challenges, this research question can be further divided into two sub-questions.

- *How do we get the structural features in the dynamic graphs?*
- *How do we describe temporal features and combine them with structural features?*

## 1.2 Outline

After the introduction, Chapter 2 reviews the related works on graph anomaly detection. Chapter 3 introduces the knowledge about the transformer and some graph-related technologies. Chapter 4 describes the key points of our model and detailed steps about how we capture the structure and temporal features and train the model. Chapter 5 elaborates experimental setups, results and analyses. Finally, we conclude our thesis with limitations and future work in Chapter 6.

# Chapter 2

# Related Work

## 2.1 Static Graphs

In this section, we review the work on anomaly detection in static graphs. The main task of detecting anomalies in static graphs is to find anomalous network entities (e.g., nodes, edges, subgraphs) in the entire graph structure. While the specific definition of the graph anomalies may vary, a general definition for the anomaly detection problem for static graphs can be stated as follows:

**Definition 2.** *Given the snapshot of a graph database, find the nodes and/or edges and/or substructures that are "few and different" or deviate significantly from the patterns observed in the graph.*

### 2.1.1 Anomaly node detection

There are several methods for detecting anomalous nodes in a static graph. First of all, some traditional methods they use the statistical features of the graph or nodes (out/in degrees, etc.) to detect anomalies. For example, OddBall[1], a method that uses 1-hop neighbors and edge weights to detect structural anomalous nodes, can also detect anomalous nodes in ring and star anomalous structures. Then, some approaches using network representation learning techniques are proposed. Such methods embed the structural information of the graph into a vector representation, which is then used in a downstream node analysis task. Renjun Hu et al.[12] firstly proposes an efficient embedding representation method to detect anomalous nodes. First, the graph partitioning algorithm is used to divide the nodes in the graph into $d$ communities, and then an embedding method is designed to learn the node representation that captures the strength of the connection between the node and the $d$ communities. If a node has a strong connection with multiple communities, the node is likely to be an anomalous node.

In further, static graphs may be attribute graphs, which contain attribute information for each node in addition to structural information between nodes, and this attribute information helps to extract the hidden features of nodes and improve the accuracy of node anomaly detection. The deep neural network-based approach is a common attribute graph anomaly detection method. Taking DONE[2] as an example, the model uses two autoencoders, namely the structural autoencoder and the attribute autoencoder, to quantify the anomaly score of each node by minimizing the reconstruction error, and then to sort the nodes according to their scores and finally the top $k$ nodes are selected as the anomaly nodes.

### 2.1.2 Anomaly edge detection

Compared to anomalous node detection, which targets individual nodes, anomalous edge detection aims to identify anomalous connections that typically indicate unexpected or unusual relationships between real entities.

There are many methods to detect anomalous edges in static graphs, like deep neural network based methods, GCN based methods and network representation based methods. Deep neural network-based approach employs autoencoders and fully connected networks for anomalous edge detection. The method proposed by Ouyang et al.[20] estimates the probability of an edge occurrence based on the properties of the two end nodes and their neighbors nodes, and then classify edges depending on probability. The GCN has the advantage of capturing graph structural information. Duan et al[7] consider that anomalous edges obstruct GCN-based models from capturing the actual edge distribution, which leads to sub-optimal detection performance. They believe that the negative impact of anomalous edges needs to be mitigated when performing node embedding. Therefore, they propose a method called AANE that iteratively updates embedding and edge detection during training to improve the performance. In each training iteration, AANE generates node embedding through the GCN layer and learns the indicator matrix to find potential anomalous edges. AS for Network representation based approach, it learns edge representation directly from the graph. The core idea of edge representation is to preserve the structural information and interaction content between pairs of nodes.

### 2.1.3 Anomaly subgraph detection

In real life, anomalies may be collusion or joint action with others in order to gain more profit. When data is represented as a graph, these anomalies and their interactions often form suspicious sub-graph structures. Different from individual graph anomalies (i.e., a single node or edge), a node or edge in an anomaly subgraph may be normal individually, however, when they are considered as a whole, they constitute an anomalous subgraph.

Methods for detecting anomalous subgraphs include traditional statistical methods and deep learning-based methods. The method proposed by Miller et al.[18] detects anomalous subgraphs by measuring the residuals between the expected subgraph structure and the observed subgraph structure. Deep learning based approach extracts features to detect anomalies using deep neural networks. Wang et al.[30] proposed a DeepFD model that expresses the online shopping network as a bipartite graph where users and products are two types of nodes. This aims to learn the anomaly representation of users so that suspicious users will be closely localized in the vector space, while benign users stay away from the anomalous group.

## 2.2 Dynamic Graphs

This section provides an overview of some anomaly detection works proposed for dynamic or time-evolving graphs. Compared to anomaly detection in static graphs, the problem of anomaly detection in dynamic graphs has recently become more popular and is usually defined as follows:

**Definition 3.** *Given a sequence of graphs, find the timestamps that correspond to a change or event, as well as the top-$k$ nodes, edges, or parts of the graphs that contribute most to the change.*

In the following subsections, we classify the dynamic graph anomaly detection algorithms based on the type of events they detect.

### 2.2.1 Anomaly node detection

Generally, an anomalous node behaves the same as a normal node in a single graph snapshot, but the dynamic changes in the time dimension can show its difference from other nodes, which helps to detect anomalous nodes. The main challenge in detecting anomalies in dynamic graphs is that dynamic graphs introduce a large amount of data, while temporal information needs to be captured for anomaly detection. Despite the challenges, there are a variety of studies on dynamic graph anomaly node detection.

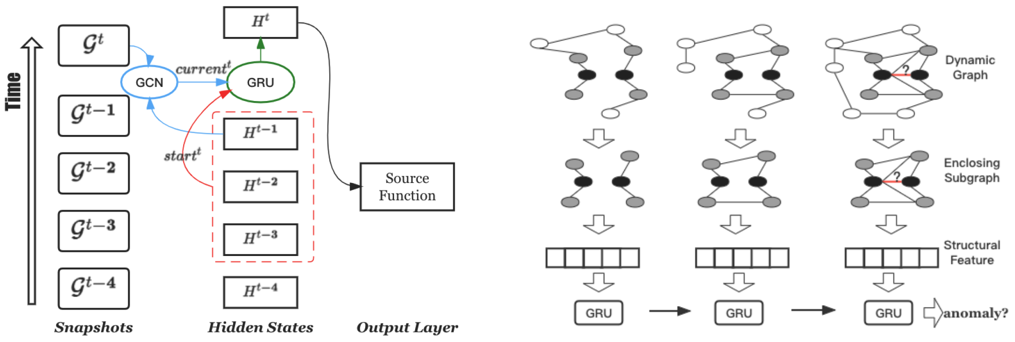
In related studies, the approaches to such question mainly consist of three categories, traditional methods, network representation-based methods and deep learning-based methods respectively. Some traditional methods[24, 31] are to model the structural changes of nodes, and assume that normal nodes usually follow a stable pattern and do not introduce large effects compared to anomalous nodes. Besides, NT Tam et al.[27] propose a non-parametric approach to detect anomalous users, tweets, tags, and links on social platforms. This approach models social platforms as heterogeneous social graphs so that the rich relationships among users, tweets, tags, and links can be captured efficiently. Anomalies are then found based on the bias of the anomalous objects. Recently, NetWalk[33], one of the network representation-based methods for anomaly detection in dynamic networks, is proposed to learn the embedding while considering the temporal factor and detect the anomaly using the density-based method. The NetWalk generates several random walks for each node and learns a embedding for each node using auto-encoder technology. Meanwhile, the representation embedding is updated along the time dimension. After that, NetWalk adopts k-means clustering algorithm to group the existing nodes in the current timestamp into different clusters, and finally the anomaly score of each node is measured as its distance to the $k$ clusters to get the anomalous nodes. Zheng et al.[36] propose a GAN(Generative Adversarial Network) based method, called OCAN, which uses only the features of normal users for anomaly detection. The basic idea of OCAN is to catch normal activity patterns and find anomalies with significantly different behavior. More specifically, OCAN firstly uses LSTM to extract the historical behavioral features of normal users. Next, a GAN model is trained to generate normal user data so that a discriminator can be obtained that can identify normal user features and therefore identify anomalies.

### 2.2.2 Anomaly edge detection

The advantage of dynamic graphs is that changes in the graph structure over time can be recorded so that the features of the edge distribution are captured for each timestamp. Also, this paper focuses on anomaly edge detection.

Statistics-based algorithms, network representation-based algorithms and GCN-based algorithms are the three popular algorithms. Statistical-based algorithms use specially designed statistical metrics to detect anomalous edges on dynamic graphs. Eswaran et al.[8] rely on dynamic graph edge change and graph structure as well as structural evolutionary pattern to construct model. Then, they define two kinds of anomalous edges: edges connecting disconnected regions of the graph; and edges appearing suddenly. Chang et al.[4] propose a frequency decomposition algorithm F-fade to discover anomalous input edges based on the observed frequencies. Specifically, F-fade combines the advantages of probabilistic models and matrix decomposition for capturing the temporal and structural changes. The NetWalk mentioned in 2.2.1 can also be applied to anomaly edge detection in dynamic graphs and uses node embedding to encode edges into a shared latent space and then identifies anomalies based on their distance to the center of the nearest edge cluster in the latent space. Actually, the edge representation generated by NetWalk is the Hadamard product of the source and target node representations. When a new edge arrives or an existing edge disappears, the node and edge representations are updated based on a temporary graph for each timestamp, and then the edge cluster centers and edge anomaly scores are recomputeda.

Although NetWalk is able to detect anomalies in dynamic graphs, it only updates the edge representation without considering long and short term evolutionary patterns of nodes and graph structures. The GCN-based approaches are popular in recent related researches, and the basic idea of GCN is to build embedding vectors based on graph structure. Zheng et al.[34] combine temporal, structural and attribute information to measure the anomalies of edges in dynamic graphs. They propose a semi-supervised model AddGraph, which consists of GCN and GRU, capturing representative structural information from each time-stamped graph and the dependencies between them, respectively. At each timestamp $t$, GCN produces the embedding of the node using the hidden state of the previous timestamp, then GRU learns the current hidden state from the node embedding and combines it with the previous hidden state. After the hidden status of all nodes is known, AddGraph assigns an anomaly score to each edge in the dynamic graph based on the nodes associated with it. The AddGraph framework is shown in the figure 2.1a. In addition, StrGNN[3] extracts the h-hop enclosing sub- graph of edges and leverages stacked GCN and GRU to capture the spatial and temporal information. The learning model is trained in an end-to-end way with negative sampling from "context-dependent" noise distribution. The StrGNN framework is shown in the figure 2.1b. H-VGRAE[32] builds a hierarchical model by combining variational graph auto-encoder and recurrent neural network. To detect anomalous edges, the edge reconstruction probability is used to measure the abnormality.



(a) AddGraph framework (b) StrGNN framework

Figure 2.1: AddGraph and StrGNN

TADDY[17], the basis of our approach, applies the Transformer model to the dynamic graph. TADDY framework can be categorized into the deep learning-based methods but has two main differences compared to the existing approaches mentioned above. Most of the above approaches employ different network modules to separately extract structural and temporal features, while TADDY uses a transformer network to model structural and temporal information simultaneously. Secondly, other methods consider naive node encoding from unattributed dynamic graphs as the network input, which may fail to provide sufficient information for the downstream neural network. In contrast to them, TADDY constructs a comprehensive node encoding that includes both structural and temporal information.

### 2.2.3 Anomaly subgraph detection

The detection methods for anomalous subgraphs include two main categories: statistical-based algorithms and deep learning-based algorithms. The idea of the former is to extract the special statistical metric of the anomaly subgraphs, while the idea of the latter is to use deep neural networks to extract features.

Chen et al.[5] introduce six statistical indicators to identify community anomalies, namely grown communities, shrunken communities, merged communities, split communities, born communities, and vanished communities. Inspired by the phenomenon that social media spam and fraudulent groups often form time-intensive subgraphs in online social networks, Bogdanov et al.[19] use manually extracted features and find unusually dense subgraphs that evolve significantly differently from the other part of the graph. In addition, the GBGP[13] uses boosted average scan statistics to identify nodes that may form anomalous subgraphs and detect suspicious groups that follow predefined irregular structures. Although these artificially crafted metric or statistical patterns are well suited for some specific types of existing anomalies, these methods are strongly limited in their ability to detect invisible and disguised anomalies, and direct application of these metric or statistical patterns may result in a high false positive rate.

FraudNE[35], a deep learning-based algorithm, models the online comment network as a bipartite graph and further detects anomalous users and associated manipulated items based on the principle of dense block detection. FraudNE uses two traditional auto-encoders to learn the user representation and the item representation, respectively. Both auto-encoders are trained to jointly minimize their corresponding reconstruction loss and shared loss functions. The shared loss function is proposed to restrict the representation learning process such that each linked user and item pair obtains a similar representation.

# Chapter 3

# Preliminaries

In this chapter, we introduce the three techniques and background knowledge used in this thesis. First of all, the acquisition of all features relies on the connectivity of the graph. Hence, in the phase of capturing structural features, we use diffusion technique to get adjacency matrix. Next, the dynamic graph transformer can simultaneously capture both structural and temporal features with a single encoder. Then, it is essential to design a mechanism to generate anomalies because of the lack of sufficient anomaly data. We solve this problem using spectral clustering technique. At the end of this chapter, we introduce the evaluation criteria that is used to measure the performance of algorithms.

## 3.1 Diffusion Technique

Diffusion matrix offers global perspectives of a graph's structure, allowing us to measure the significance of each node for a certain target node or edge. Diffusion is formulated as Eq.3.1, where $\mathbf{T} \in \mathbb{R}^{n \times n}$ is the generalized transition matrix and $\Theta$ is the weighting coefficient which determines the ratio of global-local information. Imposing $\sum_{k=0}^{\infty} \theta_k = 1, \theta_k \in [0,1]$, and $\lambda_i \in [0,1]$ where $\lambda_i$ is the eigenvalue of $\mathbf{T}$, guarantees convergence.

$$\mathbf{S} = \sum_{k=0}^{\infty} \Theta_k \mathbf{T}^k \in \mathbb{R}^{n \times n}. \tag{3.1}$$

Given an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and a diagonal degree matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$, different instantiations of graph diffusion can be computed. Personalized PageRank (PPR)[21] and heat kernel[15] are defined by setting $\mathbf{T} = \mathbf{AD^{-1}}$, and $\theta_k = \alpha(1-\alpha)^k$ and $\theta_k = e^{-t}t^k/k!$, respectively, where $\alpha \in (0,1)$ denotes teleport probability in a random walk and $t$ is diffusion time. In order to eliminate several iteration steps, the solutions to PPR and heat kernel can be expressed using Eq.3.2 and Eq.3.3, respectively.

$$\mathbf{S}^{\mathrm{PPR}} = \alpha \left(\mathbf{I}_n - (1-\alpha)\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\right)^{-1}, \tag{3.2}$$

$$\mathbf{S}^{\text{heat}} \ = \exp\left(t\mathbf{A}\mathbf{D}^{-1} - t\right). \tag{3.3}$$

## 3.2 Transformer

Since the Attention mechanism is proposed, the Seq2seq(Sequence to Sequence) model with attention joined has been improved on each task, so now the seq2seq model means the model combining RNN(Recurrent Neural Network) and Attention. Then google proposes the Transformer model, which replaces LSTM(Long Short-Term Memory) with a full-attention structure and achieves better results on translation tasks. Like most seq2seq models, the structure of the transformer is composed of encoder and decoder.

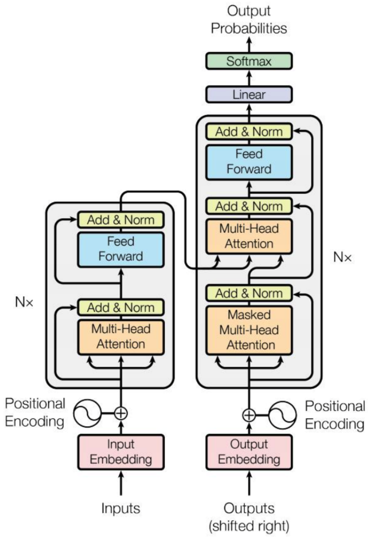


Figure 3.1: Transformer Structure. Original illustration by [28]

### 3.2.1 Encoder

Encoder consists of six identical layers, which are the units of the left side in Fig.3.1[28]. Each layer consists of two sub-layers, the multi-head attention and the fully connected feed-forward network, respectively. Each sub-layer also includes Add & Norm layers, where Add is a Residual Connection to prevent network degradation and Norm is Layer Normalization, which normalizes the activation values of each layer. Each sub-layer output is calculated as Eq.3.4, where SubLayer$(x)$ is the result of sub-layer. Next, we describe the two sub-layers.

$$\text{sub\_layer output } = \text{ LayerNorm } (x + (\text{ SubLayer } (x))). \tag{3.4}$$

**Multi-Head Attention.** The attention mechanism, in fact, is to calculate the relevance, is to pay more attention to those things that are more relevant. For each input vector $\boldsymbol{X}_{embedding}$, the model first computes three new vectors Query(Q), Key(K), Value(V), and then computes a relevance score. Q, K, V are formulated in Eq.3.5 where $W$ is the weight matrix. Self-attention is formulated in Eq.3.6 where $d_k$ is the dimension of the vector, and is shown in Fig.3.2(a).

$$\begin{aligned} Q &= X_{\text{embedding }} * W_Q, \\ K &= X_{\text{embedding }} * W_K, \\ V &= X_{\text{embedding }} * W_V, \end{aligned} \tag{3.5}$$

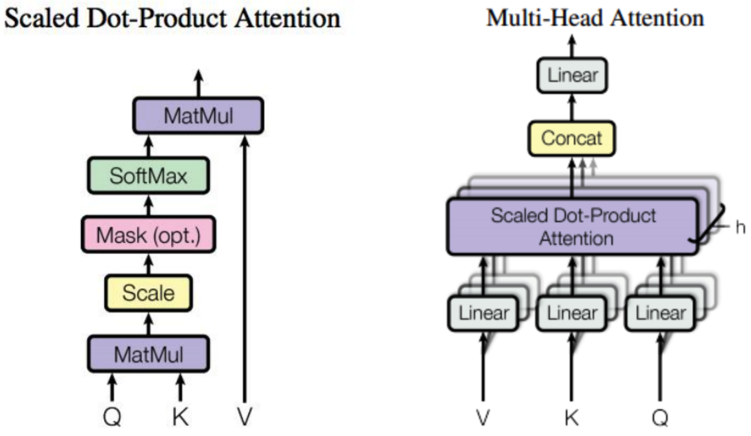
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{3.6}$$

Multi-Head Attention is formed by the combination of multiple Self-Attention. As shown in Fig.3.2(b), multi-head attention is the projection of Q, K, V by $h$ different linear transformations, and finally the different attention results are concatenated. Multi-head attention is formulated in Eq.3.7.

$$\begin{aligned} \text{MultiHead } (Q, K, V) &= \text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right) W^O, \\ \text{head}_i = &\text{ Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right). \end{aligned} \tag{3.7}$$

**Feed Forward Network.** Feed forward network(FFN) is actually a two-layer network, where the activation function of the first layer is ReLU. For a vector $x$ at each position in the input sequence:

$$\mathrm{FFN}(x) = \max\left(0, xW_1 + b_1\right) W_2 + b_2.$$



(a) Self-Attention (b) Multi-Head Attention

Figure 3.2: Self-Attention and Multi-Head Attention. Original illustration by [28]

### 3.2.2 Decoder

The decoder consists of six identical network layers stacked together. For each decoder layer, the structure is similar to that of the encoder, but decoder has an additional multi-head attention mechanism for interacting with encoder output, as shown in figure3.1. There are 3 parts in decoder. The top feed forward network and the bottom multi-head attention are the similar to encoder when masked is not considered, except that the middle part, which interacts with the encoder output. For this part, $Q$ is the output of the following multi-head attention, and both $K$ and $V$ are the output of the encoder. Masked multi-head attention means the multi-head attention receives inputs with masks so that the attention mechanism does not use information from the hidden(masked) positions. The softmax within the attention mechanisms effectively assigns zero probability to masked positions.

Decoder performs differently in training and in prediction. In the training phase, the decoder works like an encoder, calculating the input for all moments at once. The advantages of this design are that firstly, the training speed can be accelerated by parallel computation of multiple samples; secondly, the decoder's correct result can be input directly during the training phase instead of the prediction of the previous moment, which may be wrong during the training. In the prediction phase, when decoder decodes the content at the current moment, decoder takes all the predicted results before the current moment as input to predict the output of the next moment.

### 3.2.3 Position Encoding

Since the Transformer's architecture loses the positional information of the input sequence, the relative or absolute positional information of each token in the sequence is introduced into the whole training phase. The position encoding is added at the beginning of all encoder and decoder layers.

Although the attention mask has the ability to feed the sequence into the decoder moment by moment, it cannot recognize the inherent order of the sequence itself. In Transformer, the model generates the positional information by Eq.3.8, where $PE$ is this positional embedding matrix, $pos \in [0, max\_len)$ denotes a specific position, and $i \in [0, d_{\text{model}} /2)$ denotes a dimension. By adding the input embedding and the positional embedding, model can get the representation vector and this vector is the input of the Transformer.

$$\begin{aligned} PE_{\text{pos },2i} &= \sin\left(pos/10000^{2i/d_{\text{model}}}\right), \\ PE_{pos,2i+1} &= \cos\left(pos/10000^{2i/d_{\text{model}}}\right). \end{aligned} \tag{3.8}$$

## 3.3 Spectral Clustering

Spectral clustering is a widely used clustering algorithm. Compared with the traditional clustering algorithm K-Means, spectral clustering is more adaptable to data distribution and has excellent clustering effects, while being much less computationally intensive. Spectral clustering is an algorithm that evolved from graph theory. The main idea of spectral clustering is to consider all the data as nodes in a space, and these nodes can be connected to each other by edges. The edge weight value between two distant nodes is low, while the edge weight value between two closer nodes is high. In this way, the nodes are grouped into multiple communities and thus clustering is achieved.

In general, the main concerns of spectral clustering are the way to generate the similarity matrix, the way to cut the graph, and the final clustering method. The most commonly used similarity matrix generation method is the fully connected method based on Gaussian kernel distance, the most commonly used cut method is Ncut, and the last clustering method is k-means.

**Degree matrix.** For a graph $G$, we generally describe it by the node set $V$ and the edge set $E$, that is, $G(V, E)$ where $V$ is all the nodes $(v_1, v_2, \dots v_n)$. For any two nodes, they can be connected with or without edges. We define the $w_{ij}$ as the weight between $v_i$ and $v_j$. Since we research undirected graphs, $w_{ij} = w_{ji}$ if edge $e_{ij}$ exits, otherwise $w_{ij} = 0$. For any node $v_i$, its degree $d_i$ is the sum of the weights of all the edges linked to it, $d_i = \Sigma_{j-1}^{n} w_{ij}$. Relying on the definition of the degree, we can obtain a $n * n$ degree matrix $D$, which has values only on the main diagonal, corresponding to the degree of $v_i$.

$$D = \begin{bmatrix} d_1 & \dots & \cdots \\ \cdots & d_2 & \cdots \\ \vdots & \vdots & \ddots \\ \cdots & \cdots & d_n \end{bmatrix}$$

**Adjacency matrix.** Based on the weights between all nodes, we can obtain the adjacency matrix $W$, which is also a $n * n$ matrix where the value of the i-th row and j-th column correspond to weight $w_{ij}$. We select the full connected method, where the weight between all the nodes is greater than 0. Here, we use Gaussian kernel function to generate weights in Eq.3.9.

$$w_{ij} = \exp\left(-\frac{\|x_i - x_j\|_2^2}{2\sigma^2}\right). \tag{3.9}$$

**Laplace matrix.** Next, we compute the graph Laplacian. The Laplacian is just another matrix representation of the graph. It has several beautiful properties that help to perform spectral clustering. To calculate the normal Laplacian, we just subtract the adjacency matrix $W$ from the degree matrix $D$: $L = D - W$. The diagonal of the Laplacian matrix is the degree of the node, while the off diagonal is the negative edge weights. This is the representation we need when performing spectral clustering.

**NCut.** Our goal is to cut the graph $G(V,E)$ into $k$ disjoint subgraphs. The nodes set for each subgraph is $A_1, A_2, ..A_k$. They satisfy $A_i \cap A_j = \emptyset$ and $A_1 \cup A_2 \cup \ldots \cup A_k = V$. For any two subgraph node sets $A, B \subset V$, $A \cap B = \emptyset$ , we define the weight of the cut between $A$ and $B$ as:

$$W(A,B) = \Sigma_{i \in A, j \in B} w_{ij}.$$

Then for the $k$ subgraph node sets $A_1, A_2, ..A_k$, the cut function is formulated in Eq.3.10

$$\operatorname{cut}(A_1, A_2, \ldots A_k) = \frac{1}{2}\Sigma_{i=1}^{k} W\left(A_i, \bar{A}_i\right). \tag{3.10}$$

In order to make the node weights large within the subgraph and small between the subgraphs, a straightforward idea is to minimize the $\operatorname{cut}(A_1, A_2, \ldots A_k)$, but there may be problems. As shown in Fig.3.3, although such segmentation minimizes the $\operatorname{cut}(A_1, A_2, \ldots A_k)$, many individual discrete nodes are partitioned as a class, and the partitioning is not uniform.

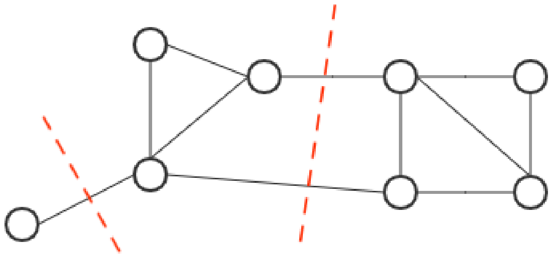


Figure 3.3: Cut

To avoid the aforementioned problem, we need to limit the size of each subgraph, and use the NCunt algorithm to cut the graph. When cutting the graph, the result of cutting not only minimizes the $\operatorname{cut}(A_1, A_2, \ldots A_k)$, but also maximizes the weight of each subgraph. Then, we need to minimize the NCut function 3.11 where $\operatorname{vol}(A_i)$ is the sum of all edge weights in $A_i$. In this way, we find $k$ smallest features in $L$, and then we can obtain a $n * k$ feature matrix. Since dimensionality reduction may result in information loss, it is usually necessary to perform a traditional clustering, like k-means, after obtaining the $n * k$ matrix.

$$\operatorname{NCut}(A_1, A_2, \ldots A_k) = \frac{1}{2}\Sigma_{i=1}^{k} \frac{W\left(A_i, \bar{A}_i\right)}{\operatorname{vol}(A_i)}. \tag{3.11}$$

**Algorithm 1** Spectral Clustering

```
Input: dataset $D = (x_1, x_2, \ldots, x_n)$, number k of clusters;
Output: k clusters $C(c_1, c_2, \ldots c_k)$;
Compute adjacency matrix $W$ and degree matrix $D$;
Compute the Laplacian matrix $L = D - W$;
Normalize Laplacian matrix $D^{-1/2}LD^{-1/2}$;
Compute the first k eigenvectors $u_1, u_2, ...u_k$ of $D^{-1/2}LD^{-1/2}$;
Let $U \in R^{n\times k}$ be the matrix containing the vectors $u_1, u_2, \ldots, u_k$, $U = [u_1, u_2, \ldots, u_k] \in \mathrm{R}^{n\times k}$;
Form the matrix $T \in \mathrm{R}^{n\times k}$ from $U$ by normalizing the rows to norm 1, set $t_{ij} = u_{ij}/\sqrt{\sum_{m=1}^{n} u_{im}^2}$;
For $i = 1, 2, \ldots, n$, let $\mathbf{y}_i \in \mathrm{R}^k$ be the vector corresponding to the $i$-th row of $T$;
Cluster the $\{\mathbf{y}_i\}_{i=1,\ldots,n}$ in $\mathrm{R}^k$ with k-means algorithm into clusters $\mathrm{c}_1,\ \mathrm{c}_2, \ldots, \mathrm{c}_k$;
return $C(c_1, c_2, \ldots c_k)$;
```

## 3.4 Evaluation Criteria

Measuring performance is vital while evaluating and analyzing the results of our method. A standardized metric is required for evaluating and comparing the performance of these anomaly detection methods. In this thesis, we apply Area Under the Curve (AUC) of Receiver Operating Characteristics (ROC) curve. Before explaining this metric, we shall describe some parameters.

- **TP:** known as true positive, the number of predictions that are correctly classified as positive class.
- **TN:** known as true negative, the number of predictions that are correctly classified as positive class.
- **FP:** known as false positive, the number of predictions that are wrongly classified as positive class but belong to the negative class.
- **FN:** known as false negative, the number of predictions that are wrongly classified as positive class but belong to the positive class.
- **TPR:** the proportion of the positive class that is correctly classified, known as true positive rate and defined as:
$$TPR = \frac{TP}{TP + FN}.$$
- **FPR:** the proportion of the positive class that is wrongly classified by the model, known as false positive rate and defined as:
$$FPR = \frac{FP}{FP + TN}.$$

For calculation, we define the positive class as the outliers and the rest are the negative class. According to the definition, AUC is the area under the ROC curve, and the x-axis of the ROC curve is FPR while the y-axis is TPR, representing a comprehensive analysis of how the model tradeoff between TP and FP. An example can be seen in Fig.3.4

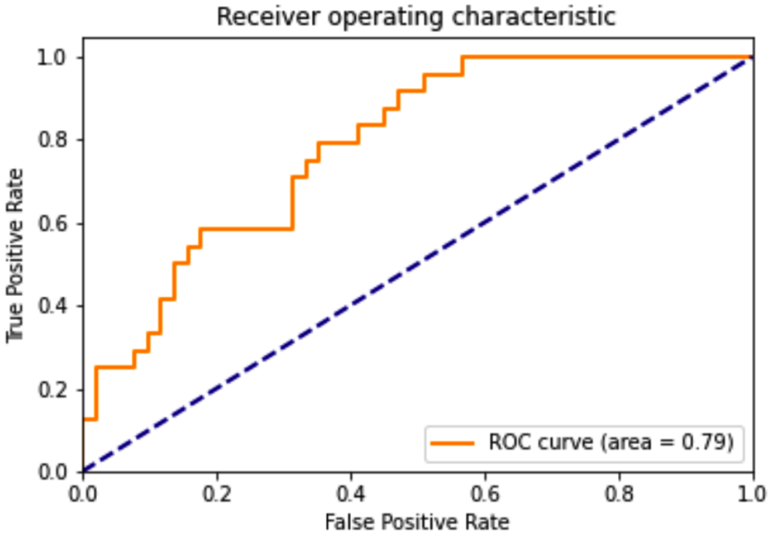


Figure 3.4: ROC

**AUC score:** This score ranges from 0 to 1 and good classifiers should achieve higher scores. A completely random classifier is expected to get a half score, which is 0.5. Thus if a score is below 0.5, the classifier could simply reverse its prediction to get a score above 0.5. This calculation method of AUC takes into account the classification ability of the classifier for both positive and negative cases, and is able to make a reasonable evaluation of the classifier in the case of class imbalance.

# Chapter 4

# Proposed Anomaly Detection Approach

In this chapter, the anomaly detection algorithms developed in our research are described in detail. In general, the whole framework consists of four parts, edge related nodes sampling, edge structure and temporal encoding, dynamic graph transformer, anomaly detector, respectively. The framework is end-to-end and outputs anomaly scores directly. First, in order to capture the structural and temporal features of the target edges, we need to capture the structural and temporal context of the target edges. Our method samples around the target edge to acquire the relevant nodes at multiple timestamps. Next, edge structure and temporal encoding module generates the encoding for the edge as input to the transformer. In this module, both the structure and temporal information of each edge are integrated into a fixed-length encoding. Then, the dynamic graph transformer extracts the structural and temporal knowledge of the edges. Finally, anomaly detector performs a negative data generation strategy to generate negative edges, and then an edge scoring module trained by binary cross-entropy loss is employed to calculate the anomaly scores. At the end of this chapter, we compare the differences between our framework and TADDY[17], on which our work is based. After that, we analyze the time complexity.

## 4.1 Edge Related Nodes Sampling

As stated in prior works[3, 16], anomalies frequently appear in local substructure of the graph, suggesting that we should expand the area of investigation to an appropriate scope. Therefore, we should sample the pertinent nodes of the target edges as research data for our anomaly detection framework rather than investigating them on a series of dynamic graphs. Since we concentrate on detecting anomalous edges and sample nodes surrounding a certain edge in a dynamic graph, this edge is deemed the target edge. For a given target edge, we denote the source and destination nodes as end nodes.

Besides the target node, some neighboring nodes must be sampled to construct a substructure. In this thesis, we define these neighbor nodes as contextual nodes. To obtain contextual nodes, we first address the sampling problem in a single snapshot, which is a static graph. Commonly, the $h$-hop neighbors of end nodes are extracted as contexts. Yet, this method has a number of disadvantages. First of all, the unbalanced distribution of node degrees in real-world datasets causes performance degradation and inefficiency when utilizing h-hop sampling. For instance, the UCI message dataset has an average degree of 14.47 and a maximum degree of 255. For nodes with a high degree, the number of h-hop neighbors will increase exponentially, resulting in a great deal of noisy data in the sampling results. Secondly, The h-hop sampling method disregards the various roles and importance of contextual nodes. When two end nodes have both shared and exclusive neighbor nodes, it is evident that the shared neighbor node contributes significantly more to identifying the target edge. The h-hop sampling strategy treats shared and exclusive neighbor nodes identically when sampling contextual nodes.

To address the aforementioned shortcomings, we employ the graph diffusion technique mentioned in Section 3.1 to extract a fixed-size and importance-aware contextual node set for each target edge. We generate the diffusion matrices from the graphs using the graph diffusion technique. Given a diffusion matrix $S$, the i-th row $s_i$ represents the connectivity between the i-th node and all other nodes from a global view. The value of $s_{i,j}$, for instance, represents the strength of the connection between the $i$-th node and $j$-th node. Accordingly, we are able to select a fixed number of the most significant relevant nodes for a given target edge. Taking edge $e_{\text{target}} = (v_1, v_2)$ as an example, we can calculate the connection vector of $e_{\text{target}}$ by summing the connection vector of target edge's two end nodes by Eq.4.1.

$$\mathbf{s}_{e_{\text{target}}} = \mathbf{s}_{v_1} + \mathbf{s}_{v_2}. \tag{4.1}$$

Subsequently, the connectivity vector $\mathbf{s}_{e_{\text{target}}}$ is sorted, and the k nodes with the largest values are picked to construct the contextual nodes set $\mathcal{V}(e_{\text{target}})$. The end nodes should be excluded when selecting the top-k connectivity nodes. Finally, the set of sampled nodes is the union of the set of contextual nodes and the end nodes, which can be denoted as $\mathcal{S}(e_{\text{target}}) = \{v_1, v_2, \mathcal{V}(e_{\text{target}})\}$. As shown in Fig.4.1, the end nodes(red) and contextual nodes(green) from multiple timestamps are sampled to construct the sampled node set, where neighboring node number $k$ and time window size $\tau$ are both 3.

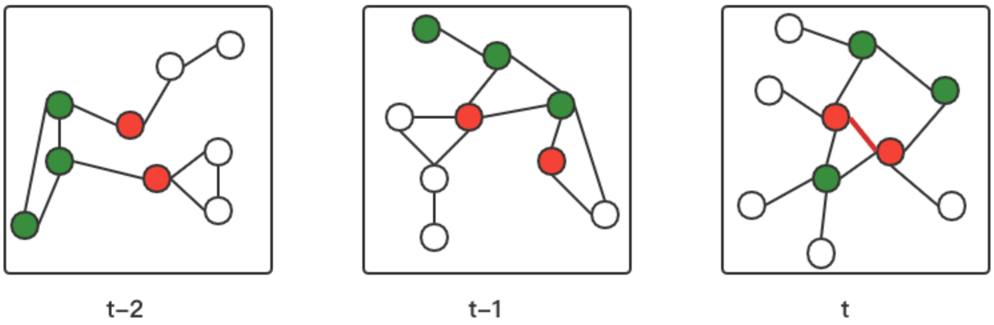


Figure 4.1: Sampling Example. Original illustration by [17]

We can filter contextual nodes from a single static graph using a diffusion-based sampling technique. To capture dynamic changes in dynamic graphs, however, several timestamp graphs need be considered. The sampling method is extended to dynamic graphs. Given a target edge $e^t_{\text{target}} = (v^t_1, v^t_2)$ at timestamp t, and we consider a sequence of graphs $\mathbb{G}^t_\tau = \{\mathcal{G}^{t-\tau+1}, \cdots, \mathcal{G}^t\}$, where the time window size $\tau$ is a hyperparameter that determines the range of this sequence of graphs on the time line. Through the sliding window mechanism, our method is able to capture the dynamic evolution between the timestamps $(t - \tau + 1)$ and $(t)$. Then, for each $\mathcal{G}^i \in \mathbb{G}^t_\tau$, we calculate the diffusion matrix $S^i$ and get the corresponding connection vector $s^i_{e_{target}}$. By picking the top k nodes and merging them with the target edge end nodes, the sampled node set of target edge at the $i$-th timestamp is $\mathcal{S}^i(e^t_{\text{target}})$. By combining the set of nodes from multiple timestamps, we can obtain the final sampled nodes set by Eq.4.2.

$$\mathcal{S}(e^t_{\text{target}}) = \bigcup_{i=t-\tau+1}^{t} \mathcal{S}^i(e^t_{\text{target}}). \tag{4.2}$$

**Algorithm 2** Edge Related Nodes Sampling

```
Input: Graph stream $\mathbb{G} = \left\{\mathcal{G}^{t-\tau+1}, \cdots, \mathcal{G}^{t}\right\}$, time window size: $\tau$
Output: Sampled node set $\mathbf{X}$ of all edges;
for each timestamped graph $\mathcal{G}^i = \left(\mathcal{V}^i, \mathcal{E}^i\right) \in \mathbb{G}$;
    Compute the diffusion matrix $\mathbf{S}^i$;
    for each edge $e_j$ in $\mathcal{E}^i$;
        Compute the $e_j$'s connectivity vector $\mathbf{s}^i_{e_j}$;
        United top-k connectivity nodes and end nodes as sampled set $\mathcal{S}\left(e_{\mathrm{j}}\right)$;
    Get $C^i(\mathcal{S}\left(e_0\right), \mathcal{S}\left(e_1\right), \cdots, \mathcal{S}\left(e_{\mathrm{j}}\right))$;
Get $\mathbf{X}\left(C^0, C^1, \cdots, C^i\right)$
return $\mathbf{X}$;
```

## 4.2 Structure and Temporal Encoding

For image data, each data instance, such as an image patch, has its own characteristics, but most dynamic graphs lack attributes. Hence, raw graph data cannot be used as input. Graph data are numerical representations of edge endpoints. Therefore, we generate the encoding or embedding from the dynamic graph as input. The available solution is to employ edge identity encoding as the original edge attribute, that is, each edge has its own unique one-hot encoding. However, edge identity encoding in this situation has some restrictions. Firstly, one-hot encoding is incapable of storing adequate structural and temporal information. The one-hot encoding can identify an edge's identification, but it is challenging to define its structure and temporal state. Secondly, edge identity encoding is inappropriate for big and dynamically changing graphs. Thirdly, one-hot encoding with fixed dimensions is unsuitable for dynamic graphs as the number of edges and nodes is always changing.

We introduce structural and temporal edge encoding for dynamic graph transformer model. The edge encoding consists of three components, diffusion-based structure encoding, importance structure encoding and normalized relative temporal encoding, respectively. The two structure encodings describe the structure of each edge from the global and local perspective. In terms of temporal encoding, it provides the temporal information of each edge. In the end, these three encodings are combined into an input encoding that contains comprehensive structural and temporal information.

In the rest of this chapter, we discuss these three encodings sequentially and the encoding merge operation.

### 4.2.1 Diffusion-based Structure Encoding

As introduced in Section 3.1, the graph diffusion provides a global view of the structural role for each node. The connection strength between the target edge and the contextual nodes can be easily obtained by the edge connection vector $\mathbf{s}_{e_{\mathrm{target}}}$ calculated by Eq.4.1. That is why we design a structure encoding based on graph diffusion. To prevent similar encoding caused by similar diffusion values, we use the sorted diffusion values as encoding rather than the original diffusion values. Specifically, for each node in the single timestamp sampled node set $v^i_j \in \mathcal{S}^i\left(e^t_{\mathrm{target}}\right)$, we sort the nodes according to their diffusion values and use the sorted results as the data source. Based on the sorted results, we compute the diffusion-based structure encoding. The algorithm to generate the diffusion-based structure encoding is described as Algorithm 3.

**Algorithm 3** Diffusion-based Structure Encoding

```
Input: Sampled node set $\mathbf{X}$;
Output: Diffusion-based structural encoding $\mathbf{x}_{\text{global}}$;
for each edge's sampled set;
    Sort the nodes of sampled set by connectivity strength;
Encode the sampled node set;
Get edge encoding $\mathbf{x}_{\text{global}}$;
return $\mathbf{x}_{\text{global}}$;
```

### 4.2.2 Importance Structure Encoding

Even though diffusion-based structure encoding captures global structural information, the local impact of each contextual node in the sampled node set must also be considered. The transformer model does not take the graph structure, such as the adjacency matrix, as input like GNN, thus we design an importance-based structure encoding to express the local information surrounding the target edges. Specifically, for each node in the single timestamp sampled node set $v_j^i \in \mathcal{S}^i \left(e_{\text{target}}^t\right)$, we utilize the distance between the context node and the target edge as a measure of this node's importance to the target edge. There is a distance between the context node and each of the two end nodes, and the shortest of these two distances is regarded as the context node's importance to the target edge. For the end nodes themselves, the distances are denoted as 0. Encoding is output from a single linear layer. In detail, the importance of the structure encoding algorithm is shown as Algorithm 4.

**Algorithm 4** Importance Structure Encoding

```
Input: Sampled node set $\mathbf{X}$;
Output: Importance structure encoding $\mathbf{x}_{\text{local}}$;
for each edge $e^i(v_1^i, v_2^i)$'s sampled set;
    for node $v_j$ in sampled set;
        Calculate $d_j = min(distance((v_1^i, v_j), distance(v_2^i, v_j))$
    Get $D^i(d_0, d_1, \cdots, d_j)$;
Encode $D^i$;
Get encoding $\mathbf{x}_{\text{local}}$;
return $\mathbf{x}_{\text{local}}$;
```

### 4.2.3 Normalized Relative Temporal Encoding

The temporal encoding is used to convey time information for the target edge. For the time representation method, we select the relative time encoding of dynamic graphs. Specifically, for each node in the sampled node set $v_j^i \in \mathcal{S}^i \left(e_{\text{target}}^t\right)$, the relative time encoding is defined as the difference between the occurring time $t$ of target edge and the current time of timestamp $i$. As the primary objective of anomaly detection is to forecast whether the target edge is anomalous, the time relative to the target edge is a more significant factor. To reduce the effect of relative time scale on the encoding, temporal encoding should ideally be limited to a value range. Thus, we normalize the temporal encoding using a time window. In the end, temporal encoding is calculated by a single linear layer. The algorithm for normalized relative temporal encoding is shown as Algorithm 5.

**Algorithm 5** Normalized Relative Temporal Encoding

```
Input: Graph stream $\mathbb{G}$, occurrence time $t$ and time window size $\tau$;
Output: Temporal encoding $\mathbf{x}_{\text{temporal}}$;
for each timestamped graph $\mathcal{G}^i$;
    Compute $T^i = \frac{\|i-t\|}{\tau-1}$;
Encode $T^i$;
Get encoding $\mathbf{x}_{\text{temporal}}$;
return $\mathbf{x}_{\text{temporal}}$;
```

### 4.2.4 Encoding Merge

As shown in Fig.4.2, after computing these three encodings, we merge them into the input edge encoding of the downstream transformer model. In order to increase running efficiency, we define the edge-merge encoding as the sum of three encodings rather than concatenating them into a vector with a larger dimension. The encoding merge is formalized as Eq.4.3

$$\mathbf{x}\left(e_{target}^{t}\right) = \mathbf{x}_{\text{global}}\left(e_{target}^{t}\right) + \mathbf{x}_{\text{local}}\left(e_{target}^{t}\right) + \mathbf{x}_{\text{temp}}\left(e_{target}^{t}\right). \tag{4.3}$$

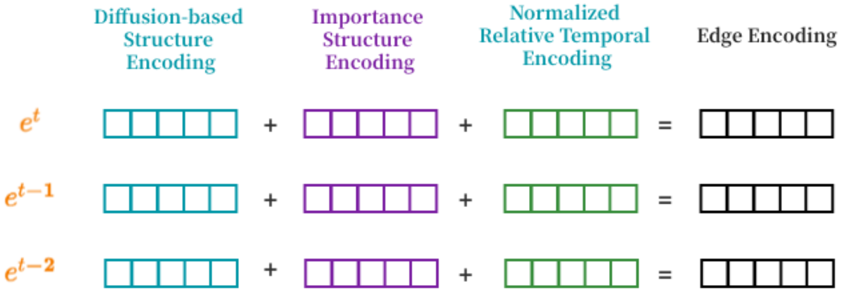

Diffusion-based Structure Encoding
Importance Structure Encoding
Normalized Relative Temporal Encoding
Edge Encoding
$e^t$
$e^{t-1}$
$e^{t-2}$
+
+
=

Figure 4.2: Edge Encoding

## 4.3 Dynamic Graph Transformer

To acquire knowledge from dynamic graphs, neural network models need to take both structural and temporal information into account. In most instances, structural and temporal data are combined. Therefore, for a dynamic graph encoder, we should concentrate on how to develop an encoder that can concurrently take structure and temporal information into account. In the majority of known works, the structural and temporal modules are stacked. The structural and temporal modules independently extract structural and temporal features. For example, in the AddGraph framework, the GCN is the structure module and GRU processes the GCN output with different timestamps to obtain temporal information. A limitation of such architectures is that they may miss some joint structural and temporal information, which may further lead to suboptimal solutions.

To learn the structure and temporal knowledge in the dynamic graphs, we apply the transformer model as an encoder only. Taking edge encoding from multiple timestamps as input, the dynamic graph transformer can capture structural and temporal features with a single encoder.

The goal of the transformer module is to generate target edge embeddings that fuse structural and temporal information. To this end, a number of attention layers are utilized to exchange the information of different embeddings. To be concrete, a single attention layer can be written as Eq.4.4.

$$\mathbf{H}^{(l)} = \text{attention}\left(\mathbf{H}^{(l-1)}\right) = \text{softmax}\left(\frac{\mathbf{Q}^{(l)}\mathbf{K}^{(l)\top}}{\sqrt{d_{emb}}}\right)\mathbf{V}^{(l)}, \tag{4.4}$$

where $\mathbf{H}^{(l)}$ and $\mathbf{H}^{(l-1)}$ is the output embedding of the $l$-th and $(l-1)$-th layer, $d_{emb}$ is the dimension of node embedding, and $\mathbf{Q}^{(l)}, \mathbf{K}^{(l)}, \mathbf{V}^{(l)}$ are the query matrix, key matrix and value matrix for feature transformation and information exchange. Concretely, the $\mathbf{Q}^{(l)}, \mathbf{K}^{(l)}$ and $\mathbf{V}^{(l)}$ are computed by Eq.4.5.

$$\begin{cases} \mathbf{Q}^{(l)} = \mathbf{H}^{(l-1)}\mathbf{W}_Q^{(l)}, \\ \mathbf{K}^{(l)} = \mathbf{H}^{(l-1)}\mathbf{W}_K^{(l)}, \\ \mathbf{V}^{(l)} = \mathbf{H}^{(l-1)}\mathbf{W}_V^{(l)}, \end{cases} \tag{4.5}$$

where $\mathbf{W}_Q^{(l)}, \mathbf{W}_K^{(l)}, \mathbf{W}_V^{(l)}$ are the learnable parameter matrices of the $l$-th attention layer. In an attention layer, $\mathbf{Q}^{(l)}$ and $\mathbf{K}^{(l)}$ calculate the contributions of different edge embeddings, while $\mathbf{V}^{(l)}$ projects the input into a new feature space. Eq.4.4 combines them and acquires the output embedding of each edge by aggregating all structural and temporal information of the edge.

In our transformer module, the input of the transformer module $\mathbf{H}^{(0)}$ is defined as the encodings of the target edge $\mathbf{X}\left(e^t_{\text{target}}\right)$. Here, we set $d = d_{emb} = d_{enc}$ so that their dimensions are equal. The output of the final attention layer $\mathbf{H}^{(l)}$ is denoted as the output edge embedding.

## 4.4 Anomaly Detector

After getting the edge embeddings, the goal of anomaly detection is to calculate anomaly score for each edge in the dynamic graph. In this thesis, we consider an end-to-end framework where a neural network-based anomaly detector computes anomaly scores. Moreover, in the learning setting, there is not any ground truth anomaly sample in the training set. Such a situation brings a new problem of how to train the model without any given anomalous sample. Our solution is to generate pseudo anomalous edges via a negative sampling strategy and train the anomaly detector with the existing edges in the training set as well as the pseudo anomalous edges together.

A negative sampling strategy is performed in our framework. For the graph of each timestamp, we apply the spectral clustering mentioned in section 3.3 to the graph. Then, the nodes in the graph are clustered into multiple communities, and for most normal edges, both end nodes are clustered into the same community. We can randomly select some pairs of nodes as the end nodes of negative edge candidates, and the two end nodes are from different communities. After that, we check all these node pairs to ensure that they do not belong to the existing normal edge set, otherwise we resample a new pair and perform validation for each illegal node pair until the node pair is valid. After negative sampling, we use edge related nodes sampling to acquire the sampled node set of each negative edge and perform structure and temporal encoding. Then, the encoding is fed into the dynamic graph transformer model to obtain the embedding of the negative edge.

The anomaly detector is constructed to discriminate the positive and negative edge embeddings. A fully connected neural network layer with Sigmoid activation is served as the scoring module which computes the anomaly scores of edge embeddings, which is formalized by Eq.4.6.

$$f(e) = \text{Sigmoid}\left(\mathbf{z}(e)\mathbf{w}_S + b_S\right), \tag{4.6}$$

where $f(e)$ and $\mathbf{z}(e)$ is the anomaly score and edge embedding of edge $e$ respectively, Sigmoid is the activation function, $\mathbf{w}_S \in \mathbb{R}^{d_{emb}}$ and $b_S \in \mathbb{R}$ are the weights and bias parameters of fully connected neural network layer. In the training phase, we adopt a binary cross-entropy loss function. For the positive edges, we expect them to have a small anomaly score, hence their label is 0; for the negative edges, conversely, the label is 1. For the $\mathcal{G}^t = (\mathcal{V}^t, \mathcal{E}^t)$ whose edge number is $m^t$, the loss function is given as Eq.4.7. The workflow of the anomaly detector is shown in Fig.4.3.

$$\mathcal{L} = -\sum_{i=1}^{m^t} \log\left(1 - f\left(e_{\text{pos},i}\right)\right) + \log\left(f\left(e_{\text{neg},i}\right)\right), \tag{4.7}$$

where $e_{\text{pos},i} \in \mathcal{E}^t$ is the $i$-th positive edge and $e_{\text{neg},i} \in \mathcal{E}^t$ is the $i$-th negative edge sampled by the negative sampling strategy.

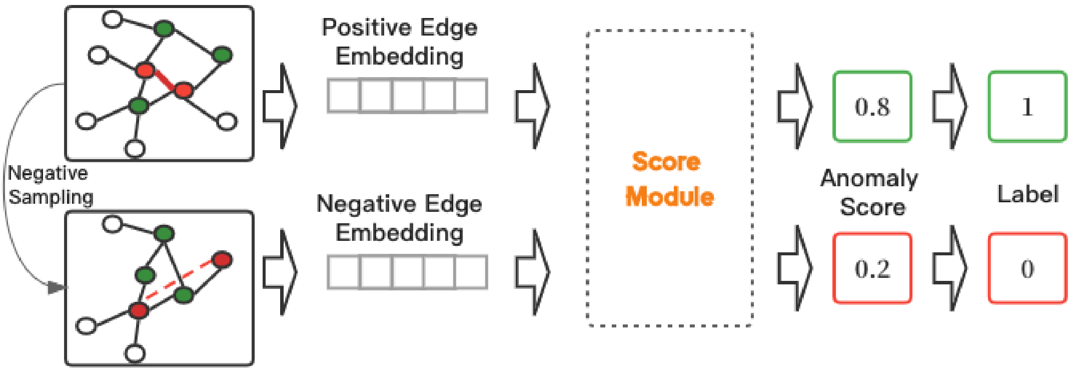


Figure 4.3: Anomaly Detector

To the end, the overall procedure of our framework is depicted in Algorithm 6. In each loop, different negative edges are sampled to prevent training bias and over-fitting. For all positive and negative edges, we perform edge related nodes sampling, structure and temporal encoding, transformer embedding and anomaly score computation sequentially. Finally, the parameters are updated by back propagation under the binary cross-entropy loss function.

**Algorithm 6** Overall Procedure

```
Input: Dynamic graph: $\mathbb{G} = \{\mathcal{G}^t\}_{t=1}^{T}$; Number of training epochs: $I$; Number of contextual
    nodes: $k$; Size of time window: $\tau$;
Output: Anomaly score $f(e)$;
Initialize the parameters of encoding function, transformer model and scoring function;
for epoch $i = 1, 2, \cdots, I$;
    for each timestamped graph $\mathcal{G}^t = (\mathcal{V}^t, \mathcal{E}^t) \in \mathbb{G}$;
        Sample negative edge set $\mathcal{E}^t_{neg}$ by negative sampling strategy;
        for $e \in \mathcal{E}^t \cup \mathcal{E}^t_{neg}$;
            Get $e$'s sampled node set $\mathcal{S}(e)$
            Calculate the edge encoding $\mathbf{x}(e)$
            Calculate the edge embedding $\mathbf{z}(e)$;
            Calculate anomaly score $f(e)$;
        Calculate loss function $\mathcal{L}$;
        Back propagation and update the parameters;
```

## 4.5 Differences

Because our framework is based on TADDY, the differences between our framework and TADDY are outlined as follows.

- Compared to the design in TADDY where the node encoding is obtained first and then the edge encoding is transformed by a module, in our algorithm design, feature engineering is computed on the edges and the edge encoding is obtained directly, which is more concise and ensures performance.
- For extracting graph structural features, the differences focus on global structural features. TADDY assigns a ranking number to the sampled nodes as a global structural feature based on the value calculated by the graph diffusion. However, the number of sampled nodes is five, and the global structure information consisting of only five numbers(1-5) can easily be repeated, due to the huge number of edges. To solve this problem, we assign a role to each node, specifically, a number to each node.
- For the temporal features, we normalize the relative time. Regarding the anomaly generation strategy, different from TADDY's random selection of two nodes to generate anomalous edges, we use the spectral clustering approach. The anomalies generated by spectral clustering more closely match the properties of real anomalies.

## 4.6 Complexity Analysis

In this section, we analyze the time complexity of each component. For edge related nodes sampling, the complexity is mainly caused by the computation of graph diffusion and the time complexity is $O\left(Tn^2\right)$, where $n$ is the average number of nodes for each timestamped graph and $T$ is the number of timestamps. In structure and temporal node encoding and dynamic graph transformer, we process all nodes in the sampled node set for each target edge, which brings a time complexity of $O(\tau k)$ for one edge, where $\tau$ is the time window and $k$ is the number of nodes in sampled node set. Therefore, for $I$ epochs, the total time complexity is $O(\tau kmI)$, where $m$ is the number of edges. For anomaly detector, the time complexity is $O(Im)$. To sum up, the overall time complexity is $O(Tn^2 + \tau kmI + Im)$.

# Chapter 5

# Experiments

In this section, we begin with the details of the datasets and experiment setup. Then we evaluate the performance of our framework compared to the baseline methods. Moreover, we discuss the performance of our framework. Importantly, how accurate is our framework in detecting anomalies in datasets with ground truth labels? How each edge encoding affects performance? How does accuracy depend on the hyper-parameters? We will answer these questions in the following.

## 5.1 Datasets

We use six real world datasets in our experiments to evaluate the performance of our framework compared to the baselines. All these datasets have ground labels, we can use the evaluation metric mentioned in section 3.4 to calculate the ROC/AUC of framework and then compare the performance. Table 5.1 shows the statistical information of the datasets. The detailed descriptions of the datasets are shown as follows:

- The UCI Messages dataset is collected from an online community platform of students at the University of California, Irvine. Each node in the constructed graph represents a user. And the edge indicates that there is a message interaction between two users.
- Bitcoin-Alpha and Bitcoin-OTC are two who-trusts-whom networks of bitcoin users trading on the platforms from www.btc-alpha.com and www.bitcoin-otc.com respectively. In these two datasets, the nodes are the users from the platform, and an edge appears when one user rates another on the platform.
- The Digg dataset 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 represents a person, and the edge represents an email communication between two people.
- The Topology dataset is the network connections between autonomous systems of the Internet. Nodes are autonomous systems, and edges are connections between autonomous systems.

The first step is to pre-process the dataset. Since each dataset's edges are labeled with timestamps, duplicate edges in the edge stream are removed. Since there is no ground-truth anomalous edge in the original datasets, we generate anomalous edges with the spectral clustering algorithm and then inject anomalous edges in each dataset. To be concrete, the training dataset does not have any anomaly edge. For the snapshot of each timestamp in the test set, we apply spectral clustering to generate $p \times m^t$ anomalous edges, where $p$ is the anomaly proportion indicating the percentage of anomalous edges in each snapshot, and $m^t$ is the number of edges in the snapshot at timestamp $t$.

| Dataset | Number of Nodes | Number of Edges | Average Degree |
|---|---|---|---|
| UCI Messages | 1899 | 13838 | 14.57 |
| Email | 1866 | 39264 | 42.08 |
| Bitcoin-Alpha | 3777 | 24173 | 12.8 |
| Bitcoin-OTC | 5881 | 35588 | 12.1 |
| Digg | 30360 | 85155 | 5.61 |
| Topology | 34761 | 171420 | 9.86 |

Table 5.1: The statistics of the datasets

## 5.2 Baselines

We compare our framework with six baselines that can be divided into two groups: graph embedding methods and deep learning dynamic graph anomaly detection methods.

- **Node2Vec**[9]: Node2Vec combines breadth-first traversal and depth-first traversal in the random walks generation procedure. The embedding is learned using Skip-gram technology.
- **DeepWalk**[23]: DeepWalk is a random walk-based method for graph embedding. It generates random walks with a given length starting from a target node and uses a Skip-gram-like manner to learn embedding for unattributed graphs.
- **Spectral Clustering**[29]: To preserve the local connection relationship, the spectral embedding generates the node embedding by maximizing the similarity between nodes in the neighborhood.
- **NetWalk**[33]: NetWalk is a representative anomaly detection method for dynamic graph. It utilizes a random walk-based approach to generate contextual information and learns node embedding with an auto-encoder model. The node embeddings are updated incrementally over time via a reservoir-based algorithm. The anomaly is detected using a dynamic-updated clustering on the learned embedding.
- **AddGraph**[34]: AddGraph is an end-to-end dynamic graph anomaly detection approach. It leverages a GCN module to capture structure information, and employs a GRU-attention module to extract short and long term dynamic evolving.
- **StrGNN**[3]: StrGNN is an end-to-end graph neural network model for detecting anomalous edges in dynamic graphs. It leverages an h-hop enclosing subgraph as the network's input and combines GCN and GRU to learn structural- temporal information for each edge.

For the first three baselines, after representation learning, the same K-means clustering based method is used for anomaly detection.

## 5.3 Experimental Setup

In this section, we introduce the experimental design, the setting of model parameters and experimental infrastructure.

### 5.3.1 Experimental Design

In the experiments, we will test the anomaly detection methods over graphs without timestamps to see whether the framework can exploit the content and structural features effectively, and then extend to the dynamic graphs with all features. We divide the dataset into two parts, the first

50% as the training data and the latter 50% as the test data. When injecting the anomalous data into the test set, we experiment with three different anomaly proportions $p$, 1%, 5%, 10%. The metric used to compare the performance of different methods is ROC-AUC. The ROC curve indicates a plot of true positive rate against false positive rate where anomalous labels are viewed as “positive”. AUC is defined as the area under the ROC curve, which indicates the probability that a randomly selected anomalous edge is ranked higher than a normal edge. The value range of AUC is 0 to 1, and a larger value represents a better anomaly detection performance.

### 5.3.2 Parameter Settings

For edge related nodes sampling, we set the number $k$ of contextual nodes to be 5 and the size $\tau$ of time window to be 3. We use PPR diffusion in our experiments, which is computed by Eq.3.2. For structure and temporal node encoding section, the dimension of encoding $d_{enc}$ is 32. The parameter $d_{emb}$ in the dynamic graphics transformer is also 32. For the experiments on all datasets, the number of attention layers is 2 and the number of attention heads is 2. The framework is trained by Adam optimizer with a learning rate of 0.001. We train UCI Messages, Bitcoin-Alpha, and Bitcoin-OTC datasets with 100 epochs and train Digg and Email datasets for 200 epochs. The snapshot size is set to be 1000 for UCI Messages and Bitcoin-OTC, 2000 for Email and Bitcoin-Alpha, and 6000 for Digg.

### 5.3.3 Experimental Infrastructures

All experiments are conducted on google colab.

## 5.4 Results

### 5.4.1 Results on Graphs without Timestamps

| Dataset | Anomaly proportion | Netwalk | AddGraph | Our method |
|---|---|---|---|---|
| UCI Message | 1% | 0.7758 | 0.8083 | 0.8955 |
| | 5% | 0.7647 | 0.8090 | 0.8321 |
| | 10% | 0.7226 | 0.7688 | 0.8277 |
| Digg | 1% | 0.7563 | 0.8341 | 0.8633 |
| | 5% | 0.7176 | 0.8470 | 0.8583 |
| | 10% | 0.6837 | 0.8369 | 0.8472 |

Table 5.2: AUC results for anomalous scores on graphs without timestamps

The experiments on graphs without timestamps focus on mining the structural features in the graph. The results are shown in Table5.2, in which the data of baseline is reported by AddGraph[34]. We can see that our framework performs better on two datasets with different proportions of anomalies, which implies a better ability in capturing structural features. This is mainly due to the edge encoding pattern, which allows the framework to better capture nodes with greater connectivity strength to the target edge, so that the sampled node set can effectively represent the structural features of the target edge. Especially on large data sets like Digg, our method improves more than other baselines. This outstanding result demonstrates the ability of our framework to effectively utilize structural features.

### 5.4.2 Results on Dynamic Graphs

In the tests over dynamic graphs, we split the training data and test data into snapshots. According to the size of dataset, the snapshot size is set to 1,000 and 6,000 for UCI Message and Digg, 1,000 and 2,000 for Bitcoin-alpha and Bitcoin-OTC, 2,000 and 6,000 for Email and Topology, respectively. Then, we would like to show the performance of our framework compared to the baselines. The anomaly detection performance comparison of average AUC on all test set is demonstrated in Table 5.3-5.5.

| Methods | UCI | | | Digg | | |
|---|---|---|---|---|---|---|
| | 1% | 5% | 10% | 1% | 5% | 10% |
| Node2Vec | 0.7371 | 0.7433 | 0.6960 | 0.7364 | 0.7081 | 0.6508 |
| Spectral Clustering | 0.6324 | 0.6104 | 0.5794 | 0.5949 | 0.5823 | 0.5591 |
| DeepWalk | 0.7514 | 0.7391 | 0.6979 | 0.7080 | 0.6881 | 0.6396 |
| NetWalk | 0.7758 | 0.7647 | 0.7226 | 0.7563 | 0.7176 | 0.6837 |
| AddGraph | 0.8083 | 0.8090 | 0.7688 | 0.8341 | 0.8470 | 0.8369 |
| StrGNN | 0.8179 | 0.8252 | 0.7959 | 0.8162 | 0.8254 | 0.8272 |
| TADDY | 0.8912 | 0.8398 | 0.8370 | 0.8617 | 0.8545 | 0.8440 |
| Our Framework | **0.8954** | **0.8421** | **0.8347** | **0.8601** | **0.8561** | **0.8350** |

Table 5.3: AUC comparison on UCI and Digg

| Methods | Bitcoin-Alpha | | | Bitcoin-OTC | | |
|---|---|---|---|---|---|---|
| | 1% | 5% | 10% | 1% | 5% | 10% |
| Node2Vec | 0.6910 | 0.6802 | 0.6785 | 0.6951 | 0.6883 | 0.6745 |
| Spectral Clustering | 0.7401 | 0.7275 | 0.7167 | 0.7624 | 0.7376 | 0.7047 |
| DeepWalk | 0.6985 | 0.6874 | 0.6793 | 0.7423 | 0.7356 | 0.7287 |
| NetWalk | 0.8385 | 0.8357 | 0.8350 | 0.7785 | 0.7694 | 0.7534 |
| AddGraph | 0.8665 | 0.8403 | 0.8498 | 0.8352 | 0.8455 | 0.8592 |
| StrGNN | 0.8574 | 0.8667 | 0.8627 | 0.9012 | 0.8775 | 0.8836 |
| TADDY | 0.9451 | 0.9341 | 0.9423 | 0.9455 | 0.9340 | 0.9425 |
| Our Framework | **0.9306** | **0.9310** | **0.9416** | **0.9498** | **0.9365** | **0.9438** |

Table 5.4: AUC comparison on bitcoin-Alpha and Bitcoin-OTC

| Methods | Email | | | Topology | | |
|---|---|---|---|---|---|---|
| | 1% | 5% | 10% | 1% | 5% | 10% |
| Node2Vec | 0.7391 | 0.7284 | 0.7103 | 0.6821 | 0.6752 | 0.6668 |
| Spectral Clustering | 0.8096 | 0.7857 | 0.7759 | 0.6685 | 0.6563 | 0.6498 |
| DeepWalk | 0.7481 | 0.7303 | 0.7197 | 0.6844 | 0.6793 | 0.6682 |
| NetWalk | 0.8105 | 0.8371 | 0.8305 | 0.8018 | 0.8066 | 0.8058 |
| AddGraph | 0.8393 | 0.8627 | 0.8773 | 0.8080 | 0.8004 | 0.7926 |
| StrGNN | 0.8775 | 0.9103 | 0.9080 | 0.8553 | 0.8352 | 0.8271 |
| TADDY | 0.9348 | 0.9257 | 0.9423 | 0.8953 | 0.8952 | 0.8934 |
| Our Framework | **0.9431** | **0.9350** | **0.9311** | **0.8921** | **0.8896** | **0.8897** |

Table 5.5: AUC comparison on benchmark Email and Topology

From the experimental results, we make some summaries of the observed results.

- Our framework performs better than TADDY on most datasets, but the gap is narrow. In terms of other baselines, no matter the proportion of anomalies, our framework consistently outperforms them and achieves an average performance gain of 5.26% on AUC compared to the best one of them. This is because our framework builds inclusive edge encoding using extracted structure and temporal information and simulates capturing structural dynamic features and temporal dynamic features using a transformer encoder.

- Compared to graph embedding-based baselines, deep learning-based dynamic graph anomaly detection methods AddGraph, StrGNN, TADDY and our framework perform better. This is because deep neural network models effectively summarize structural features. And by considering the appearance time of snapshots, these methods can discover the graph's evolution.

- Our framework is more advantageous when the proportion of anomalies is low. In particular, the average difference in AUC between the our framework and the best baseline other than TADDY is 6.2% when the percentage of anomalies is 1%, whereas the gap is 4.46% and 5.04% when there are 5% and 10% of anomalies. This is because our algorithm for generating anomalies adequately considers the structural features of the graph, making the generated anomalies evident. This guarantees robustness under varying test set anomaly proportions.

- Our framework has a much better performance on datasets Btc-Alpha and Btc-Otc. Our framework results in a 7.19% improvement in AUC for Btc-Alpha and Btc-Otc compared to the ideal baseline other than TADDY, which is much greater than the 4.21% increase in AUC for other datasets.

## 5.5 Parameter Sensitivity

In this section, we look into how the framework is affected by hyper-parameters such as the number of contextual nodes and the size of time windows in edge-related nodes sampling, the dimension of embedding, the number of attention layers and the proportion of training data. We conduct the experiments on three datasets, UCI Messages, Bitcoin-Alpha and Bitcoin-OTC, respectively. During these tests, all of the other parameters have their default settings maintained, and the performance is evaluated with a 10% anomaly percentage setting.

### 5.5.1 Parameters of Edge-related Nodes Sampling

In order to investigate how the number of contextual nodes $k$ and the size of the temporal window $\tau$ influence edge-related nodes sampling, we restrict the value of $k$ from 1 to 10 and the value of $\tau$ from 1 to 5. Fig. 5.1 and Fig. 5.2 illustrate the sensitivity of $k$ and $\tau$.

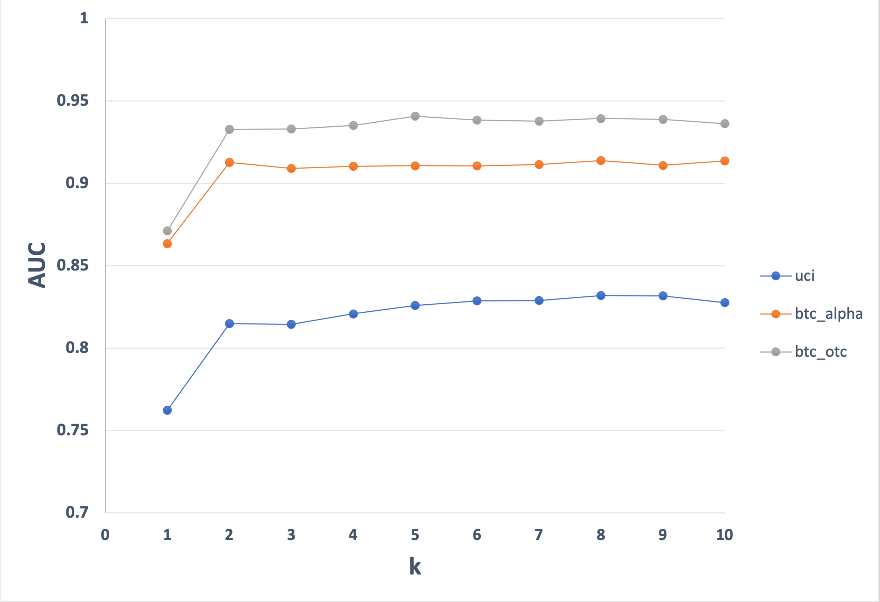


Figure 5.1: Sensitive of k

As shown in Fig.5.1, the performance of anomaly identification is relatively bad when the number of contextual nodes, $k$, is very small. There is a significant improvement in AUC as $k$ increases. The AUC reaches its greatest value when $k$ is equal to 8. The maximum value does not, however, stand out since the detection performance stabilizes and the AUC exhibits minute fluctuations when $k$ is larger than 5. The three lines in the Fig.5.1 have a similar pattern, which suggests that a sufficient number of contextual nodes are critical for anomaly detection, since the anomalous properties of edges depend heavily on their associated subgraph structure. Furthermore, the performance improvement is small when too many context nodes are taken into account. In addition, the improvement in performance is negligible when an excessive number of contextual nodes are taken into account. Nevertheless, as the temporal complexity is related to $k$, the larger $k$, the less efficiently the framework operates. To achieve a compromise between framework performance and execution efficiency, we set $k$ to 5 as a result.

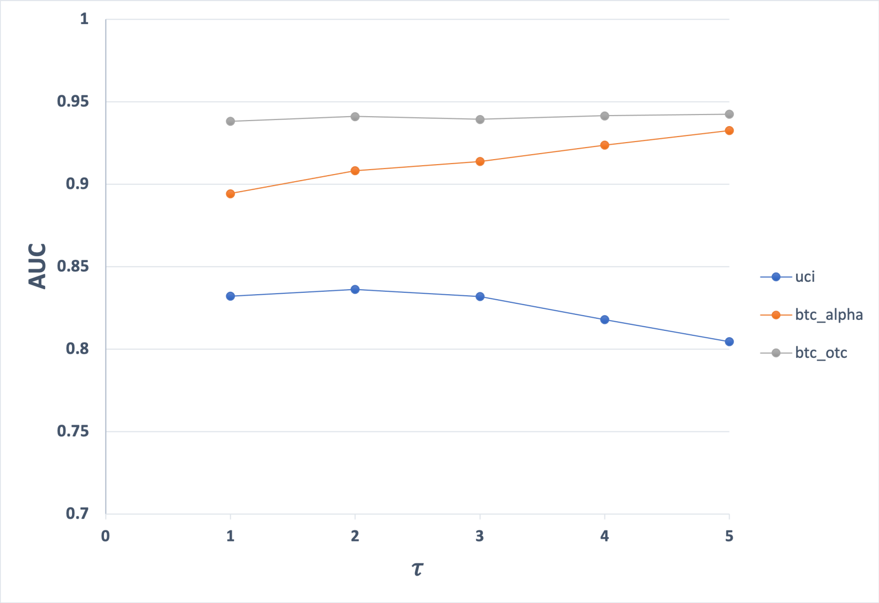


Figure 5.2: Sensitive of $\tau$

Fig.5.2 displays the AUC of the framework with varying time window sizes on three distinct datasets. The optimal size of the time window varies across different datasets. For instance, with the UCI dataset, there is a modest declining trend in the AUC line overall, so a narrower time window is more advantageous. For the Btc-alpha and Btc-otc datasets, however, wider time windows produce better experimental outcomes. The fundamental reason for this is that the temporal dependence of edges in dynamic graphs is reliant on the underlying dataset. When the edge appearance is dependent on the evolution of the preceding graph over an extended period of time, a wider time window is necessary to capture the temporal features. For datasets that are less time-dependent, such as the UCI dataset, where anomalous edges are relevant to the most recent snapshot, a broad time window may bring more noise and irrelevant information. As a result, in our trials, we set the $\tau$ to 2 for the UCI dataset and 5 for the other datasets.

### 5.5.2 Parameters of Transformer

In this subsection, we examine the influence of the Transformer model's hyperparameters on the framework, namely the embedding dimension $d$ and the number of hidden layers $l$. The values of $d$ are selected from the range [4, 8, 16, 32, 64] and the values of $l$ are selected from the range [1, 2]. The experimental results are presented in Fig.5.4, Fig.5.5 and Fig.5.3

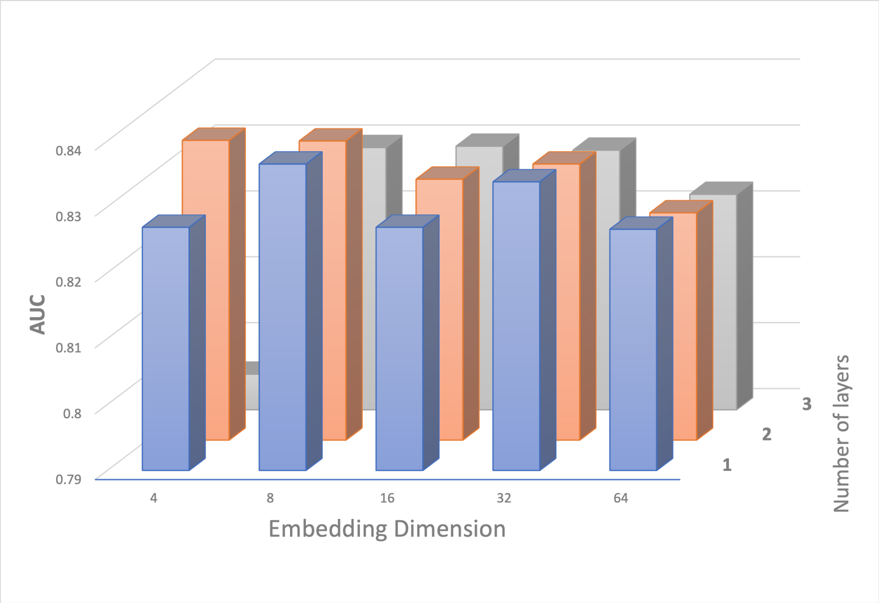


Figure 5.3: UCI

As shown in Fig.5.3, the optimal choice for the UCI dataset is $d = 8$. The performance of the framework degrades and fluctuates as $d$ increases. We believe that when $d$ is too large, the transformer model captures noise information, and when $d$ is too small, the framework is unable to collect sufficient feature information. Compared to the embedding dimension, the number of layers has a modest impact on performance; only when there are three hidden layers and the embedding dimension is four does the AUC alter dramatically and reach its minimum value.

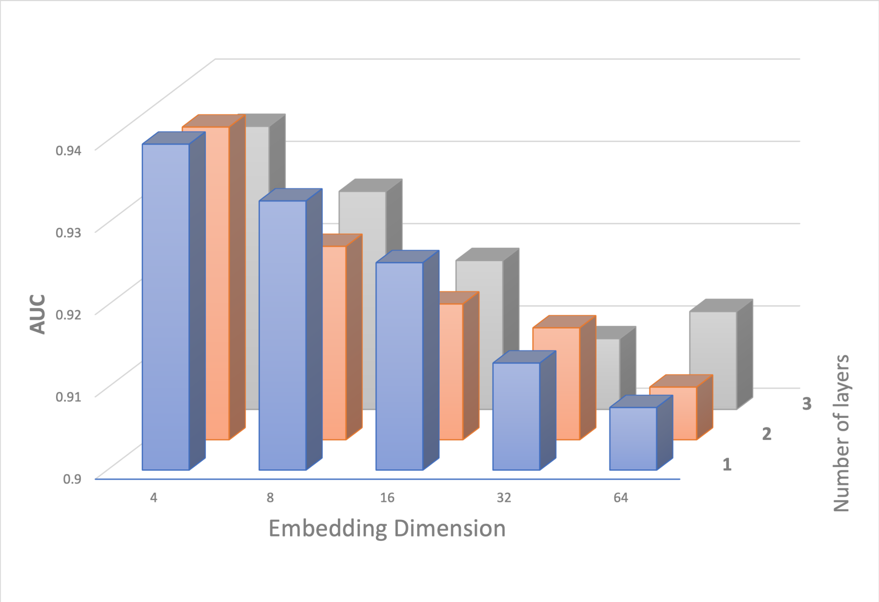


Figure 5.4: Btc-Alpha

For Btc-Alpha dataset, the performance of the framework degrades continually as $d$ increases gradually. This discovery shows that when $d$ increases, a great deal of noise may be introduced into the model. As for the number of hidden layers, the experimental results are better when $l$=1.

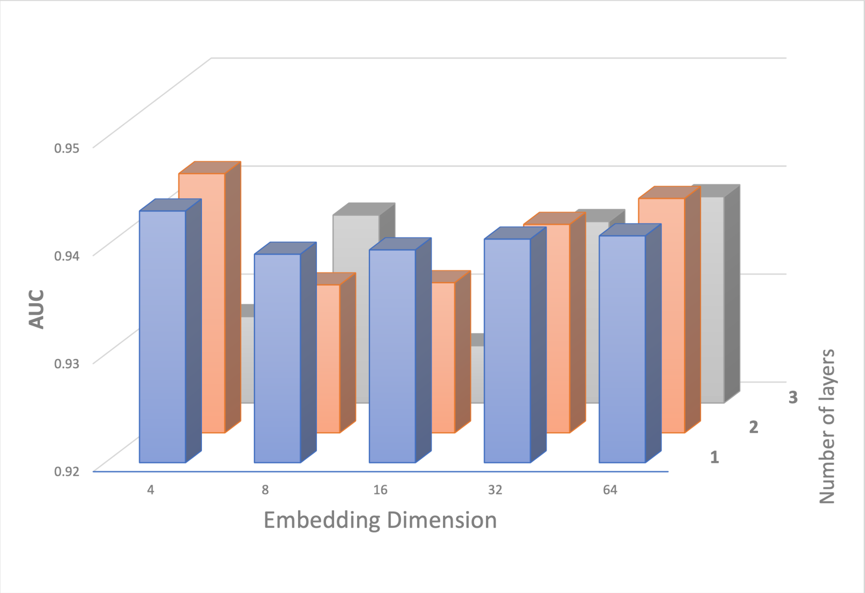


Figure 5.5: Btc-Otc

As shown in Fig.5.5, when the number of layers is 1 and 2, the performance of the framework changes in a similar trend as the embedding dimension increases, first decreasing and then gradually increasing, with the AUC reaching the lowest value when $d$=8. And when there are three layers, the trend is distinct: first ascending, then decreasing, then ascending and stabilizing.

### 5.5.3 Training data proportion

In this subsection, we examine how the framework performs using various ratios of training data. The model is trained with 40%, 45%, 50%, 55%, 60%, 65% and 70% of the data as the training set, while the other hyperparameters remain unchanged. The AUC results on three datasets are shown in Fig.5.6, Fig.5.7 and Fig.5.8.

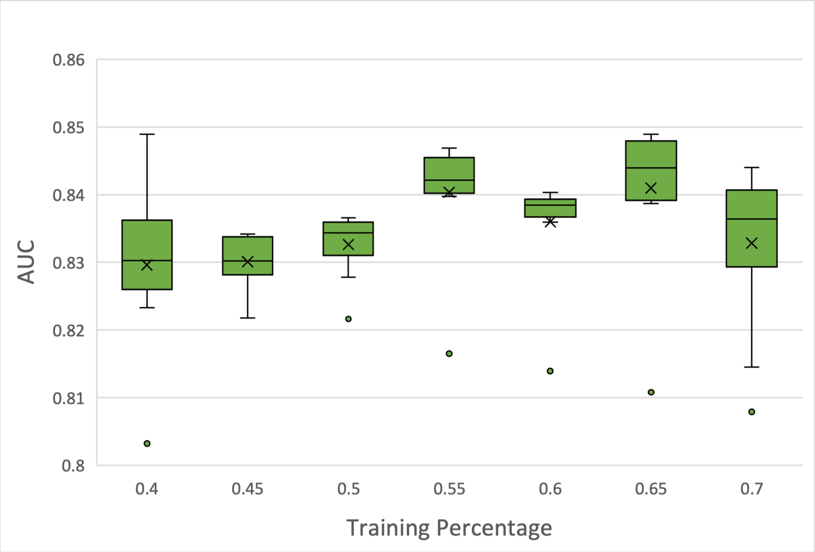


Figure 5.6: UCI

Fig.5.6 illustrates the AUC for the UCI dataset. As shown by the results, the AUC increases from 40% to 55% of training data, after which the performance fluctuates, and the optimal performance is reached when the ratio is 65%. This suggests that a sufficient amount of training data can enhance the model's performance.

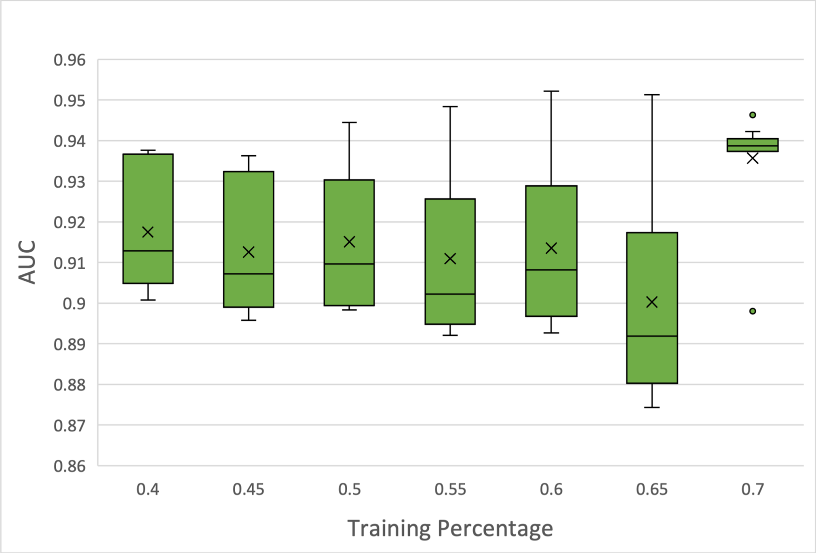


Figure 5.7: Btc-Alpha

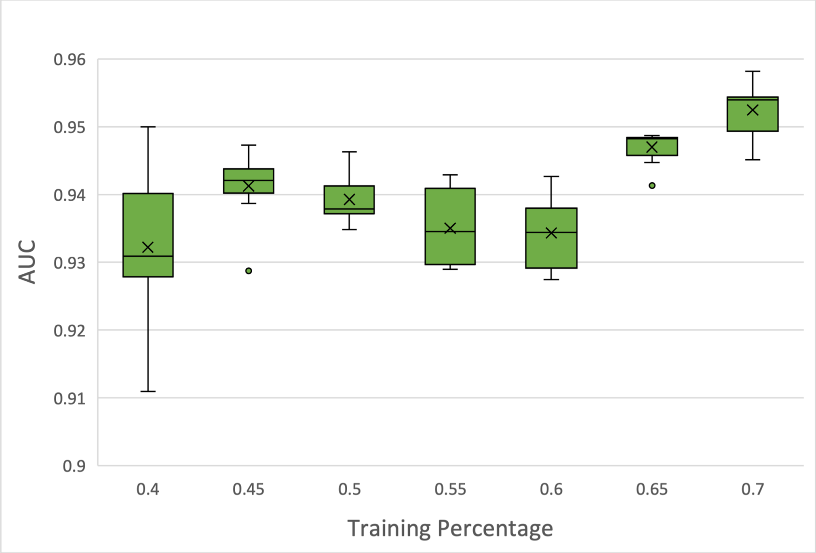


Figure 5.8: Btc-Otc

When doing trials on both Btc-Alpha and Btc-Otc datasets, the models' performance varies differently as the proportion of training data increases. For the Btc-Alpha dataset, the AUC gradually decreases as the training percentage increases to 65%. And for the Btc-Otc dataset, the AUC increases to over 0.94 when the training data is 45%, before decreasing progressively. Nonetheless, there are parallels between the experimental outcomes of the two datasets. When the percentage of training data exceeds 70%, the AUC improves significantly and reaches its maximum value.

## 5.6 Ablation Study

The edge encoding consists of three parts, namely diffusion-based structure encoding(global) and importance structure encoding(local) and normalized relative temporal encoding(time). We perform ablation experiments on three datasets: UCI, Btc-Alpha and Btc-Otc. The results are summarized in Table.5.6. In addition, we examine the UCI dataset in further depth. Specifically, with two components held constant, continuously vary the weights of the remaining components to assess each component's impact on the total performance. Each dataset is analyzed with 10% anomalies, and all other parameters are set to their default values. The results are shown in Fig.5.9.

| | UCI | Btc-Alpha | Btc-Otc |
|---|---|---|---|
| global+local+time | **0.8347** | 0.9416 | **0.9438** |
| local+time | 0.8279 | **0.9423** | 0.9431 |
| global+time | 0.5002 | 0.512 | 0.5031 |
| global+local | 0.8287 | 0.9104 | 0.9408 |

Table 5.6: Ablation Study

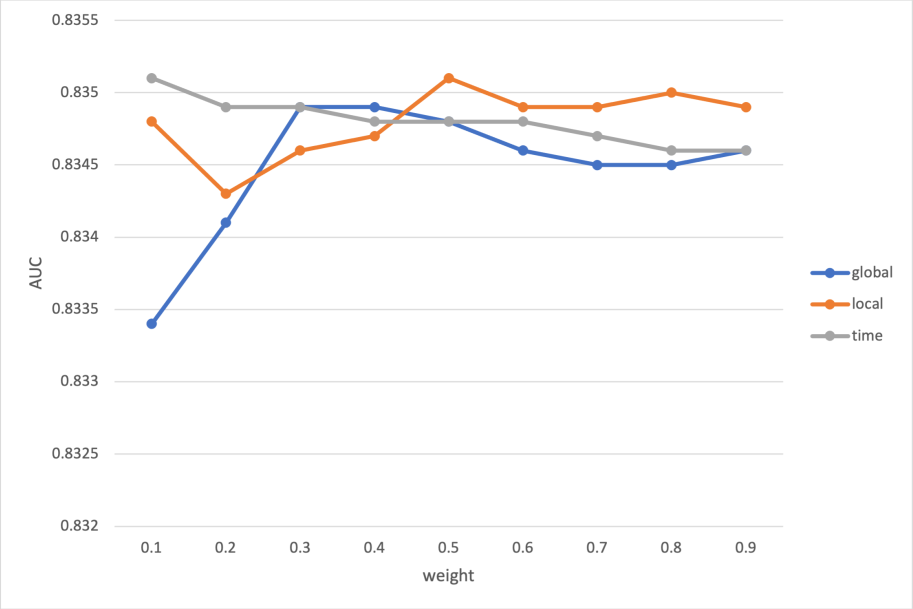


Figure 5.9: Experiment of varying the weight of each component

By comparing and analyzing the data in Table.5.6 and Fig.5.9, we can get some insights.

- The most crucial component of edge encoding is importance structure encoding. Without this component, the AUC value plummets to roughly 50%, rendering the abnormalities undetectable. This finding highlights the significance of local structural information in anomaly detection.
- The contribution of diffusion-based structure encoding to anomaly detection is insignificant. Ignoring this component of the encoding results in a slight drop in AUC. For the small dataset Btc-Alpha, removing this part of the encoding improves the performance of the framework instead. We conclude that the global features of the graph structure provided by the diffusion-based structural encoding have a minimal impact in comparison to the local features.

- Normalized relative temporal encoding provides only a tiny improvement in anomaly detection. The temporal encoding states the time of appearance, which is relatively unimportant. Because the relevant subgraphs of the target edges barely change within the time window and already cover enough features. With the exception of the Btc-Alpha dataset, the combination of all three types of encoding yielded the highest AUC values. This further demonstrates the usefulness of structure and temporary encoding for anomaly detection.

- As seen in Fig.5.9, each encoding contributes to the improvement of the framework's performance. As the weight of structure features continue to climb, the performance of the framework rises and then falls. As for the time feature, the AUC decreases as the weight increases, and the weight set to 0.1 may be the optimal choice.

# Chapter 6

# Conclusions

In the last chapter, all the results achieved from the previous chapters are summarized. At first, we answer the research questions and then summarize our framework. After that, we discuss limitations and future work.

## 6.1 Summary

We try to investigate the anomaly detection problem using transformer model in dynamic graph scenarios. For the structure features of the dynamic graph, we capture two structure features of the target edges, global features and and local features. For the temporal features, we use normalized relative time. Before inputting into the transformer, we sum up the three encodings so that the structure and temporal encoding of the target edges are obtained after fusion. In terms of the the framework, we propose an end-to-end anomaly detection framework, which is composed of four components: edge related nodes sampling, structure and temporal edge encoding, dynamic graph transformer, and anomaly detector. The framework constructs an informative and comprehensive edge encoding to better represent the role of edges in the evolving graph, and captures the joint structural and temporal information of dynamic graphs through the transformer model. After that, experiments on several benchmark datasets show that the proposed framework is efficient at detecting anomalies in dynamic graphs.

## 6.2 Limitations

**Temporal features provide limited support** Temporal features offer simply the moment at which a snapshot is generated, which is merely a number and is unrelated to the dynamic graph's content. In addition, it is evident from the experiments that the time feature improve the performance of the framework slightly. Regarding the outstanding performance of the framework, it almost depends on the local structure features.

**No test on practical applications** We perform testing on benchmark datasets that are created with no anomalous data, so we design the anomaly generation algorithm. However, The anomaly generation algorithm considers structural features, making it simple for the framework to identify anomalies. Therefore, it is more persuasive to conduct tests on real applications, where anomalies are genuine.

**Models constrained by data volume** Obtaining the primary features rely on the graph's adjacency matrix. This means that it is almost impossible to achieve when the graph has millions of nodes because it requires memory at the level of TBs.

## 6.3 Future Work

Based on the experimental findings and the analysis of the model limitations, future work can focus on the following directions.

**Backbone Network** Our model's backbone network is designed with a basic transformer encoder construction. We can still try more complex backbone networks to optimize the model.

**Temporal Feature** Because our temporal features bring only a small boost to the framework, and a possible overfitting problem can be seen through the weight experiments. We can seek a strategy for extracting temporal features that is more strongly linked with the graph.

**Architecture** In real-world applications, the volume of data is enormous, and our model cannot handle graph data in the millions or higher. The adjacency matrix is the foundation of the model, so for matrix calculations, we can do optimization at the architectural level, such as MapReduce and parallel computing, so that the million-level matrix operations can be performed.

# Bibliography

[1] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 410–421. Springer, 2010. 5

[2] Sambaran Bandyopadhyay, Saley Vishal Vivek, and MN Murty. Outlier resistant unsupervised deep architectures for attributed network embedding. In *Proceedings of the 13th international conference on web search and data mining*, pages 25–33, 2020. 5

[3] Lei Cai, Zhengzhang Chen, Chen Luo, Jiaping Gui, Jingchao Ni, Ding Li, and Haifeng Chen. Structural temporal graph neural networks for anomaly detection in dynamic graphs. In *Proceedings of the 30th ACM international conference on Information & Knowledge Management*, pages 3747–3756, 2021. 3, 8, 19, 28

[4] Yen-Yu Chang, Pan Li, Rok Sosic, MH Afifi, Marco Schweighauser, and Jure Leskovec. F-fade: Frequency factorization for anomaly detection in edge streams. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 589–597, 2021. 7

[5] Zhengzhang Chen, William Hendrix, and Nagiza F Samatova. Community-based anomaly detection in evolutionary networks. *Journal of Intelligent Information Systems*, 39(1):59–85, 2012. 8

[6] Qiang Cui, Shu Wu, Yan Huang, and Liang Wang. A hierarchical contextual attention-based network for sequential recommendation. *Neurocomputing*, 358:141–149, 2019. 2

[7] Dongsheng Duan, Lingling Tong, Yangxi Li, Jie Lu, Lei Shi, and Cheng Zhang. Aane: Anomaly aware network embedding for anomalous link detection. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1002–1007. IEEE, 2020. 6

[8] Dhivya Eswaran and Christos Faloutsos. Sedanspot: Detecting anomalies in edge streams. In *2018 IEEE International conference on data mining (ICDM)*, pages 953–958. IEEE, 2018. 7

[9] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016. 28

[10] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126, 2004. 1

[11] Renjun Hu, Charu C. Aggarwal, Shuai Ma, and Jinpeng Huai. An embedding approach to anomaly detection. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 385–396, 2016. 2

[12] Renjun Hu, Charu C Aggarwal, Shuai Ma, and Jinpeng Huai. An embedding approach to anomaly detection. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 385–396. IEEE, 2016. 5

[13] Fei Jie, Chunpai Wang, Feng Chen, Lei Li, and Xindong Wu. Block-structured optimization for anomalous pattern detection in interdependent networks. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1138–1143. IEEE, 2019. 8

[14] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 2

[15] Risi Imre Kondor and John Lafferty. Diffusion kernels on graphs and other discrete structures. In *Proceedings of the 19th international conference on machine learning*, volume 2002, pages 315–322, 2002. 11

[16] Yixin Liu, Zhao Li, Shirui Pan, Chen Gong, Chuan Zhou, and George Karypis. Anomaly detection on attributed networks via contrastive self-supervised learning. *IEEE transactions on neural networks and learning systems*, 33(6):2378–2392, 2021. 19

[17] Yixin Liu, Shirui Pan, Yu Guang Wang, Fei Xiong, Liang Wang, Qingfeng Chen, and Vincent CS Lee. Anomaly detection in dynamic graphs via transformer. *IEEE Transactions on Knowledge and Data Engineering*, 2021. 2, 8, 19, 20

[18] Benjamin A Miller, Nicholas Arcolano, and Nadya T Bliss. Efficient anomaly detection in dynamic, attributed graphs: Emerging phenomena and big data. In *2013 IEEE International Conference on Intelligence and Security Informatics*, pages 179–184. IEEE, 2013. 6

[19] Misael Mongiovi, Petko Bogdanov, Razvan Ranca, Evangelos E Papalexakis, Christos Faloutsos, and Ambuj K Singh. Netspot: Spotting significant anomalous regions on dynamic networks. In *Proceedings of the 2013 Siam international conference on data mining*, pages 28–36. SIAM, 2013. 8

[20] Linshu Ouyang, Yongzheng Zhang, and Yipeng Wang. Unified graph embedding-based anomalous edge detection. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020. 6

[21] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999. 11

[22] Bryan Perozzi and Leman Akoglu. Scalable anomaly ranking of attributed neighborhoods. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 207–215. SIAM, 2016. 1

[23] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014. 28

[24] Ryan A Rossi, Brian Gallagher, Jennifer Neville, and Keith Henderson. Modeling dynamic behavior in large evolving graphs. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 667–676, 2013. 7

[25] Srijan Sengupta. Anomaly detection in static networks using egonets. *arXiv preprint arXiv:1807.08925*, 2018. 1

[26] Kumar Sricharan and Kamalika Das. Localizing anomalous changes in time-evolving graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1347–1358, 2014. 2

[27] Nguyen Thanh Tam, Matthias Weidlich, Bolong Zheng, Hongzhi Yin, Nguyen Quoc Viet Hung, and Bela Stantic. From anomaly detection to rumour detection using data streams of social platforms. *Proceedings of the VLDB Endowment*, 12(9):1016–1029, 2019. 7

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 2, 12, 13

[29] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007. 28

[30] Haibo Wang, Chuan Zhou, Jia Wu, Weizhen Dang, Xingquan Zhu, and Jilong Wang. Deep structure learning for fraud detection. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 567–576. IEEE, 2018. 6

[31] Huan Wang, Jia Wu, Wenbin Hu, and Xindong Wu. Detecting and assessing anomalous evolutionary behaviors of nodes in evolving social networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 13(1):1–24, 2019. 7

[32] Chenming Yang, Liang Zhou, Hui Wen, Zhiheng Zhou, and Yue Wu. H-vgrae: A hierarchical stochastic spatial-temporal embedding method for robust anomaly detection in dynamic networks. *arXiv preprint arXiv:2007.06903*, 2020. 8

[33] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. 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*, pages 2672–2681, 2018. 2, 3, 7, 28

[34] Li Zheng, Zhenpeng Li, Jian Li, Zhao Li, and Jun Gao. Addgraph: Anomaly detection in dynamic graph using attention-based temporal gcn. In *IJCAI*, pages 4419–4425, 2019. 3, 7, 28, 29

[35] Mengyu Zheng, Chuan Zhou, Jia Wu, Shirui Pan, Jinqiao Shi, and Li Guo. Fraudne: a joint embedding approach for fraud detection. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018. 9

[36] Panpan Zheng, Shuhan Yuan, Xintao Wu, Jun Li, and Aidong Lu. One-class adversarial nets for fraud detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1286–1293, 2019. 7

## Appendix A

# My First Appendix

In this file (appendices/main.tex) you can add appendix chapters, just as you did in the thesis.tex file for the ‘normal’ chapters. You can also choose to include everything in this single file, whatever you prefer.