Graph Anomaly Detection with Link Prediction

Luo, Yao

Award date:
2021

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science

Graph Anomaly Detection with Link Prediction

Master Thesis

Yao Luo

Supervisors:
dr. Morteza Monemizadeh
dr. Tim Ophelders
dr. Yulong Pei

Final version

Eindhoven, December 2021

Abstract

Given a stream of edges in a dynamic graph, how can we detect anomalous edges in near real-time with constant memory? In this thesis, we propose algorithms with link prediction measures to detect anomalies and leverage these measures to implement the performance of the baseline algorithm. Reservoir sampling is used to ensure the time and memory complexity of our approach. Experimental results on five real-world datasets show that our algorithms outperform the baseline approach.

Preface

First of all, I would like to thank my supervisor, dr. Morteza Monemizadeh, for his valuable guidance in this thesis. Without his help and advice, I could not have completed this graduation project. Second, I would also like to thank dr. Tim Ophelders and dr. Yulong Pei, for accepting to be on my assessment committee. Thirdly, I would like to express my gratitude to my supportive friends for your presence in my life. Last but not least, I am eternally grateful to my parents for their unfailing support and continuous encouragement throughout my life. Thank you all.

Contents

Chapter 1

Introduction

Anomalies [28], also referred to *outliers* [42, 37], imply something that deviates from what is standard, normal, or expected. An example could be the red fish in Figure 1.1[1]. The idea behind anomaly detection is to find patterns in data that deviate significantly from expected behavior and these nonconforming patterns could represent a detrimental event or a positive opportunity, which might have great importance but are hard to find. In reality, one common type of anomalies are anomalies in *graph and network structures*. Typical examples of this kind of anomalies are *intrusion detection* [20], *financial fraud* [41], *scan emails* [29], *fake ratings* [27], etc.

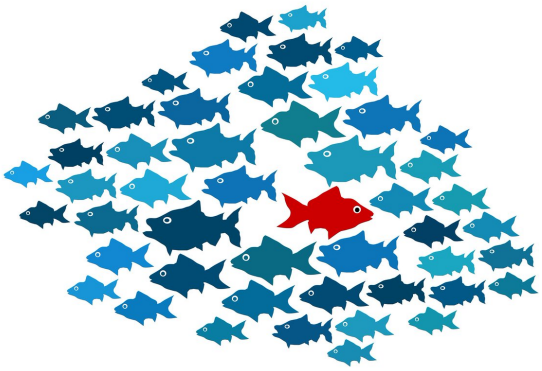

Figure 1.1: An example of anomalies

Anomaly detection in graphs has been a well-explored research field and several proposed algorithms focus on static graphs [6, 45, 26, 14, 16, 40]. However, these approaches require a large amount of memory to store the entire graph for analysis, which is sometimes impossible since edges and nodes are drawn from the real world and the graph is rapidly evolving and expanding[2] [19]. Furthermore, anomaly detection algorithms for static graphs are ineffective

[1] https://www.analyticsvidhya.com/blog/2021/04/dealing-with-anomalies-in-the-data/
[2] https://sproutsocial.com/insights/social-media-changes/

in capturing changes in real-world data because many real-world graphs are dynamic and evolve over time, and methods relying on static graphs may represent only a single snapshot and miss the temporal characteristics of the graph [5].

Since real-world networks are rapidly changing, *dynamic graph models* (that allow and track updates to the structure of evolving graphs) have become popular techniques for modeling real-world networks. Typically, in dynamic graph models, we assume that edges (e.g., interactions or connections among nodes) of an underlying dynamic network is given in a streaming fashion and that is why in dynamic graph models, we often model the *dynamism* as a stream of updates to an underlying dynamic network [7].

Many domains of science could benefit from graph anomaly detection techniques and algorithms. Here some concrete examples are given.

- Social networks [42, 43]: Twitter[3], Facebook[4], and Tiktok[5] are well-known examples. These *online social networking services* build enormous social networks that allow users to connect with others based on various types of interactions, e.g., friendships, follows, likes, messages, posts, and so on. In these large networks, vertices represent the users and edges simulate their interactions, while anomalies can be the discovery of new trending topics, bot users, attacks, and abnormal behaviors.

- Intrusion attacks [20, 11]: This system can be modeled as a dynamic graph where nodes representing machines and edge representing connections between machines. The typical anomalous behaviors are that a group of attackers pose as normal machines and then connect to a set of targeted devices multiple times in order to restrict access or search for potential vulnerabilities.

- Financial fraud [41, 54]: A realistic graph of all entities in a financial system might have millions of linked nodes. Take the bank system as an example. The nodes can be regarded as bank accounts and the edges are the transactions, while the anomalous behaviors can be fraudulent credit, or a large number of transactions in a short time.

In dynamic graph algorithms, we often *aggregates* edges into graph snapshots and then process the graph stream over time [49, 44, 10, 53, 50, 9, 22]. However, these algorithms tend to aggregate edges in a short time period as a snapshot, resulting in anomalies that may not be detected in real-time, whereas to capture and recover from the malicious activities in a timely manner, we are supposed to detect anomalous behavior in real-time or near real-time, that is, to determine whether a coming edge is anomalous or not as soon as it arrives. Moreover, the global comparison of graphs often has subtle changes that cannot be noticed and thus a more granular analysis of the changes may be needed.

Anomaly detection in edge streams, i.e. the objects in the stream are individual edges in the graph and are processed independently, has also gained great attention recently [55,

[3]https://twitter.com/
[4]https://www.facebook.com
[5]https://www.tiktok.com

42, 48, 21, 12, 13]. However, many of these existing methods focused on assigning anomaly scores based on the occurrence of edges or changes in structural patterns. They rarely take advantage of connectivity patterns of link prediction in graphs and networks and thus completely ignore combining the information from link prediction with structural patterns.

Many approaches, such as [32, 52, 56], have introduced the importance of link prediction on predicting the future changes of evolving graphs. Link prediction is usually used to find anomalous edges that should appear do not [31], while in this paper, we utilize link prediction to determine whether the incoming edges is expected to occur or not, i.e. estimate the likelihood of the occurrence of the incoming edges, and then update the anomaly scores, thus reducing or increasing the anomaly probability of the edges.

In this paper, we focus on graph anomaly detection in edge streams and intend to process the edges in constant time and memory, regardless of the stream size. Importantly, we leverage the link prediction method to implement the performance of the baseline algorithm, that is, combine the information of connectivity patterns with other patterns.

1.1 Problem Statement

Before discussing the structure of the algorithm, it is essential to to present the details of the data we use for modeling. Given a time-evolving graph G, there is a stream of edges $E = \{e1, e2, e3, ...\}$. Every coming edge is a tuple $e_i = (u_i, v_i, t_i)$ consisting of a source node $u_i \in V$, a destination node $v_i \in V$ and its time occurrence $t_i \in T$ which is assumed as a discrete variable.

Here we define V as the set of all nodes which is not assumed to be known as a priori, that is, this set can grow as this dynamic graph G evolves. For each node in the set V, it is supposed to have a unique label that does not change over time, such as a user ID, an IP address, or a bank account, thus $label(u) = label(v)$ iff $u = v$. E is the set of all edges in the stream and every edge inside represents the connection of two nodes in the graph. Edges are allowed to arrive simultaneously, i.e., for $e_i = (u_i, v_i, t_i)$ and $e_{i+1} = (u_{i+1}, v_{i+1}, t_{i+1})$, we have $t_{i+1} \geq t_i$ (equal is allowed). The edges in the graph can be modeled as either directed or undirected since undirected edges can simply be regarded as two directed edges coming at the same time, in different directions. The dynamic graph G can be multigraph, that is, there is no limit to the number of times an edge can appear between the same pair of nodes and two nodes can be linked multiple times in the stream. However, since we only focus on some specific measures of link prediction which introduced in section 3.3, we attach more importance on the neighbors of two nodes of the arriving edge, rather than the number of that edge. Therefore we do not use the weight option to store the occurrences of the edge.

Then we define our research question as

Given a stream of edges in a dynamic graph, in order to detect anomalous behavior, how can we assign anomaly scores to edges in an online manner using constant memory and near real-time processing time, regardless of the size of the stream?

To implement this algorithm to meet the above requirements, we maintain a fixed-size

sample to represent the edges seen so far and update it for every arriving edge, then leverage this sample to score these edges. Thus this research question can be further divided into the problem of *edge sampling* and *scoring function*.

1.2 Outline

Following the introduction, chapter 2 begins with the review of related works on graph anomaly detection. Then chapter 3 focuses on the background knowledge on which this research is based. Chapter 4 are the main section that describe the construction of our algorithms in detail. Next chapter 5 present our experiments and results. Finally, the conclusions are given in chapter 6.

Chapter 2

Related Works

Since the 19th century, the statistics society has conducted substantial research on spotting anomalies or outliers [37]. As time goes by, various graph anomaly detection approaches have been developed. In this chapter, a selection of previous researches from the literature is introduced and a brief statement is given for each category.

2.1 Static Graphs

Anomaly detection in static graphs is performed on static graphs or snapshots of a changing graph at a certain time, and anomalies such as nodes, edges, and subgraphs are found based on the structural and node information of the graph.

Anomalous node detection. The features of nodes mainly include node outness, inness, betweenness centrality, eigenvector centrality, and clustering coefficient, etc. The representative method is OddBall [6] published by Akoglu et al. in 2010, a feature-based anomaly detection method, introduces egonet features, such as density, ranks, egonet weights, and eigenvalues. This algorithm observes the distribution pattern of egonet-based features and finds anomalous nodes that deviate from the pattern, but it is only applicable to anomaly detection in weighted graphs, which are essentially heuristic rules without explicit test statistics and decision rules. Sengupta et al. [45] propose a statistical decision rule for detecting anomalous clusters using the P-value of egonet and can identify the nodes that form the clusters.

Anomalous edge detection. The edge that is connected between nodes or graphs with different properties might be defined as an anomaly. Gyongyi et al. [26] implement TrustRank, a spam detection algorithm for spam web pages, which assumes that the connection represents the trust of the two web pages. Chakrabarti at al. [14] propose their algorithm based on graph partitioning, which can detect anomalous edges that deviate from the large clusters of the graph. The core idea of this method is that if removing an edge makes the graph easier to partition, then the two connected nodes of the removed edge are

anomalous.

Anomalous subgraph detection. The characteristics of graphs are mainly about the number of connected components, the distributions, the principal eigenvalues, the minimum spanning tree weights, the average node depth, global concentration factor, etc. Charikar et al. [16] propose to define the density of a subgraph using the average degree of the subgraph. Perozzi et al. [40] implement an algorithm to detect anomalous neighborhoods in attribute graphs. A measure called normality is introduced to quantify the quality of attributed neighborhoods with their internal consistency, and external separability on the boundaries.

Static graphs not only have to process the entire graph in a large memory or offline method, but also ignore the temporal dimension when modeling data, i.e., the graph data in the real world always changes over time. Therefore, to address the limitations in static graph anomaly detection, dynamic graph anomaly detection techniques are further developed.

2.2 Dynamic graphs

In the real world, graphs are constantly changing while dynamic graphs can serve as a powerful way to model an evolving stream. Thus, anomaly detection in dynamic graphs requires real-time detection of anomalous behavior or anomalous time points, where anomalous behavior includes anomalous vertices, edges, subgraphs, or features of the graph.

2.2.1 Graph Streams

For anomaly detection on graph streams, the input is a stream of graph snapshots over time. When modeling the data, many methods process the arriving edge into the graph snapshots and then detect anomalies.

Anomalous node detection. For each time snapshot, features in the nodes are extracted as evaluation metrics and nodes that exhibit irregular behavior compared to most other nodes are detected as anomalies. Sun et al. [49] propose the dynamic tensor analysis method called DTA and its variants with streaming tensor called STA, which could summarize the hidden correlations of high-order and high-dimensional data and then detect nodes with high reconstruction error. Rossi et al. [44] introduce a dynamic behavioral mixed-membership model called DBMM to detect anomalous nodes in large-scale, dynamic attribute graphs. This method uses a feature-based representation to capture temporal behavioral transition patterns of anomalous nodes and then extrapolate to unobserved nodes.

Anomalous subgraph detection. The graph structure of anomalous subgraphs usually differs from normal subgraphs and when a set of subgraphs is obtained, anomalous subgraphs can be identified based on the anomaly scores assigned to adjacent time-step subgraphs. Beutel et al. [10] propose CopyCatch to analyze the bipartite graph with timestamps for every edge that forms this graph. This algorithm searches for near-bipartite cores with certain edge constraints and then checks temporal coherence within these cores for the beha-

vior to be considered anomalous. Wang et al. [53] detect anomalies in dynamic graphs from both node-based and subgraph-based approaches. Two models for subgraphs are developed to track the structural changes of graphs and reduce the amount of false-positive results obtained from the node-based model. This combination of multiple anomaly detection methods to detect outliers in dynamic graphs is one of the research directions.

Anomalous event detection. Event anomaly detection is to find the time points that deviate from the normal distribution in the time series data, and then leverage the static graph anomaly detection method to investigate the cause of the anomaly. To detect anomalies in temporal data, the features are extracted from each time snapshot, and the adjacent data are evaluated using different similarity methods. Sun et al. [50] propose the Compact Matrix Decomposition method call CMD to compute sparse low-rank approximations for high-dimensional matrix. They further extend this method to analyze dynamic graphs and spot event anomalies with the estimation of the reconstruction error. Berlingerio et al. [9] introduce NetSimile to extract the structural features and compute network similarity for every given two graphs that may have no common nodes or edges. Eswaran et al. [22] present a randomized sketching-based algorithm called SPOTLIGHT to sketch a sequence of weighted, directed or bipartite graphs. This approach exploits the distance gap in the sketch space and identifies anomalies as the sudden appearance (or disappearance) of a large dense directed subgraph.

2.2.2 Edge Streams

Anomaly detection in graph stream usually focuses on the comparison of the entire graph objects which may ignore some subtle changes. Thus it is necessary to find more fine-grained methods, i.e., transfer to the analysis of edge streams. For anomaly detection on edge streams, the input is a stream of edges over time. In contrast to graph streams, there are relatively few methods that focus directly on the edge streams, but much attention has been attracted in this area and some approaches have achieved significant performance.

Anomalous node detection. This focuses on the dynamic properties of the nodes over time when every edge comes and many methods provide scoring functions to summarize the behavior of each node. Onat et al. [38] introduce a node-based algorithm to observe the arrival processes of each node and maintain short-term dynamic statistics so as to detect anomalous changes in their arrival process. Yu et al. [55] present a localized principal component analysis algorithm called HOTSPOT to analyze the eigenvectors related to local edge correlation patterns. This approach identifies anomalies by detecting the sudden structural pattern changes and significant activity level changes corresponding to any particular node.

Anomalous subgraph detection. Since subgraphs are often related to unusual behavior, some methods process a series of edges coming over time as a subgraph and leverage temporal information for further analyzing structural patterns. Shin et al. [48] propose two incremental algorithms to maintain a dense subtensor in continuous changes in a tensor and then detect the sudden appearances of dense subtensors. Bhatia et al. [13] implement the

algorithms by leveraging dense subgraph search. These methods first map the graphs to a higher-order sketch and then iteratively make use of dense subgraph search to find the densest submatrix and spot graph anomalies.

Anomalous edge detection. Usually anomalous edges show unusual evolving trends compared with most edges in the graph and can be detected based on the scoring function of evolving attributes, such as edge occurrence, edge weights or the addition/removal of edges. Ranshous et al. [42] introduce a high-level model for anomaly detection based on global and local structural properties of a given edge stream. They sketch these structural properties and they make use of edge outlier scoring functions for every coming edge on its historical evidence as well as connectivity patterns, such as edge occurrence, preferential attachment and common neighbors. Eswaran et al. [21] implement a principled randomized algorithm called SEDANSPOT to identify anomalous edges that occur as bursts of activity or connect sparsely parts of a graph. When scoring the anomalousness of an edge, this method leverages the whole graph in the edge stream. Bhatia et al. [12] propose an anomaly detection algorithm called MIDAS, which focuses on detecting microcluster anomalies or sudden groups of suspiciously similar edges in graphs. Bases on edge occurrence after sketching, this paper uses the chi-squared goodness-of-fit test to assign an anomalous score for every edge and takes into account the temporal and spatial relations inside the stream.

Several deep learning based algorithms for graph anomaly detection have also been proposed recently. [15, 30, 34] provide the comprehensive survey in this area. However, most of these methods can not discover anomalies in an online manner and require supervised information to optimize the hyperparameters.

Chapter 3

Preliminaries

In this chapter, we will introduce the concepts and the background knowledge used in this thesis. To start with, remind that it is essential to maintain the historical information of the graph and satisfy the constant memory constraints in an online manner. In data stream mining, two common approaches to solve these problems are sampling and sketching [4], thus we introduce the two techniques used in this thesis. Then we present the concept of link prediction in graph theory and the related measures used for further analysis. Next, we discuss the definition of power law distribution which is found to be pervasive in many fields [25] and this may provide a theoretical explanation for link prediction. After that, we explain the baseline algorithm in detail, since we aim to combine link prediction with its detection and thus improve the performance. We also reproduce the relevant code and give the pseudo code here. At the end of this chapter, we introduce the evaluation criteria that is used to measure the performance of algorithms.

3.1 Reservoir Sampling

A sample is a subset of the target stream, selected to be representative of the larger stream [1]. Since we cannot store the whole information of the graph given the large universe where nodes and edges grow from, sampling retains a selection but important data in the data stream for future analysis. The main problem is how to sample the data in a single pass while remaining agnostic about the order in which the data is viewed. Here we employ reservoir sampling.

Reservoir sampling is a family of randomized algorithms for randomly selecting a small sample of k items from a set with unknown size n in one single process over the items [51, 3]. The size of the population n is a large or unknown number and is particularly suitable for situations where all n items cannot be stored in the main memory. The algorithm sketches the items over time, and cannot look back at previous items.

Assume there is a sequence of items and it comes over one at a time. We want to choose

k items randomly from the sequence and store them in memory. If the total number of items n is known to the algorithm and can be accessed arbitrarily, the solution is simple which can be that choose k distinct indexes i from 1 to n with the same probability. However, the problem is that the exact number n is not always known in advance and the k samples cannot be updated if more items come later. Thus, the main task of this algorithm is

How to take an element at random from a set of unknown size n?
How to remove an element when the number of samples reaches the threshold k?

Since the size n of the data stream may be unknown, to ensure that each sample is extracted with equal probability, the probability of each number being extracted should be $\frac{1}{p}$. Moreover, the probability of deleting any sample from the reservoir is supposed to be the same or follow a specific strategy in the process. Later we give the detail of our reservoir sampling algorithm in section 4.2.

3.2 Count-Min Sketch

The Count–Min sketch was first proposed in 2003 by Graham Cormode and S. Muthu Muthukrishnan [18]. It is a probabilistic data structure capable of summarising a high-dimensional vector and answering queries on this vector, especially frequency queries in data streams, with a strong accuracy guarantee. The main purpose of this sketch is to process a data stream, one at a time, and compute the frequency of different elements in the data stream. Then the frequency of any given element can then be queried from the sketch at any time and an estimated frequency in approximation to the true result will be obtained with a certain probability. Since this data structure efficiently processes updates in additions or subtractions of the vector, it can work at high speeds in the updating stream.

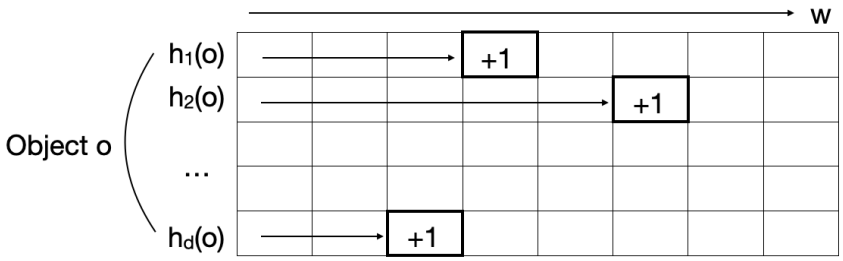

Figure 3.1: The structure of Count-Min Sketch

As can be seen in Figure 3.1, the data structure for Count-Min sketch is a two-dimensional array with width w and depth d, i.e., w columns and d rows, from $CMS[1,1]$ to $CMS[d,w]$. These two parameters w and d are fixed when constructing the sketch which also determines the fixed memory space of the structure. Initially, each element in the array starts with a

value of zero. There are d distinct hash functions,

$$\forall_{i\in\{1,...,d\}} h_i : \{1, ..., n\} \to \{1, ..., w\}$$

each associated with a row, and these hash functions must be pairwise independent.

Figure 3.1 also shows the procedure of the update operation. Consider a stream of objects. When a new object o comes, the sketch will be updated. The hash function h_j corresponding to each row is applied to obtain a column index $k = h_j(o)$, where $j \in d, k \in w$. Set CM_{t-1} to the number of this object before time t. Then, in row j, column k,

$$CM_t[j, k] = CM_{t-1}[j, k] + 1.$$

Similar to update procedure, the query operation accesses the array cells indicated by each hash function and returns the minimum value of the result. Since each hash function requires $O(1)$ computational time, the total time for an update or query is $O(d)$, regardless of the width w. Thus these operations can be efficient.

However, since the space used for this sketch is usually much smaller than the space required for a precise representation, there are some approximations bound to the estimation. Graham Cormode [18] proves that if $w = \lceil \frac{e}{\varepsilon} \rceil$ and $d = \lceil \ln \frac{1}{\delta} \rceil$, the estimate $\hat{a}_i$ has the following guarantees for $a_i \leq \hat{a}_i$, with probability at least $1 - \delta$,

$$\hat{a}_i \leq a_i + \varepsilon \|\boldsymbol{a}\|_1$$

where e is the base of the natural logarithm, i.e., 2.71828..., ε is an additive factor, and $\|\boldsymbol{a}\|_1$ is the stream size, i.e. the total number of items seen by the sketch.

Moreover, this sketch employs hash functions to map events to frequencies, but it uses only sub-linear space, which means that certain events might be overcounted due to collisions. Thus the Count-Min sketch is a biased estimator, i.e., they may overestimate but never underestimate the true frequency. This is why we choose the minimum count when obtaining the result.

3.3 Link Prediction

In graph theory, link prediction is to identify the existence of a link between two nodes in the future [52]. Graphs could be highly dynamic objects that develop and alter rapidly over time as the addition or elimination of the edge, indicating the change of the underlying structure. The mechanism behind this evolution remains a tough problem and the potential information obtained from the graph structure is vital for further analysis.

As has been mentioned before, we intend to identify anomalous edges by leveraging link prediction to estimate the likelihood of the emergence of the incoming edges. However, to what extent can we model the graph and capture the potential information, and how to calculate this likelihood based on the potential information of the graph are key questions. To address these issues, we need to find which proximity measures lead to more accurate link

predictions. Liben et al. [31] discover that a variety of proximity measures make predictions that outperform chance by factors of forty to fifty, implying that future interactions can be inferred from the latent information in the graph structure. Moreover, they mention that some measures that involve infinity sums over graph paths usually outperform the direct measures, such as neighborhood-based link prediction measures. Zhao et al. [56] design different sketches for neighborhood-based link prediction measures and achieve both theoretically guaranteed accuracy and highly accurate empirical results

Therefore, in this paper, we focus on three representative neighborhood-based proximity measures, which are introduced below. More measures can be analyzed and implemented in further researches. To start with, let $\Gamma(x)$ denote the set of neighbors of x in the graph and $|\Gamma(x)|$ defines the number of neighbors of x. Any node x can have zero or infinity neighbors, that is $|\Gamma(x)| \in [0, +\infty)$.

Common neighbors This is a standard method for calculating the number of common neighbors in link prediction but it does not take into account the relative number of common neighbors. Entities that have more neighbors in common are more likely to have a connection, which can be defined as

$$CN(x, y) = |\Gamma(x) \cap \Gamma(y)|.$$

Similar to the number of neighbors of any node, the range of common neighbors is $[0, \infty)$.

Jaccard coefficient It addresses the common neighbor problem by measuring the relative number of common neighbors and the formula is

$$JC(x, y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}.$$

Since the number of common neighbors of two vertices is always less than or at most equal to the total number of neighbors, the range for jaccard coefficient is $[0, 1]$.

Preferential attachment Instead of considering common neighbors, this measure focuses on the total number of neighbors of nodes, which means that nodes with more neighbors are more likely to generate new connections and its formula can be defined as

$$PA(x, y) = |\Gamma(x)| \cdot |\Gamma(y)|.$$

Since link prediction is used for newly arriving edge $e_i = (u_i, v_i, t_i)$, it is natural that u_i and v_i are connected, i.e., neighbors of each other. Thus u_i and v_i both have at least one neighbor, then the range for preferential attachment is $[1, +\infty)$.

3.4 Power Law Distribution

Power law distributions (also knowns as heavy-tail distributions, Pareto distributions, Zipfian distributions, etc.) are currently found to be pervasive in computer science and have attracted extensive research interest [2, 23]. It is a distribution of some variable whose

probability density function is a power function and displays as a linear line in logarithmic coordinates with a slope of the negative of the power exponent.

The earliest discovery of the power-law distribution was in the 19th century, which was summarized by Pareto [39], in the economy domain. Pareto, an Italian economist, studied the statistical distribution of individual income and found that the income of a minority was much higher than that of the majority, proposing the famous 80/20 rule, which states that 20% of the population occupies 80% of the wealth of society. However, it was not until the 1940s that Zipf et al. [57]. discovered this phenomenon in words frequency domain and gave a preliminary mechanical explanation that it was noticed by the academic community. The linguist Zipf, while studying the frequency of English words, found that only a few words were used frequently, and most words were rarely used. A similar rule has been rediscovered in the Internet era. For example, the number of followers of all users on Facebook and Twitter is roughly power-law distributed, i.e., celebrities are only a small number of people but own most of the followers.

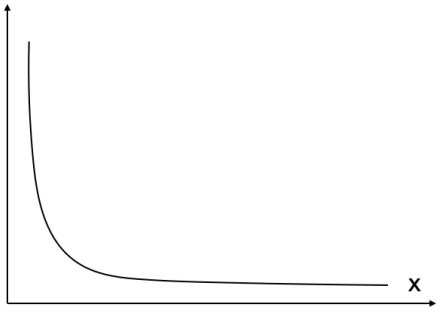

Figure 3.2: Power Law Distribution

Figure 3.2 shows an example of power law distribution. On the right is the long tail, while on the left are the few but dominant ones. For the definition, a nonnegative random variable X is said to have a power law distribution if its probability density function is defined as

$$f(x) \sim cx^{-\alpha-1},$$

and its complementary cumulative distribution is defined as

$$Pr[X \geq x] \sim cx^{-\alpha},$$

where constants $c > 0$ and $\alpha > 0$. In general, the tails fall according to the power α and would be much heavier in the power low distribution when compared with other distributions such as normal distribution.

As has been mentioned before, this paper focuses on graph anomaly detection. Many studies found that in random graphs, indegrees and outdegrees of vertices are pow-law distributed. Barabasi et al. [8] discover that large networks with a complex topology tend

to follow a scale-free power-law distribution for the vertex connectivities. Cooper et al. [17] describe that a general random graph model whose proportional degree sequence obeys a power law.

For a random graph, let us start with a single node. A new node with outdegree 1 appears at each time step. With probability $\alpha < 1$, the edge connects with a node chosen uniformly at random. With probability $1 - \alpha$, this new edge connects with a node chosen according to the indegree of the nodes. This model could be an example of the preferential attachment which we have discussed in section 3.3. That is, a new node is more likely to connect with nodes with higher indegree [35].

3.5 Midas Algorithm

There are many important anomaly detection algorithms in past decades, but here we will focus on MIDAS and introduce this algorithm in detail, as our algorithm aims to combine graph theory with its detections and then improve the performance. In 2020, Bhatia et al.[11] proposed an anomaly detection algorithm called MIDAS, which stands for Microcluster-Based Detector of Anomalies in Edge Streams. As the name suggests, MIDAS paper focuses on detecting microcluster anomalies or sudden groups of suspiciously similar edges in graphs. This algorithm stores the occurrence of edges in the stream and them use this statistical information to compute anomaly scores for every edge. As a recent algorithm, it outperforms many state-of-the-art approaches and thus could serve as a new baseline for anomaly detection.

Data structure This algorithm is online, which means processing each edge in constant time and constant memory. To ensure the real-time performance of the algorithm, it is impossible to store all the graphical data information, as this would take up large of memory and computation time. Thus Count-Min sketch (CMS) data structures, which introduced in section 3.2, are used to store the statistical information in MIDAS and can be updated in constant time when processing the stream. Assume time is a discrete variable, then at any time, an approximate count can be obtained from the data structure. Specifically, it uses s_{uv} to store the total number of edges from u to v up to the current time and a_{uv} to store the number of edges from u to v in the current time.

Distribution assumption The common idea would be to assume the normal data follow a specific distribution and account for a large proportion, while the anomaly data has a relatively large deviation in comparison. Then, the distribution likelihood can be computed with the number of edge occurrences in the current timestamp and be defined as an anomaly if this likelihood is beyond its threshold. However, this would require data following a strict distribution, but different anomalies tend to have different distributions and even data in the same type could have different distributions over time. Therefore, MIDAS adopts a weaker assumption in the process, that is, the mean level (i.e. the average rate at which edges appear) remains the same for the current time and all past time, which avoids a strict distributional assumption and can be adapted to different data types.

Recall that the data structures and the time variable is assumed to be discrete, this assumption means that processed edges can be divided into two parts, the number of edges that come in time $t = t_i$ which can be represented by a_{uv}; the number of edges that come before time $t < t_i$ which can be regarded as the number of edges in time $t_i - 1$ and represented by $s_{uv} - a_{uv}$.

Anomaly scores As has been mentioned before, it maintains two CMS data structures to store the number of past edges, the current time and the time before. Then under the chi-squared goodness-of-fit test, the chi-squared statistic is defined as

$$\frac{(observed_{(t=t_i)} - expected_{(t=t_i)})^2}{expected_{(t=t_i)}} + \frac{(observed_{(t<t_i)} - expected_{(t<t_i)})^2}{expected_{(t<t_i)}}.$$

Combining this statistic with the mean level assumption, for any new coming edge $e_i = (u_i, v_i, t_i)$, anomaly scores could be computed as

$$\text{score}(e_i) = \left(\hat{a}_{u_i v_i} - \frac{\hat{s}_{u_i v_i}}{t_i}\right)^2 \frac{t_i^2}{\hat{s}_{u_i v_i}(t_i - 1)},$$

where $\hat{a}_{u_i v_i}$ and $\hat{s}_{u_i v_i}$ are the approximate count obtained from CMS structures a_{uv} and s_{uv}. Note that the score indicates the extent to which the observation deviates from the expectation, i.e. mean level, so the higher the score, the more anomalous it is.

In their algorithms, the anomaly score is computed under the chi-squared goodness-of-fit test. They first introduce the basic algorithm MIDAS and then further implement MIDAS-R which incorporates temporal and spatial relations.

MIDAS This algorithm computes anomaly scores for every edge, but this score does not determine whether the edge is an anomaly or not. Instead, a threshold is usually required to make a binary determination. Besides, its probabilistic guarantee can be viewed in the original paper for proof and details.

Algorithm 1: MIDAS

input : An edge stream E over time
output: Anomaly scores for every edge

Initialize CMS data structures for the total count s_{uv} and the current count a_{uv};
Initialize a time variable t;
for *every coming edge* $e_i = (u_i, v_i, t_i)$ *in the stream* **do**
    if $t_i > t$ **then**
        Initialize the CMS data structure for current count a_{uv};
    Update the CMS data structures for s_{uv} and a_{uv};
    Retrieve $\hat{s}_{u_i v_i}$ and $\hat{a}_{u_i v_i}$ for this edge e_i;
    output $midas(e_i) = \left(\hat{a}_{u_i v_i} - \frac{\hat{s}_{u_i v_i}}{t_i}\right)^2 \frac{t_i^2}{\hat{s}_{u_i v_i}(t_i - 1)}$

MIDAS-R This approach takes spatial and temporal relationships into account, which means that it does not only consider the occurrence of the coming edge as in the previous

algorithm, but also other edges associated with the nodes of this edge, while also considering the temporal factor.

Temporal relations In the previous algorithm, a_{uv} only represents the occurrence of the coming edge in the current timestamp. However, if taking the time factor into account, the edges occurring in the past time, while not as important as the edge of the present, may also have some impact. Thus they use a reduced weight $\alpha \in (0, 1)$ to modify the sketches. That is, whenever each new timestamp arrives, instead of initializing the data structure a_{uv}, a fixed α will be multiplied on the sketch a_{uv} as the decay factor. Then, a range from 0 to 1 determines its decay weight, where 0 represents the removal of all previous effects and 1 represents no decay applied.

Spatial relation Instead of just considering the coming edges, spatial relationships mean that possible anomalies are also taken into account, i.e. observe the sudden appearance of a large group of edges nearby. They assume that nodes of the coming edge could be used as observations of anomalies nearby, e.g. a node suddenly having connections with many other nodes. To store information about the nodes, they create new CMS data structures s_u and a_u to estimate the total number of edges and the current number of edges associated with node u. Then the chi-square test can be applied in three pairs, (a_{uv}, s_{uv}) for the edge occurrence, (a_u, s_u) for the for the source node u and (a_v, s_v) for the source node v. In the end, three anomaly scores are computed and the maximum is selected.

Complexity For memory constraint, since these two algorithms only maintain the CMS data structures when processing edges over time, the requirement of memory is $O(wb)$, where w and b are the numbers of the hash functions and buckets in the CMS data structures, respectively. See section 3.2 for more information. In terms of time complexity, the update and query steps in the algorithms would take $O(w)$, while other operations only require constant time, resulting in total time complexity of $O(w)$.

Weakness Like other algorithms, MIDAS only focuses on the occurrence of the edges and uses this information to find anomaly microclusters, but this CMS data structure may be hard to store the connective relationships in the graph, e.g., the information of their neighbors. Therefore, we intend to utilize this baseline score and then leverage graph information, i.e. link prediction, to further update this score and improve the performance.

Algorithm 2: MIDAS-R

input : An edge stream E over time
parameter: A decay factor α
output : Anomaly scores for every edge

Initialize CMS data structures for the total count s_{uv} and the current count a_{uv};
Initialize CMS data structures for the total count s_u, s_v and the current count a_u, a_v;
Initialize a time variable t;
for *every coming edge* $e_i = (u_i, v_i, t_i)$ *in the stream* **do**
  if $t_i > t$ **then**
    Updated the CMS for current count a_{uv}, a_u, a_v with the decay factor α;
  Update the CMS data structures for the edge s_{uv} and a_{uv};
  Update the CMS data structures for two nodes s_u, s_v and a_u, a_v;
  Retrieve edge counts $\hat{s}_{u_i v_i}$ and $\hat{a}_{u_i v_i}$;
  Retrieve node counts $\hat{s}_{u_i}, s_{v_i}$ and $\hat{a}_{u_i}, \hat{a}_{v_i}$;
  Compute $score(e_i), score(u_i), score(v_i)$ for edge e_i and nodes u_i, v_i;
  $score(e_i) = \left(\hat{a}_{u_i v_i} - \frac{\hat{s}_{u_i v_i}}{t_i}\right)^2 \frac{t_i^2}{\hat{s}_{u_i v_i}(t_i - 1)}$;
  $score(u_i) = \left(\hat{a}_{u_i} - \frac{\hat{s}_{u_i}}{t_i}\right)^2 \frac{t_i^2}{\hat{s}_{u_i}(t_i - 1)}$;
  $score(v_i) = \left(\hat{a}_{v_i} - \frac{\hat{s}_{v_i}}{t_i}\right)^2 \frac{t_i^2}{\hat{s}_{v_i}(t_i - 1)}$;
  output $midas(e_i) = \max\{\text{score}(e_i), \text{score}(u_i), \text{score}(v_i)\}$;

3.6 Evaluation

Performance measurement is an essential task when evaluating our algorithm and analyzing the results. In order to estimate and compare the performance of all these anomaly detection algorithms, a unified metric is needed. In this paper, we apply Area Under the Curve (AUC) of Receiver Operating Characteristics (ROC) curve. Before introducing this metric, we first introduce some measures.

TP: the number of predictions that are correctly classified as positive classes by the model, known as true positive.

TN: the number of predictions that are correctly classified as positive classes by the model, known as true negative.

FP: the number of predictions that are wrongly classified as positive classes by the model but belong to the negative class, known as false positive.

FN: the number of predictions that are wrongly classified as positive classes by the model but belong to the positive class, known as false negative.

TPR: the proportion of the positive class that is correctly classified by the model, known

as true positive rate and defined as

$$\frac{TP}{TP+FN}.$$

FPR: the proportion of the positive class that is wrongly classified by the model, known as false positive rate and defined as

$$\frac{FP}{FP+TN}.$$

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 Figure 3.3.

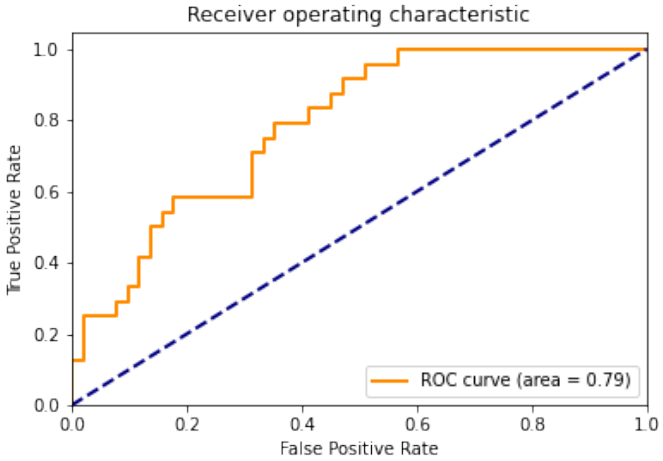

Figure 3.3: ROC Curve

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

Algorithms

In this chapter, the anomaly detection algorithms developed in our research are described in detail. First, we present an offline approach to find how the three selected measures of link prediction affect the baseline algorithm. Then, we switch to online approaches. A basic online algorithm without sampling is presented, which can determine the impact of the sample on performance, and then reservoir sampling is used to address the limitation of the memory constraint.

4.1 Offline Approach

This algorithm is our first approach to gain an intuition about how link prediction performs and affects accuracy. For observation, it is assumed that the edges in the datasets and the ground labels of these edges are known in advance, because we intend to use this information to construct a graph without anomalous edges and then obtain information from the graph for link prediction later. Sampling is also not applied in this approach. Therefore, this algorithm can not process the edge streams in an online manner, but we can gain some insights for our later algorithms.

To start with, we assume that the input is a set of edges E. Each edge $e_i \in E$ is a tuple $e_i = (u_i, v_i, t_i)$ where u_i and v_i are nodes and t_i is the timestamp (i.e. occurrence time). Based on the timestamps T, we first divided the dataset into training and testing datasets. The graph is then constructed using the training dataset. Since we assume the edges and their ground labels are known, when constructing the graph G, normal edges in the training dataset are added sequentially, while the anomalous edges are discarded and not added to the graph. This graph G is fixed and will not be updated after the training dataset has been processed. In the testing dataset, we obtain a link prediction score $pred(e_i)$ for each new coming edge and combine it with the baseline score $midas(e_i)$ to get the final anomaly score.

Our goal is to analyze the underlying information from the dynamic graph so as to

estimate the likelihood of the occurrences of the coming edges. That is, if this edge has a high link prediction score, which implies a higher probability that these two nodes are related, then this coming edge is more likely to be a normal edge. Recall the measures mentioned in section 3.3, we define the algorithm for calculating the link prediction score $pred(e_i)$ as algorithm 3.

Algorithm 3: LinkPrediction(G, e_i)

input : an graph G, an edge $e_i = (u_i, v_i, t_i)$
output: link prediction score $pred(e_i)$ for the input edge e_i

Let $\Gamma(x)$ denote the set of neighbors of node x in the graph G;
procedure `Retrieve the set of neighbors`;
Retrieve the neighbor set $\Gamma(u_i)$ of u_i from graph G;
Retrieve the neighbor set $\Gamma(v_i)$ of v_i from graph G;
procedure `Compute the measures in link prediction`;
if *Common Neighbors* **then**
    $pred(e_i) = |\Gamma(u_i) \cap \Gamma(v_i)|$;
else if *Jaccard Coefficient* **then**
    $pred(e_i) = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$;
else if *Preferential Attachment* **then**
    $pred(e_i) = |\Gamma(x)| \cdot |\Gamma(y)|$;
output $pred(e_i)$;

Then we formally define our offline approach as algorithm 4. This algorithm can be divided into three parts, split the edge stream into training and testing dataset based on the timestamp, build the graph from the training dataset and compute the anomaly scores for edges in the testing dataset. For the input, the edge stream E can be any dataset for anomaly detection in which the records are processed in the form $e_i = (u_i, v_i, t_i)$, while the timestamp t is served as a boundary to divide the dataset and can be any number within the interval of the dataset, but for the experiments we choose to divide the timestamps of the dataset into the same ten parts, i.e., first ten percent for training and ninety percent for testing, then twenty percent for training and eighty percent for testing, and so on.

After spliting the dataset, the fixed graph G used to obtain link prediction score is constructed in the training dataset, then the anomaly score is computed by combining these two scores $midas(e_i)$ and $pred(e_i)$. Note that $midas(e_i)$ represents the the extent to which the observation deviates from the expectation, while $pred(e_i)$ indicates the likelihood of the occurrences of the coming edges, which means the arriving edge with higher $midas(e_i)$ and lower $pred(e_i)$ is more likely to be anomaly. Therefore the inverse of $pred(e_i)$ is used to reduce the corresponding anomaly score of this edge.

Algorithm 4: Offline Approach

input : an edge stream E containing edges $e_i = (u_i, v_i, t_i)$
parameters: the split timestamp t
output : anomaly scores for each edge

procedure `Split the edge stream into training and testing dataset based on the timestamp`;
for *every edge* $e_i = (u_i, v_i, t_i)$ *in the dataset* **do**
    if $t_i <= t$ **then**
        Add e_i to the training dataset;
    else
        Add e_i to the testing dataset;
procedure `Build a graph` G `from the training dataset`;
Initialize an empty graph G;
for *every edge* $e_i = (u_i, v_i, t_i)$ *in the training dataset* **do**
    if e_i *is not anomaly* **then**
        Add e_i to the graph G;
procedure `Compute the anomaly scores for edges in the testing dataset`;
for *every edge* $e_i = (u_i, v_i, t_i)$ *in the testing dataset* **do**
    Retrieve the score $pred(e_i)$ from G using `LinkPrediction` (G, e_i);
    Retrieve the baseline score $midas(e_i)$ for this edge e_i;
    output $score(e_i) = midas(e_i)/pred(e_i)$;

4.2 Online Approach

The purpose of graph anomaly detection is to detect anomalous behavior in the real world, thus in terms of time feature, i.e., in near real-time, it is crucial to our detection. The value of a newly discovered intrusion attack or credit fraud is in the moment and needs to be fixed as soon as possible, not a week later. Moreover, since the nodes and edges will be updated in the real universe, we need to update and query the information in sublinear memory, instead of storing a counter for every edge and node. Therefore, we implement an online approach and its variants to answer the research question posed in chapter 1.

We first present a basic online algorithm, which can process arriving edges and store their information in real-time. Although the graph is not sampled at this point, its performance can be compared with the subsequent algorithm to determine the impact of the sample on performance. The detail is shown in algorithm 5, which can be divided into three procedures, update the graph, retrieve the scores and compute the final anomaly score for every arriving

edge.

Algorithm 5: Online Approach without Sampling

input : an edge stream E over time
output: anomaly scores for each edge
Initialize an empty graph structure G;
for *every edge* $e_i = (u_i, v_i, e_i)$ *in the stream* **do**
  procedure `Update the graph`;
  Add e_i to graph G;
  procedure `Retrieve the scores`;
  Retrieve the prediction score $pred(e_i)$ from `LinkPrediction`(G, e_i);
  Retrieve the baseline score $midas(e_i)$ for this edge e_i;
  procedure `Compute the anomaly score in two ways`;
  1) $score(e_i) = 1/pred(e_i)$;
  2) $score(e_i) = midas(e_i)/pred(e_i)$;
  output $score(e_i)$

When updating the graph, we simply add the edge to the modeled graph without sampling. Then the link prediction score can be obtained from algorithm 3 and the baseline score will be computed from algorithm 1 or algorithm 2. As for computing the anomaly score, we provide two ways, one only considers the influence of link prediction, that is, the likelihood of the occurrence of the coming edge, and uses its inverse, which means an edge with a higher score is more likely to have less possibility to occur. While the other one is similar to the offline approach, combined with baseline score, thus this can detect the sudden microcluster anomalies and also take the connectivity patterns into account. Take preferential attachment as an example, if at one moment, an edge suddenly occurs many times but the two nodes of this edge have relatively more neighbors, then we will reduce the baseline score using the connectivity information because of the relatively high probability that this edge appears.

Next, to address the limitation of the offline approach, we use reservoir sampling to meet the memory constraint, which is introduced in section 3.1. Here we defined the algorithm with reservoir sampling as algorithm 6, containing three procedures, reservoir sampling, retrieve the scores and compute the anomaly score. The last two parts are the same with the previous algorithm. Therefore we focus on the first procedure, reservoir sampling.

The algorithms starts with an initialized graph G with the sample size k and the probability p where k and p are fixed for every edge in one edge stream, and then each edge e_i is entered into the algorithm in sequence. The s and p should be an integer, greater than or equal to 1. If $p = 1$, this means that no sampling is applied, just a graph with the most recent k edges is maintained. Then for reservoir sampling, when processing each edge e_i, a number r is first randomly obtained from $(1, ..., p)$ where the probability of each number selected should be the same. In the conditional statement about random number r in this algorithm, we should make $r = 1$ (1 can be replaced by any number from $1, ..., p$)

Algorithm 6: Online Approach with Reservoir Sampling

input : an edge stream E over time
parameters: the sample size k, the probability p
output : anomaly scores for each edge

```
1  Initialize an empty graph structure G with sample size k and the probability p;
2  for every edge e_i = (u_i, v_i, e_i) in the stream do
3      procedure Reservoir Sampling (G, e_i);
4      Randomly obtain a number r from (1, ..., p);
5      if r = 1 then
6          Retrieve the current size s of the graph G;
7          while s >= k do
8              Delete a sample from the graph G;
9              1) Random deletion;
10             2) Temporal deletion;
11             Update the current size s;
12         Add the edge e_i to the sample graph G;
13     procedure Retrieve the scores;
14     Retrieve the prediction score pred(e_i) from LinkPrediction(G, e_i);
15     Retrieve the baseline score midas(e_i) for this edge e_i;
16     procedure Compute the anomaly score in two ways;
17     1) score(e_i) = 1/pred(e_i);
18     2) score(e_i) = midas(e_i)/pred(e_i);
19     output score(e_i)
```

consistent throughout the processing of the whole data stream, so that each edge has the same probability $1/p$ of being sampled. Then in line 8, there is a delete step if the graph G reaches the sample size k. We define two deletion methods:

- **Random Deletion** Randomly delete an edge in the graph, ensuring that the probability of each edge being selected is the same.

- **Temporal Deletion** Take into account the time factor, i.e., edges appearing at an earlier time may have relatively little relevance to the present. Delete according to time, removing the edge that was added to the graph earliest.

After reservoir sampling, the remaining two parts are the same as the previous algorithm and we do not introduce them in detail here.

4.3 Time and Memory Complexity

Here we only discuss the time and memory complexity of the online algorithm with reservoir sampling and without combining the baseline algorithm, because of its representativeness.

In terms of time complexity, the reservoir sampling procedure for updating the graph takes $O(1)$ for every update step, and the last score computing part also takes $O(1)$. However, for the second procedure to retrieve the link prediction scores, these three measures are not the same. For *preferential attachment*, only the degree (i.e. the number of neighbors) of nodes is required. The query step for obtaining the degree of each node take $O(1)$, thus *preferential attachment* takes $O(1)$ in total. Then for *common neighbors* and *jaccard coefficient*, the query step for obtaining the set of neighbors takes $O(m)$, where m is the number of neighbors. Since we sample the graph with size k and probability p, the maximum number of neighbors would be $k-1$. Also there are many more comparison steps to find the number of common neighbors and distinct neighbors for *common neighbors* and *jaccard coefficient*, these two measures may need perhaps $O(k^2)$ at worst. Then for memory complexity, only a sample graph is maintained over time, which takes $O(k)$, where k is the size of the sample graph.

Chapter 5

Experiment

In this section, we begin with the details of the datasets and experiment setup. Then we evaluate the performance of our algorithm compared to the baseline method. Moreover, we discuss the scalability of our algorithm. Importantly, how accurate is our algorithm in detecting anomalies in datasets with ground truth labels? How well do our algorithms improve on the baseline? How does accuracy depend on the parameters of reservoir sampling? How is the scalability of our algorithms? We will answer these questions in the following.

5.1 Datasets

We use five real world datasets in our experiments to evaluate the performance of our approaches compared to the baselines. All these datasets have ground labels, i.e., to determine whether this edge is anomalous or not, so we can use the evaluation metric mentioned in section 3.6 to calculate the ROC/AUC of algorithms and thus compare the performance.

Table 5.1 shows the statistical information of the datasets, where $|V|$ represents the number of unique nodes and $|E|$ is the total number of edges that occur in all timestamps $|T|$. For all the datasets, timestamps are modified to start from 1 to simplify the processing, and the same timestamp is used for edges that arrive at the same time.

Datasets	$\|V\|$	$\|E\|$	$\|T\|$	Anomalies
CIC-DDoS2019 [47]	1,290	20,364,525	12,224	99.72%
DARPA [33]	25,525	4,554,344	46,572	60.10%
CIC-IDS2018 [46]	33,176	7,948,748	38,478	7.27%
CTU-13 [24]	371,076	2,521,286	33,090	4.64%
UNSW-NB15 [36]	50	2,540,047	85,348	0.64%

Table 5.1: Statistical information of the datasets

5.2 Experimental setup

All the algorithms are implemented in Python and experiments are performed on macOS Catalina with a 2.9 GHz Intel Core i5 processor and 8GB main memory.

5.3 Performance

Here we would like to show the performance of our algorithms compared to the baselines. In the order of chapter 4, the performance of the offline method is presented first, followed by the online approach and its variants.

To start with, recall the offline algorithm introduced in section 4.1, the dataset will be divided into the training dataset and the testing dataset. Then a graph is constructed on the training dataset and the link prediction score is computed from the constructed graph for every edge in the training dataset. The final anomaly score is obtained based on the baseline score and the link prediction score, where the baseline score is from algorithm 2 and the link prediction score is from algorithm 3.

Figure 5.1 shows the ROC/AUC of the offline approach for the four datasets. For each dataset, the figure above indicates the accuracy and the blue horizontal line represents the accuracy of the baseline itself without combing link predictions, and then the bottom figure indicates the percentage of anomaly edges in the testing dataset. The percentage of anomaly edges in the training dataset is not given because the anomaly edges are not used to construct the graph and do not influence the result. As has been mentioned before, we divide the timestamps of the dataset into the same ten parts, thus for the x-axis, 0:10 means there is no training dataset, 1:9 means first ten percent for training and ninety percent for testing, 2:8 means twenty percent for training and eighty percent for testing, and so on.

As can be seen in Figure 5.1, in some datasets, the accuracy line for *common neighbors* seems invisible, and that is because it largely overlaps with the line of *jaccard coefficient*. Since the computing formulas for *common neighbors* and *jaccard coefficient* are similar, i.e., both require the number of common neighbors, it is understandable that they are similar in accuracy. This may also mean that the total number of distinct neighbors of two nodes, which is additionally required by *jaccard coefficient*, has little effect on accuracy.

Apart from that, *preferential attachment* performs best among three measures, exceeding the baseline score in most cases and remaining over 0.8 even in the worst case. We believe that the power law distribution may serve as an explanation for why this measure performs well in the graph. The detail is given in section 3.4. If the *preferential attachment* of an edge is large, it means that the two nodes of this edge have many neighbors. Take social networks as an example. If two people have many friends respectively, then their social circles are very likely to intersect and there is a high probability that the two people will meet later. This also follows the idea of the power law distribution, i.e., a small group of nodes will occupy the most connections, and thus the more neighbors it has, the higher probability it will have new connections.

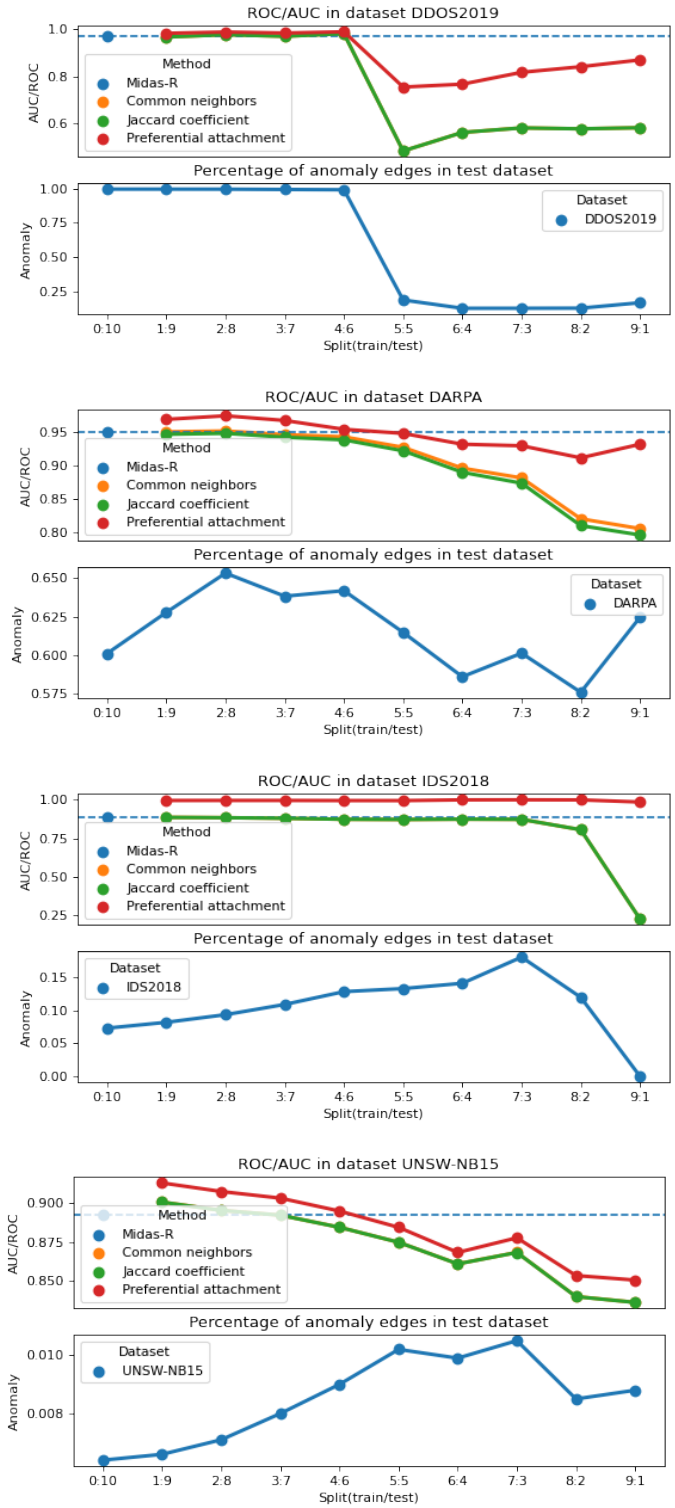

Figure 5.1: The ROC/AUC of the offline approach

As for the percentage of anomalous edges, it seems to have some correlation with accuracy. Although the trend is roughly opposite in the last dataset, they go very similarly in most of the datasets. However, we did not find a relevant reason for this, which could perhaps be studied as a follow-up.

There is no figure for dataset `CTU13` here because we did not finish the running part. For each splitting method, it may take several hours for one measure, i.e., *common neighbors* or *jaccard coefficient*. Referring to the time complexity of *common neighbors* and *jaccard coefficient* in section 4.3, dataset `CTU13` has 0.37M nodes and even we can store the graph of the whole dataset, it is slow to go through the graph to compute *common neighbors* or *jaccard coefficient* for so many nodes. Given the large universe, there may even be more nodes and edges. This also explains the importance of sampling the graph.

Next, we discuss the performance of our online approaches introduced in section 4.2. It mainly contains three procedures when processing the arriving edge, updating the graph with or without sampling, retrieving the scores, and then computing the anomaly score. In reservoir sampling, two deletion methods are used, i.e., random deletion and temporal deletion. In terms of retrieving the scores, the link prediction score is retrieved from algorithm 3 and the baseline score is retrieved from algorithm 1 or algorithm 2. As for the anomaly score, two computing formulas are given, thus we will try different permutations for the baseline score and the measure scores. Since *preferential attachment* outperforms other two measures and *common neighbors* and *jaccard coefficient* would take much more time to be computed, here we only show the results related to *preferential attachment*.

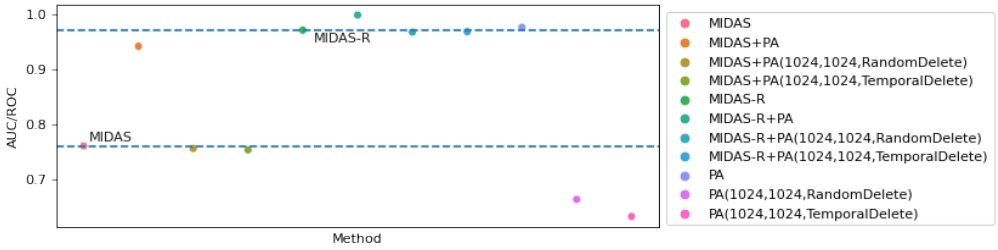

Figure 5.2: The ROC/AUC of the online approach for dataset `DDOS`

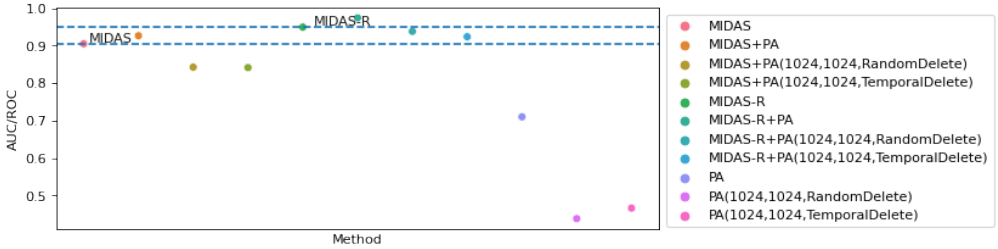

Figure 5.3: The ROC/AUC of the online approach for dataset `DARPA`

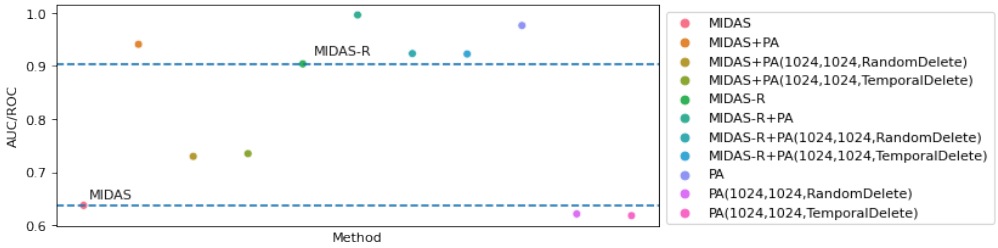

Figure 5.4: The ROC/AUC of the online approach for dataset `IDS`

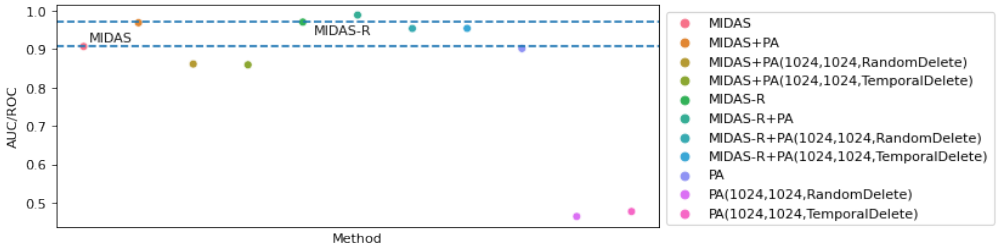

Figure 5.5: The ROC/AUC of the online approach for dataset `CTU13`

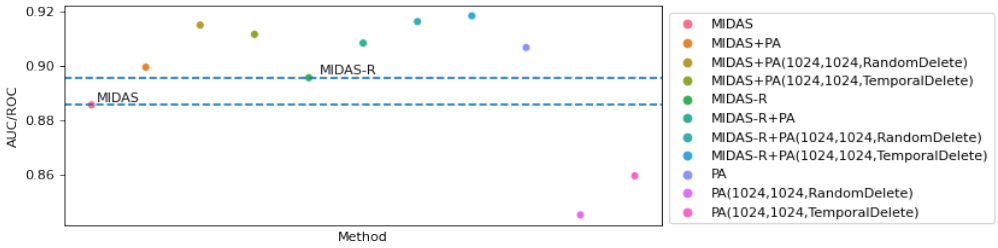

Figure 5.6: The ROC/AUC of the online approach for dataset `NB15`

Figures 5.2, 5.3, 5.4, 5.5, 5.6 show the ROC/AUC of the online approach for each dataset, separately. The points in the figures indicate the accuracy of each method in the experiment, denoted by different colors, and the blue line represents the accuracy of the baseline algorithms, as the comparisons of our improved algorithm. In the figure legend, the methods in the experiment are given. Note that the $'+'$ refers to the combination of two scores defined in the algorithm, and does not represent addition in mathematical operations. Moreover, (k, p, d) is used to describe the parameters of the reservoir sampling algorithm, where k represents the size of the sampling graph, $1/p$ represents the probability of the arriving edge being sampled, and d represents the deletion method. Take $PA(1024, 1024, RandomDelete)$ as an example, the size of the sample graph is 1024, the probability of sampling for new coming edge is 1/1024 and random deletion is used.

As can be seen in these figures, after combining the baseline score with *preferential attachment*, the accuracy is always better than the original ones, which means that *preferential*

attachment indeed plays an important role in the graph and thus can be useful for graph anomaly detection. Meanwhile, even without combining the baseline algorithm, *preferential attachment* itself achieves high accuracy in the real world datasets, which also validates the significance of *preferential attachment*. As for after reservoir sampling, the accuracy of the two deletion methods is similar, thus it seems that there is no difference in the effect of random deletion and temporal deletion. In some datasets, even after sampling, our algorithm can also improve the performance of the baseline, and although it reduces the accuracy for some datasets, it still maintains a high level. But still, we can tell that the performance of combining baseline and link prediction is the best and this also confirms our claim that *preferential attachment* as a measure in link prediction reflects the the possibility of the arriving edge appearing, i.e., what we expect to occur, and thus the anomaly score of this edge should be reduced accordingly.

Recall the meaning of *preferential attachment*, which is the product of the number of neighbors of the two vertices of the coming edge. After we find the meaning of *preferential attachment*, we also try to extend *preferential attachment* to the 2 degrees, that is, find the number of neighbors of the neighbors of the two vertices of the arriving edge, because we want to know whether this extension is meaningful for the performance. The result is not good. This could indicate that similar extensions may not make sense.

Next, we study the parameters k and p to determine if they have an effect on performance, where k is the size of the graph and p is the probability of the arriving edge being sampled. The method of $MIDAS + PA(k, p, RandomDelete)$ is used in this experiment and we use control variates, i.e., keep one parameter unchanged and vary the other one to compare the results. The results are shown in Figure 5.7 and Figure 5.8. In Figure 5.7, we keep the size k as 1024 unchanged, and change p from $2^1(2)$ to $2^{10}(1024)$. In Figure 5.8, we keep the probability p as 1024 unchanged, and change k from $2^5(32)$ to $2^{14}(16384)$. There is no clear uniform pattern in the figure, so we may conclude that the change of these two parameters has little effect on the final performance, at least at the scale of the dataset we tested. This means that our experiments can be scaled to much larger datasets by using different sample size k or different probability p.

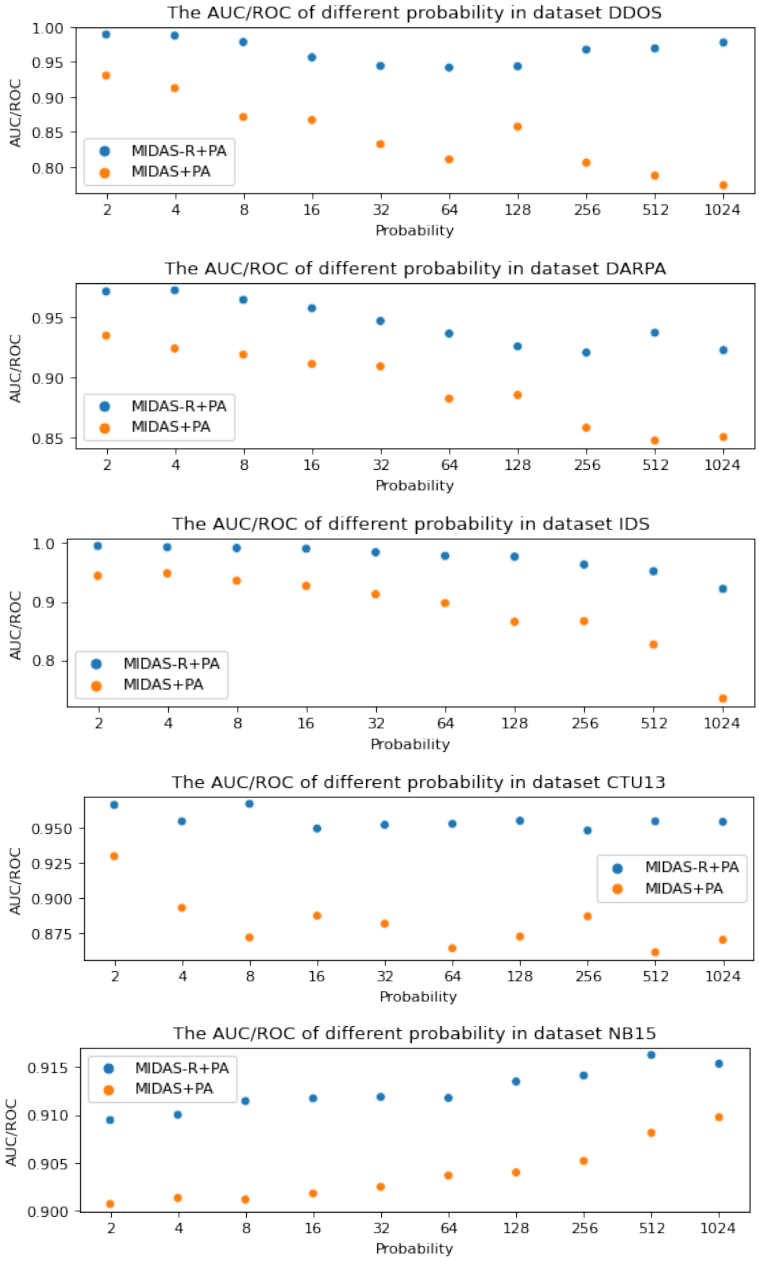

Figure 5.7: The ROC/AUC of different probability

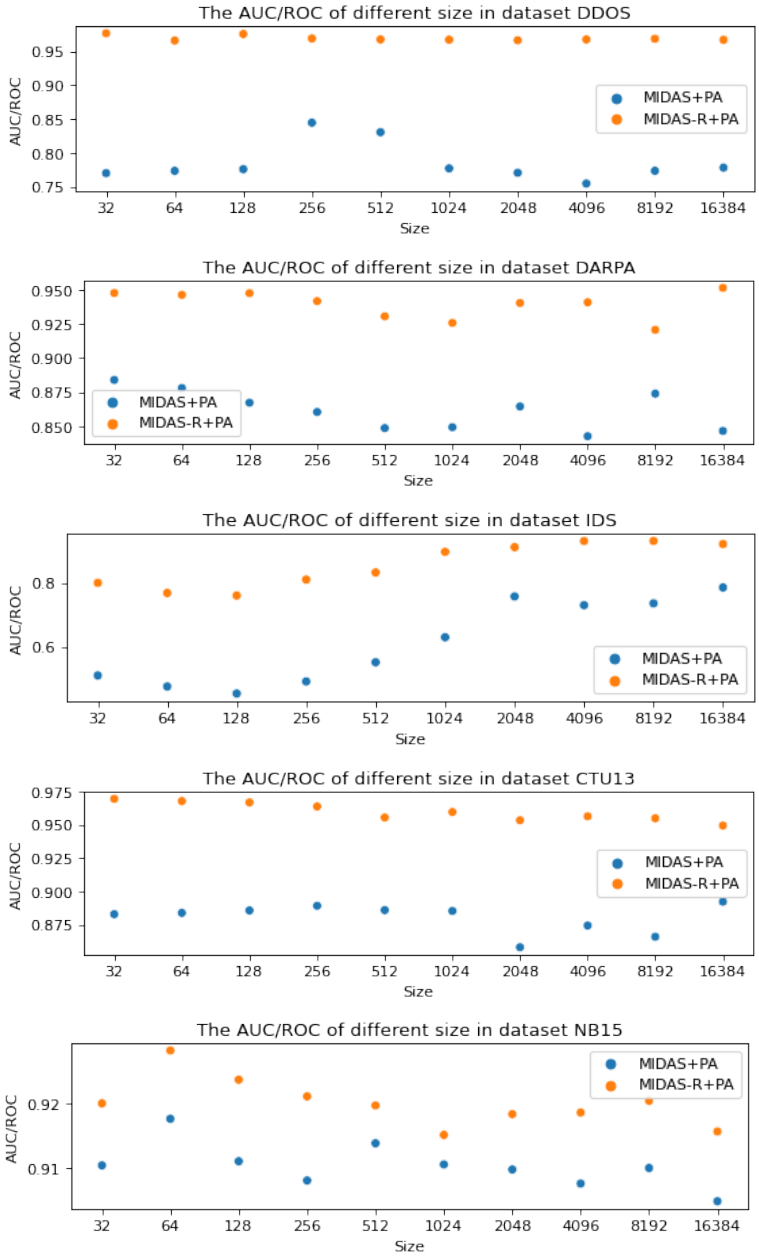

Figure 5.8: The ROC/AUC of different size

5.4 Scalability

Here we discuss the scalability of our algorithms to determine whether it is suitable for scaling to larger datasets.

Figure 5.9 shows the running time of different size in our online approaches. In the figure legend, PA is an abbreviation of the method $PA(k, 1024, RandomDelete)$, where the probability is 1024 and the sample size k is range from $2^5(32)$ to $2^{14}(16384)$, and $'+'$ means combining *preferential attachment* with the baseline algorithm $MIDAS$ or $MIDAS-R$. We can see that the runtime is not affected by the sample size in any of the three approaches, thus the sample size is not an influencing factor of time complexity, at least not at the current scale of the datasets. This could prove our discussion about the time complexity of *preferential attachment*, which takes $O(1)$ for every arriving edge, regardless of the sample size.

Figure 5.10 presents the scalability of edge records. We test the required time to process the first 10^4 to 10^6. The three methods in the illustration are very similar in meaning to the previous ones, except that $PA(1024, 1024, RandomDelete)$ is used to represent PA. It shows that the running time of our algorithms is linear to the number of the arriving edges on the five datasets. Thus we can confirm that the time complexity of our approaches to process any individual edge is constant.

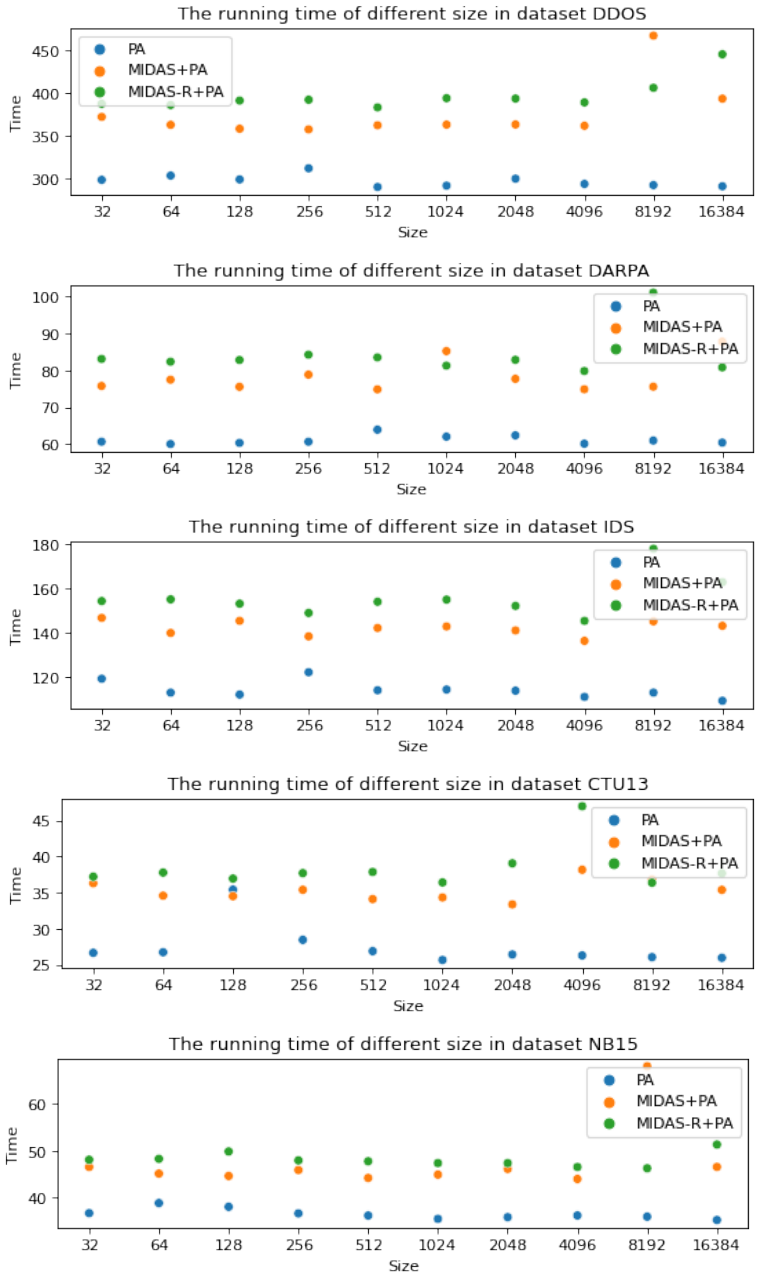

Figure 5.9: The running time of different size

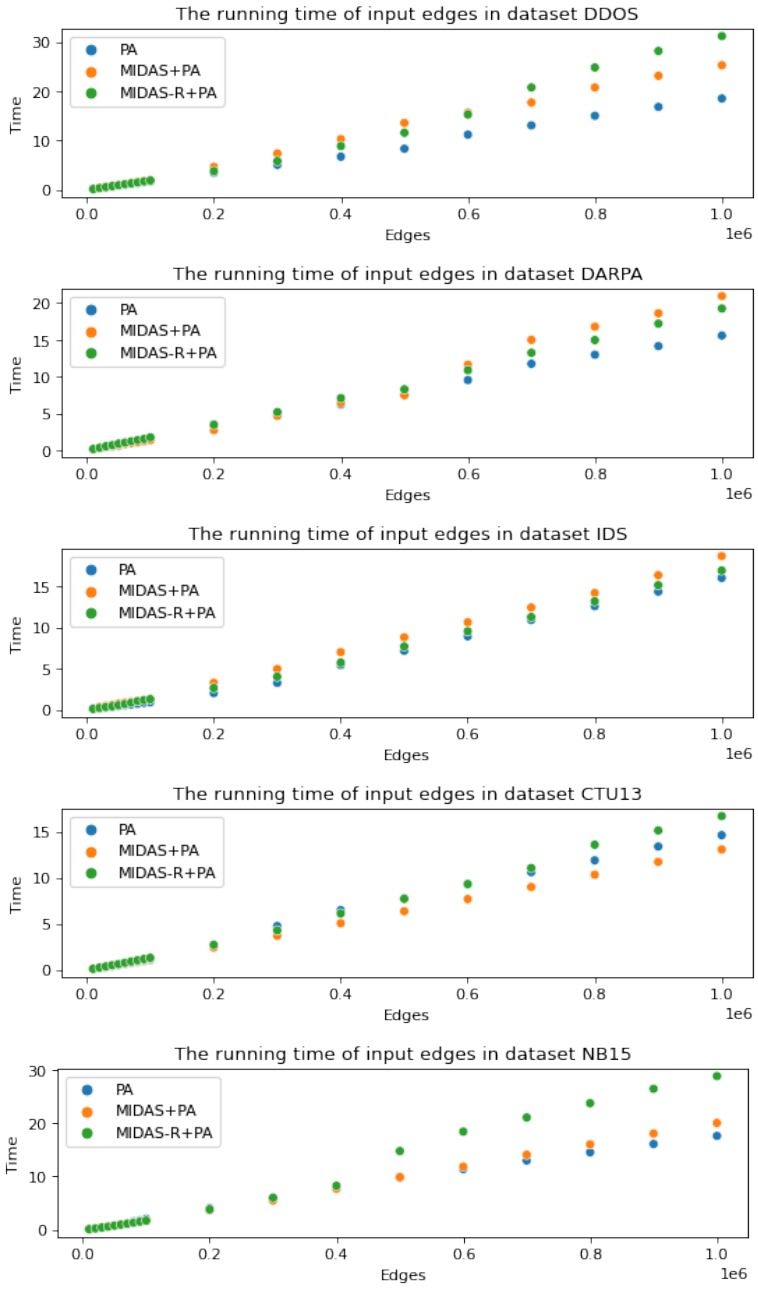

Figure 5.10: The running time of input edges

Chapter 6

Conclusions

In the last chapter, all the results achieved from the previous chapters are summarized. At first, we present the extent to which we have answered the research question mentioned in section 1.1, as well as the performance of our algorithms. Then, we discuss the possible future research directions.

We consider the problem of graph anomaly detection given a stream of edges, where anomalies are edges with a low probability of occurrence as determined by link prediction. Moreover, it is essential to process the newly arriving edges in constant time and memory, given the unbounded number of edges and vertices in the real world. To meet the above requirements, we implement our algorithms with link prediction measures to detect anomalies, which can run in an online manner with constant memory and update the arriving edge in constant time. We ensure the time and memory complexity by reservoir sampling. Experimental results on five real-world datasets show that our algorithms outperform the baseline approach and demonstrate the scalability effectiveness, i.e, the time complexity of processing an individual edge remains constant with respect to the data scale.

However, we only use some measures from the link prediction. Since the importance of link prediction in the graph is also demonstrated in this thesis, more measures could be considered for future research. In addition, different sampling or sketching techniques can be used to store the information for link prediction to balance complexity and performance. Furthermore, link prediction measures can be combined with more baseline algorithms, so to incorporate with other representations for anomaly detection.

Bibliography

[1] Anita S Acharya, Anupam Prakash, Pikee Saxena, and Aruna Nigam. Sampling: Why and how of it. *Indian Journal of Medical Specialties*, 4(2):330–333, 2013. 9

[2] Lada A Adamic, Bernardo A Huberman, AL Barabási, R Albert, H Jeong, and G Bianconi. Power-law distribution of the world wide web. *science*, 287(5461):2115–2115, 2000. 12

[3] Charu C Aggarwal. On biased reservoir sampling in the presence of stream evolution. In *Proceedings of the 32nd international conference on Very large data bases*, pages 607–618. Citeseer, 2006. 9

[4] Charu C Aggarwal. *Data streams: models and algorithms*, volume 31. Springer Science & Business Media, 2007. 9

[5] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017. 2

[6] 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. 1, 5

[7] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688, 2015. 2

[8] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999. 13

[9] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Netsimile: A scalable approach to size-independent network similarity. *arXiv preprint arXiv:1209.2684*, 2012. 2, 7

[10] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *Proceedings of the 22nd international conference on World Wide Web*, pages 119–130, 2013. 2, 6

[11] Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. Midas: Microcluster-based detector of anomalies in edge streams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3242–3249, 2020. 2, 14

[12] Siddharth Bhatia, Rui Liu, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. Real-time streaming anomaly detection in dynamic graphs. *arXiv preprint arXiv:2009.08452*, 2020. 3, 8

[13] Siddharth Bhatia, Mohit Wadhwa, Philip S Yu, and Bryan Hooi. Sketch-based streaming anomaly detection in dynamic graphs. *arXiv preprint arXiv:2106.04486*, 2021. 3, 7

[14] Deepayan Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 112–124. Springer, 2004. 1, 5

[15] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019. 8

[16] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 84–95. Springer, 2000. 1, 6

[17] Colin Cooper and Alan Frieze. A general model of web graphs. *Random Structures & Algorithms*, 22(3):311–335, 2003. 14

[18] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005. 10, 11

[19] Nur Nasuha Daud, Siti Hafizah Ab Hamid, Muntadher Saadoon, Firdaus Sahran, and Nor Badrul Anuar. Applications of link prediction in social networks: A review. *Journal of Network and Computer Applications*, 166:102716, 2020. 1

[20] Dorothy E Denning. An intrusion-detection model. *IEEE Transactions on software engineering*, (2):222–232, 1987. 1, 2

[21] 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. 3, 8

[22] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. Spotlight: Detecting anomalies in streaming graphs. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1378–1386, 2018. 2, 7

[23] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *The Structure and Dynamics of Networks*, pages 195–206. Princeton University Press, 2011. 12

[24] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An empirical comparison of botnet detection methods. *computers & security*, 45:100–123, 2014. 25

[25] Michel L Goldstein, Steven A Morris, and Gary G Yen. Problems with fitting to the power-law distribution. *The European Physical Journal B-Condensed Matter and Complex Systems*, 41(2):255–258, 2004. 9

[26] Zoltan Gyongyi, Hector Garcia-Molina, and Jan Pedersen. Combating web spam with trustrank. In *Proceedings of the 30th international conference on very large data bases (VLDB)*, 2004. 1, 5

[27] Atefeh Heydari, Mohammadali Tavakoli, and Naomie Salim. Detection of fake opinions using time series. *Expert Systems with Applications*, 58:83–92, 2016. 1

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

[29] Xuan Hu, Banghuai Li, Yang Zhang, Changling Zhou, and Hao Ma. Detecting compromised email accounts from the perspective of graph topology. In *Proceedings of the 11th International Conference on Future Internet Technologies*, pages 76–82, 2016. 1

[30] Donghwoon Kwon, Hyunjoo Kim, Jinoh Kim, Sang C Suh, Ikkyun Kim, and Kuinam J Kim. A survey of deep learning-based network anomaly detection. *Cluster Computing*, 22(1):949–961, 2019. 8

[31] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007. 3, 12

[32] Ryan N Lichtenwalter, Jake T Lussier, and Nitesh V Chawla. New perspectives and methods in link prediction. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 243–252, 2010. 3

[33] Richard Lippmann, Robert K Cunningham, David J Fried, Isaac Graf, Kris R Kendall, Seth E Webster, and Marc A Zissman. Results of the darpa 1998 offline intrusion detection evaluation. In *Recent advances in intrusion detection*, volume 99, pages 829–835, 1999. 25

[34] Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z Sheng, Hui Xiong, and Leman Akoglu. A comprehensive survey on graph anomaly detection with deep learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021. 8

[35] Michael Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet mathematics*, 1(2):226–251, 2004. 14

[36] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015. 25

[37] Gopinath Muruti, Fiza Abdul Rahim, and Zul-Azri bin Ibrahim. A survey on anomalies detection techniques and measurement methods. In *2018 IEEE Conference on Application, Information and Network Security (AINS)*, pages 81–86. IEEE, 2018. 1, 5

[38] Ilker Onat and Ali Miri. A real-time node-based traffic anomaly detection algorithm for wireless sensor networks. In *2005 Systems Communications (ICW'05, ICHSN'05, ICMCS'05, SENET'05)*, pages 422–427. IEEE, 2005. 7

[39] Vilfredo Pareto, Ann S Schwier, Alfred N Page, et al. *Manual of political economy*. Macmillan London, 1971. 13

[40] 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, 6

[41] Tahereh Pourhabibi, Kok-Leong Ong, Booi H Kam, and Yee Ling Boo. Fraud detection: A systematic literature review of graph-based anomaly detection approaches. *Decision Support Systems*, 133:113303, 2020. 1, 2

[42] Stephen Ranshous, Steve Harenberg, Kshitij Sharma, and Nagiza F Samatova. A scalable approach for outlier detection in edge streams using sketch-based approximations. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 189–197. SIAM, 2016. 1, 2, 3, 8

[43] Stephen Ranshous, Shitian Shen, Danai Koutra, Steve Harenberg, Christos Faloutsos, and Nagiza F Samatova. Anomaly detection in dynamic networks: a survey. *Wiley Interdisciplinary Reviews: Computational Statistics*, 7(3):223–247, 2015. 2

[44] 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. 2, 6

[45] Srijan Sengupta. Anomaly detection in static networks using egonets. *arXiv preprint arXiv:1807.08925*, 2018. 1, 5

[46] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018. 25

[47] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A Ghorbani. Developing realistic distributed denial of service (ddos) attack dataset and taxonomy. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–8. IEEE, 2019. 25

[48] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. Densealert: Incremental dense-subtensor detection in tensor streams. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1057–1066, 2017. 3, 7

[49] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 374–383, 2006. 2, 6

[50] Jimeng Sun, Yinglian Xie, Hui Zhang, and Christos Faloutsos. Less is more: Compact matrix decomposition for large sparse graphs. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 366–377. SIAM, 2007. 2, 7

[51] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985. 9

[52] Peng Wang, BaoWen Xu, YuRong Wu, and XiaoYu Zhou. Link prediction in social networks: the state-of-the-art. *Science China Information Sciences*, 58(1):1–38, 2015. 3, 11

[53] Teng Wang, Chunsheng Fang, Derek Lin, and S Felix Wu. Localizing temporal anomalies in large evolving graphs. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 927–935. SIAM, 2015. 2, 7

[54] Jarrod West and Maumita Bhattacharya. Intelligent financial fraud detection: a comprehensive review. *Computers & security*, 57:47–66, 2016. 2

[55] Weiren Yu, Charu C Aggarwal, Shuai Ma, and Haixun Wang. On anomalous hotspot discovery in graph streams. In *2013 IEEE 13th International Conference on Data Mining*, pages 1271–1276. IEEE, 2013. 3, 7

[56] Peixiang Zhao, Charu Aggarwal, and Gewen He. Link prediction in graph streams. In *2016 IEEE 32nd international conference on data engineering (ICDE)*, pages 553–564. IEEE, 2016. 3, 12

[57] George Kingsley Zipf. *Human behavior and the principle of least effort: An introduction to human ecology.* Ravenio Books, 2016. 13