INFORMATION THEORETIC, PROBABILISTIC AND MAXIMUM PARTIAL SUBSTRUCTURE ALGORITHMS FOR DISCOVERING GRAPH-BASED ANOMALIES

by

WILLIAM FRED EBERLE

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2007

ACKNOWLEDGEMENTS

I would like to thank my supervising professor, Dr. Lawrence Holder, whose continued support and encouragement made this possible. Throughout every phase of this research, he challenged me to the best of my ability. His guidance and mentoring during my entire graduate studies have been an inspiration to me, for which I hope some day to be able to instill in other students.

I would like to express my gratitude to Dr. Lynn Peterson, who supervised my Master’s work and not only encouraged me to pursue my PhD, but was always a source of encouragement throughout the graduate school process. I would also like to thank the rest of the committee, Dr. Diane Cook, Dr. Sharma Chakravarthy and Dr. Gautam Das, whose input and encouragement were instrumental in the completion of this work.

I would like to thank my parents who have always encouraged me in my academic endeavors. Their examples of scholarly pursuits have always been an inspiration to me. I would also like to thank the rest of my family and friends for their encouragement during this process.

And last, but not least, I would like to thank my beautiful wife Robin. Without her love and encouragement through what has been a tough three years, this work and all of my graduate studies would not have been possible.

April 23, 2007

ABSTRACT

INFORMATION THEORETIC, PROBABILISTIC AND MAXIMUM PARTIAL SUBSTRUCTURE ALGORITHMS FOR DISCOVERING GRAPH-BASED ANOMALIES

Publication No. ______

William Fred Eberle, PhD.

The University of Texas at Arlington, 2007

Supervising Professor: Lawrence B. Holder

The ability to mine data represented as a graph has become important in several domains for detecting various structural patterns. One important area of data mining is anomaly detection, particularly for fraud. However, less work has been done in terms of detecting anomalies in graph-based data. While there has been some previous work that has used statistical metrics and conditional entropy measurements, the results have been limited to certain types of anomalies and specific domains.

In this work we present graph-based approaches to uncovering anomalies in domains where the anomalies consist of unexpected entity/relationship alterations that closely resemble non-anomalous behavior. We have developed three algorithms for the purpose of detecting anomalies in all three types of possible graph changes: label modifications, vertex/edge insertions and vertex/edge deletions. Each of our algorithms focuses on one of these anomalous types, using the minimum description length principle to first discover the normative substructure. Once the common pattern is known, each algorithm then uses a different approach to discover particular anomalous types. The first algorithm uses the minimum description length to find substructures that closely compress the graph. The second algorithm uses a probabilistic approach to examine substructure extensions and their likelihood of existence. The third algorithm analyzes substructures that come close to matching the normative pattern, but are unable to make some of the final extensions.

Using synthetic and real-world data, we evaluate the effectiveness of each of these algorithms in terms of each of the types of anomalies. Each of these algorithms demonstrates the usefulness of examining a graph-based representation of data for the purposes of detecting fraud, where some individual or entity is cloaking their illegal activities through an attempt at closely resembling legitimate transactions.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Detecting anomalies in various data sets is an important endeavor in data mining. Using statistical approaches has led to various successes in environments such as intrusion detection. Recent research in graph-based anomaly detection has paved the way for new approaches that not only compliment the non-graph-based methods, but also provide mechanisms for handling data that cannot be easily analyzed with traditional data mining approaches. Using information theoretic, probabilistic and maximum partial substructure approaches, we have developed three novel algorithms for analyzing graph substructures for the purpose of uncovering all three types of graph-based anomalies: modifications, insertions and deletions.

The key to the algorithms presented in this work lies in our definition of an anomaly. Basing our definition on the assumption that an anomaly is not random, for instance in the case of committing fraud, we believe that this type of anomaly should only be a minor deviation from the normal pattern. Because anyone who is attempting to commit fraud or hide devious activities would not want to be caught, it only makes sense that they would want their activities to look as real as possible. For example, the United Nations Office on Drugs and Crime states the first fundamental law of money laundering as "The more successful money-laundering apparatus is in imitating the patterns and behavior of legitimate transactions, the less the likelihood of it being exposed." [Hampton and Levi 1999]. Thus, if some set of data is represented as a graph, any nefarious activities should be identifiable by small modifications, insertions or deletions to the normative patterns within that graph.

Our first algorithm uses the *minimum description length* principle [Rissanen 1989] to determine the normative pattern, and from that pattern, find patterns that while structurally similar, have some relational deviation that is within an acceptable level of change. By determining what substructure minimizes the description length of the graph after compressing the instances of the normative pattern, we are able to calculate the cost of transformation for instances within the graph that do not exactly match the discovered normative pattern, and as such, are indicative of an unexpected change.

Our second algorithm again determines the normative pattern as the one that minimizes the description length of a graph, but instead of looking at changes to this pattern we examine the *probability* of extensions to the pattern. If the normative pattern does not completely compress the graph, meaning there are other vertices and edges connected to the normative pattern, we examine each of these extensions in terms of the probability of their existence. If the probability of existence is low enough, we mark the instance as anomalous. We can then compress the graph by this anomalous instance, and repeat the process until there are no more extensions to the anomalous substructure.

Our third algorithm uses a trail of pattern expansion to discover the instances that are structurally deficient from the normative pattern. When we attempt to discover the pattern that minimizes the description length of the graph, we maintain a parental relationship between the structures. Once we have discovered the normative pattern, we traverse these relationships to find the instance that is the *maximum partial substructure*. In this case, we are looking for patterns that are unable to extend to the normative pattern, and are a maximal representation of that normative pattern. In other words, the maximum partial substructure is found in the instance that requires the fewest additions (IF they would have existed) for transforming the instance into an instance consisting of the normative structure.

Up until now, graph-based approaches to fraud detection have been limited to specific anomaly types and certain domains. Taking into account the “mind of the fraudster”, we have developed algorithms that can discover any of three types of anomalous changes where the illegitimate actions consist of minor changes to a normal set of activities.

The following work represents the development of graph-based anomaly detection algorithms for the discovery of anomalies in data represented as a graph. In our work, we assume that an anomaly is a minor deviation from a normative pattern, where the instances being considered are connected substructures. Using the minimum description length principle, we have developed algorithms to examine substructures that have unexpected modifications, insertions or deletions. In Chapter 2 we present some related work, particularly the more recent research in graph-based anomaly detection. In Chapter 3 we present our definition of a graph-based anomaly, including the assumptions we are making regarding the data. In Chapter 4 we present each of the three algorithms, as well as some small examples to explain their approaches. In Chapter 5 we present our empirical evaluation of the algorithms using synthetic data sets. In Chapter 6 we present results using real-world data, including cargo shipments and network intrusions. Finally, in Chapter 7, we present our conclusions and future work.

CHAPTER 2

RELATED WORK

2.1 Anomaly Detection Methods

In 1987, Dorothy Denning wrote one of the first papers on what was called an "intrusion-detection model" [Denning 1987]. It became perhaps one of the most cited papers in the area of intrusion detection, where the goal was to define a model by which attacks on a network could be identified. The idea was that through a series of rule-based pattern matching, one could create a computer program that would detect in near real-time whether or not a particular event, or events, broke a specified threshold, or what was called a "confidence interval". With the evolution of the internet in the 1990's, this became an even bigger issue, as attacks on networks were now global and not just restricted to a single company's internal network or machines [Lee and Stolfo 1998].

In the mid 1990's, the ideas of intrusion detection led to a broader definition: *anomaly detection*. As Kumar stated in his paper on the subject of intrusion, "Anomaly detection attempts to quantify the usual or acceptable behavior and flags other irregular behavior as potentially intrusive." [Kumar 1995]. In 1998, Lane expanded on the previous research by also adding in the concept of time and classification layers [Lane 1998]. Using rule-based detectors, a form of profiling was created, whereby data could be classified into different layers. There became two approaches to looking at what was now being viewed in generic terms as anomalous data. One was the searching for different patterns in a data set, and another was the comparing of two data sets and classifying how anomalous they were to each other.

The next generation of anomaly detection systems (ADSes) followed many different directions. [Maxion and Tan 2000] hypothesized that the difference in data regularities affected detector performance and was making it difficult to apply one approach to all domains. They also described the regularity of the data as its intrinsic structure. The notion that entropy, or the measure of uncertainty of a collection of data, was starting to be applied to this field. This also led to considering conditional entropy, or the measure of regularity of sequential dependencies, and relative entropy, which measured the distance of the regularities between two data sets. From this concept, [Lee and Xiang 2000] discussed the idea of information gain, whereby it was observed that the larger the data set (i.e., more information), the more *regular* it becomes, decreasing the conditional entropy.

2.2 Graph-based Anomaly Detection Methods

Recently there has been an impetus towards analyzing multi-relational data using graph-theoretic methods. Not to be confused with the mechanisms for analyzing "spatial" data, graph-based data mining approaches are an attempt at analyzing data that can be represented as a graph (i.e., vertices and edges). Yet, while there has been much written as it pertains to graph-based intrusion detection [Staniford-Chen et al. 1996], very little research has been accomplished in the area of graph-based anomaly detection.

In 2003, Noble and Cook used the SUBDUE application to look at the problem of anomaly detection from both the anomalous substructure and anomalous subgraph perspective [Noble and Cook 2003]. They were able to provide measurements of anomalous behavior as it applied to graphs from two different perspectives. *Anomalous substructure* detection dealt with the unusual substructures that were found in an entire graph. In order to distinguish an anomalous substructure from the other substructures, they created a simple measurement whereby the value associated with a substructure indicated a degree of anomaly. They also presented the idea of *anomalous subgraph* detection which dealt with how anomalous a subgraph (i.e., a substructure that is part of a larger graph) was to other subgraphs. The idea was that subgraphs that contained many common substructures were generally less anomalous than subgraphs that contained few common substructures. In addition, they also explored the idea of conditional entropy and data regularity using network intrusion data as well as some artificially created data.

[Lin and Chalupsky 2003] took a different approach and applied what they called rarity measurements to the discovery of unusual *links* within a graph. Using various metrics to define the commonality of paths between nodes, the user was able to determine whether a path between two nodes was interesting or not, without having any preconceived notions of meaningful patterns. One of the disadvantages of this approach was that while it was domain independent, it assumed that the user was querying the system to find interesting relationships regarding certain nodes. In other words, the unusual patterns had to originate or terminate from a user-specified node.

The AutoPart system presented a non-parametric approach to finding outliers in graph-based data [Chakrabarti 2004]. Part of Chakrabarti's approach was to look for outliers by analyzing how *edges* that were removed from the overall structure affected the minimum descriptive length (MDL) of the graph. Representing the graph as an adjacency matrix, and using a compression technique to encode node groupings of the graph, he looked for the groups that reduced the compression cost as much as possible. Nodes were put into groups based upon their entropy.

In 2005, the idea of entropy was also used by [Shetty and Adibi 2005] in their analysis of a real-world data set: the famous Enron scandal. They used what they called "event based graph entropy" to find the most interesting people in an Enron e-mail data set. Using a measure similar to what [Noble and Cook 2003] had proposed, they hypothesized that the important nodes (or people) were the ones who had the greatest effect on the entropy of the graph when they were removed. Thus, the most interesting node was the one that brought about the maximum change to the graph's entropy. However, in this approach, the idea of important nodes did not necessarily mean that they were anomalous.

In the December 2005 issue of SIGKDD Explorations, a couple of different approaches to graph-based anomaly detection were presented. Using just *bipartite* graphs, [Sun et al. 2005] presented a model for scoring the normality of nodes as they relate to the other nodes. Again, using an adjacency matrix, they assigned what they called a "relevance score" such that every node *x* had a relevance score to every node *y*, whereby the higher the score the more related the two nodes. The idea was that the nodes with the lower normality score to *x* were the more anomalous ones to that node. The two drawbacks with this approach were that it only dealt with bipartite graphs and it only found anomalous nodes, rather than what could be anomalous substructures. In [Rattigan and Jensen 2005], they also went after anomalous links, this time via a *statistical* approach. Using a Katz measurement, they used the link structure to statistically predict the likelihood of a link. While it worked on a small dataset of author-paper pairs, their single measurement just analyzed the links in a graph.

2.3 Using Graph Properties to Discover Anomalous Graphs

The ability to mine relational data has become important in several domains (e.g., counter-terrorism), and a graph-based representation of this data has proven useful in detecting various relational, structural patterns [Mukherjee and Holder 2004]. In [Eberle and Holder 2006], we analyzed the use of *graph properties* as a method for uncovering anomalies in data represented as a graph.

While our initial research examined many of the basic graph properties, only a few of them proved to be insightful as to the structure of a graph for anomaly detection purposes: average shortest path length, density and connectedness. For a measurement of *density*, we chose to use a definition that is commonly used when defining social networks [Scott 2000]. For *connectedness*, we used a definition from [Broder et al. 2000]. Then, for some of the more complex graph properties, we investigated two measurements. First, there is the maximum *eigenvalue* of a graph [Chung et al. 2003]. Another, which was used in identifying e-mail “spammers”, is the *graph clustering coefficient* [Boykin and Roychowdhury 2005].

In order to test these graph properties on synthetic data, we created 6 different graph size types consisting of approximately 35, 100, 400, 1000, and 2000 vertices, and another being a dense graph of 100 vertices and 1000 edges. For each of these increment sizes, we created 30 non-anomalous graphs. We then generated 30 anomalous graphs for each of the graph types and for each of the following structural anomalies: add substructure, remove substructure, move edge, and add isolated substructure.

The *density* (Figure 2.1) of small graphs lessens when an anomalous substructure is connected to existing vertices in the graph. This makes sense, as the ratio of actual vertices and edges to the number of *possible* pairs would increase, resulting in a lower density. This also explains why the density of graphs that contain *isolated* substructures is less, due to containing unconnected vertices. Also, the removal of a substructure results in a wide deviation in the density measurement. The *connectedness* of the smaller graphs varies for each of the different types of anomalies. The insertion and isolation anomalies result in lower values, and insertion of an isolated substructure has an even greater variation on the measurement. The same behavior is also found in dense graphs. Changes in the *clustering coefficient* (Figure 2.2) on smaller graphs are only evident for inserted isolated anomalous substructures and the anomaly of moved edges. This variance, because of the moved edges, is significant due to the way the deviation changes. As the graphs get larger, the distribution still holds, but the coefficient of the graphs with moved edges increases significantly. The *average shortest path length* (Figure 2.3) and *eigenvalue* (Figure 2.4) metrics behave similarly to the above metrics, except that they are better indicators of inserted substructures and moved edges.

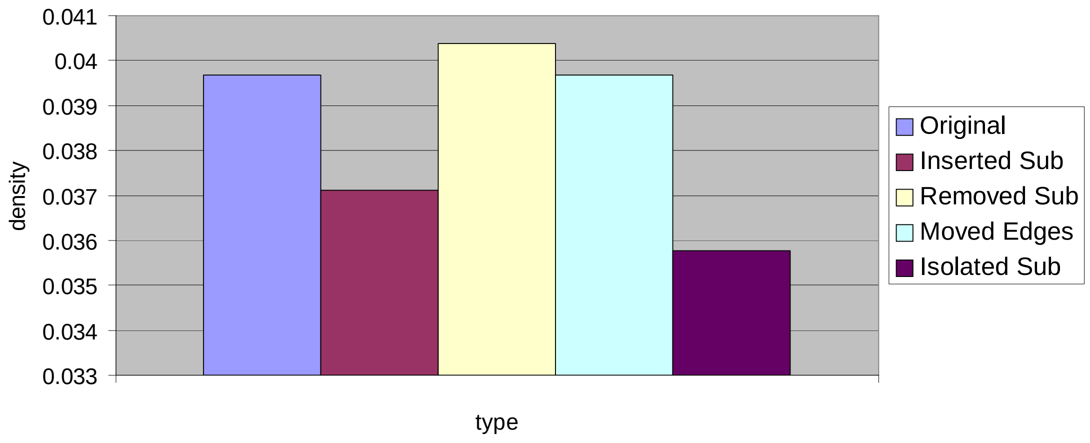

Figure 2.1 Density (mean) of different anomalies on small graphs.

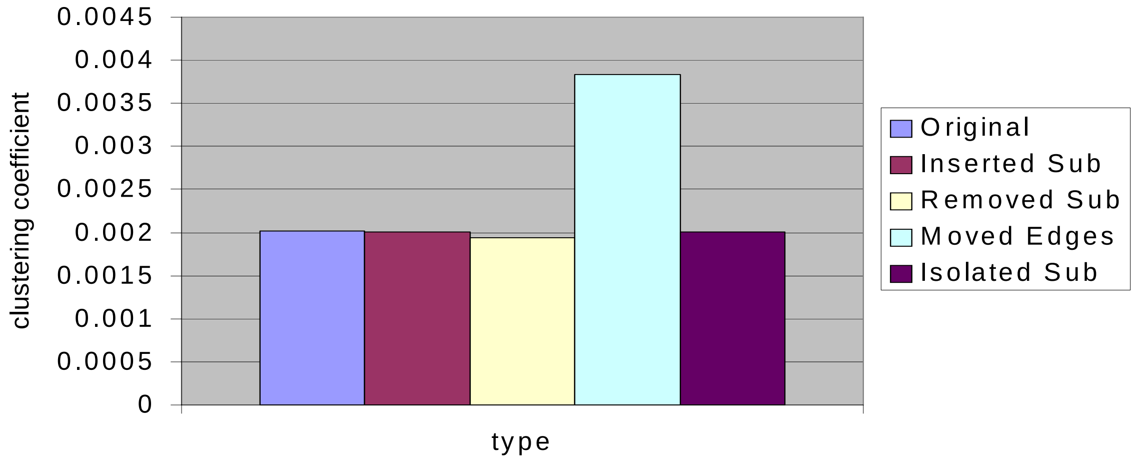

Figure 2.2 Clustering coefficient (mean) on larger graphs.

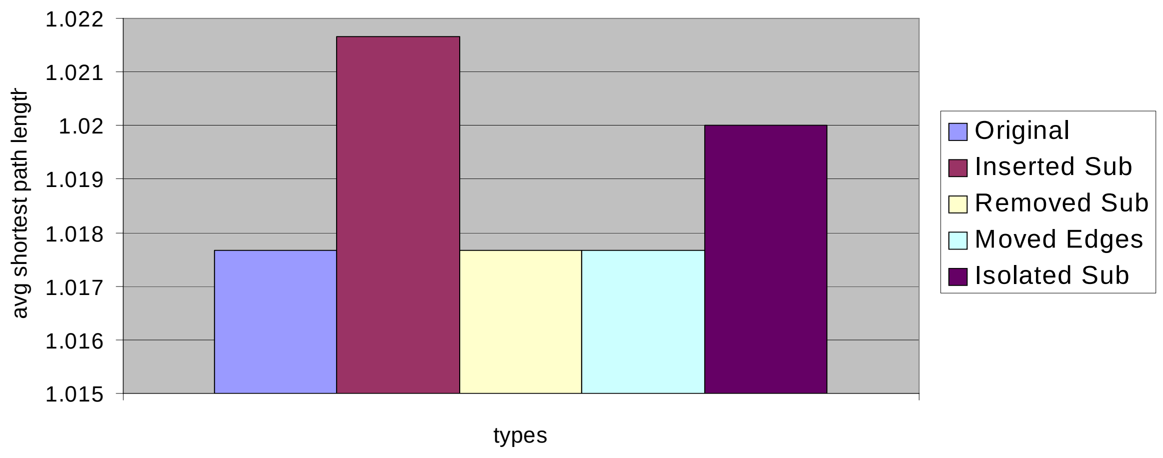

Figure 2.3 Average shortest path length (mean) on small graphs.

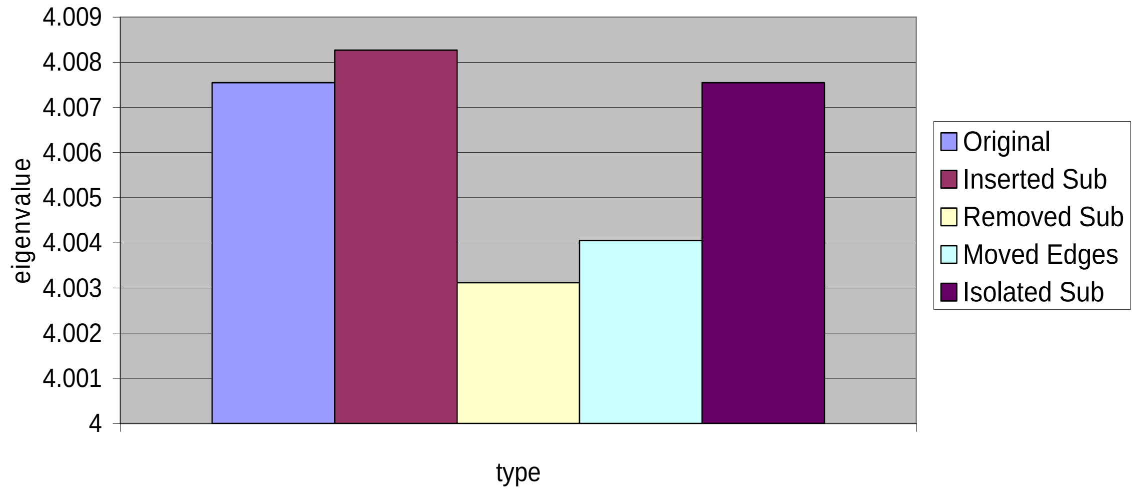

Figure 2.4 Highest eigenvalue (mean) on larger graphs.

We also tested these approaches on data from cargo shipments represented as graphs. When injecting real-world anomalies into the data, no significant deviations are displayed using the average shortest path or eigenvalue metrics. However, there are visible differences for the density, connectedness and clustering coefficient measurements. Figures 2.5, 2.6 and 2.7 represent the values for each of the corresponding metrics in each test example. The bottom plot lines in each chart show the values on the original cargo data. As shown, for every test where the anomalous examples were randomly interjected into the cargo data, a noticeable deviation occurs in the corresponding graph property.

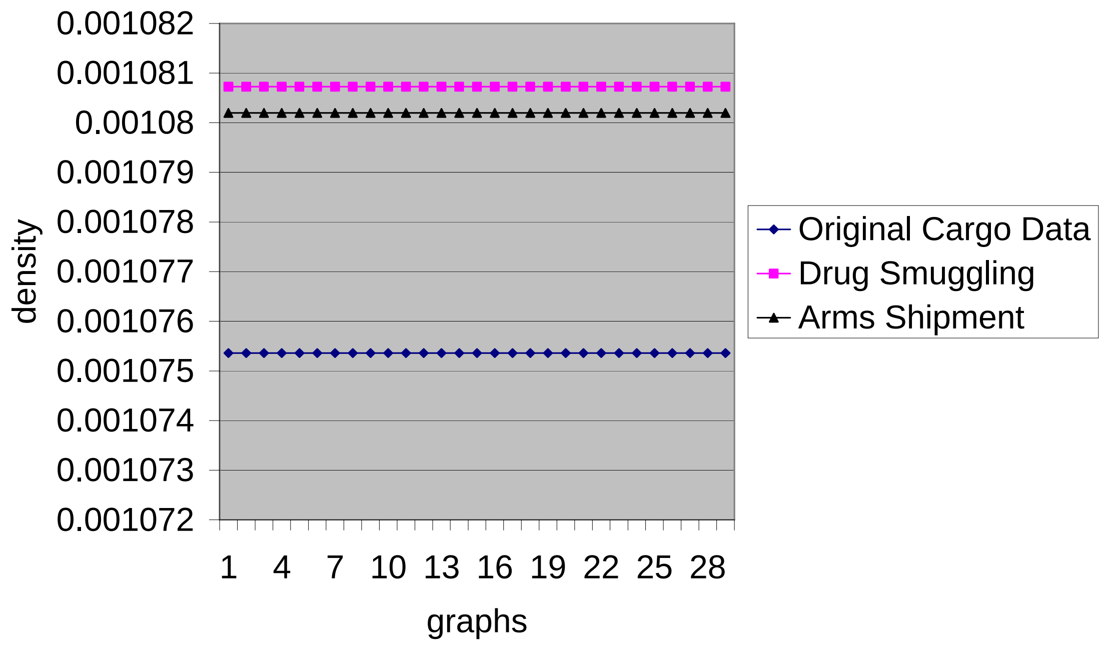

Figure 2.5 Density changes on cargo data.

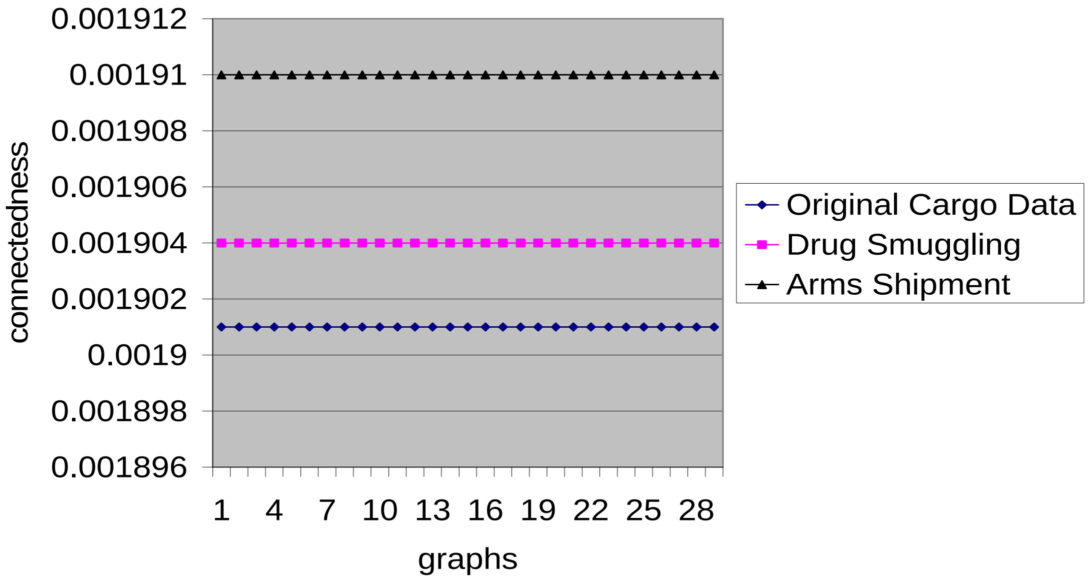

Figure 2.6 Connectedness changes on cargo data.

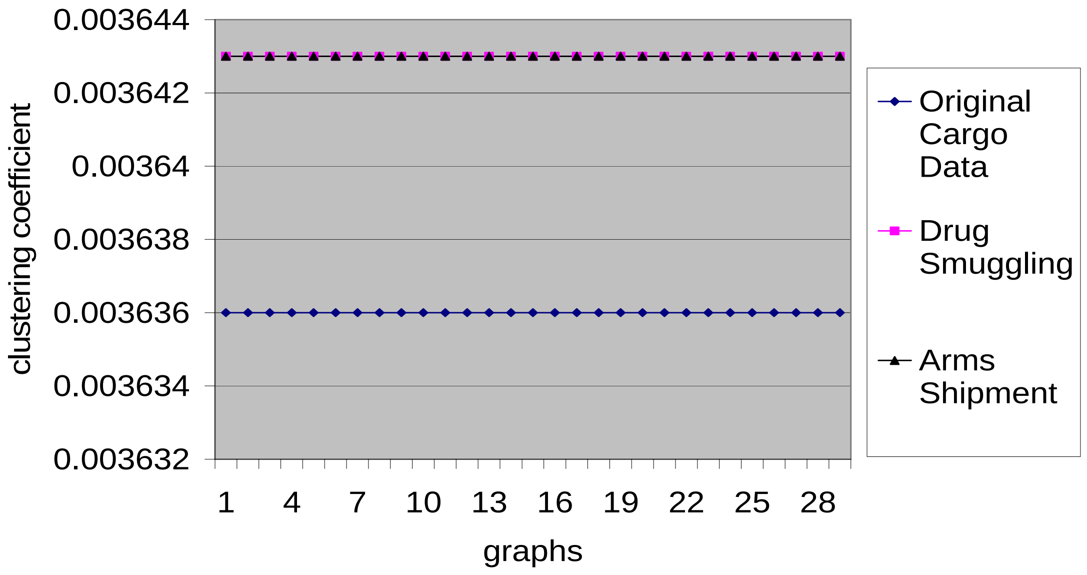

Figure 2.7 Clustering coefficient changes on cargo data.

Another metric that can be used is the *combination* of individual measurements to provide a clearer view. For instance, when combining the density, connectedness and clustering coefficient measurements, we get values that clearly indicate anomalies. Similar results are evident when applying different combinations on the synthetic data sets.

The issue with this approach is that while graphs are indicated as anomalous, this does not provide the specific anomaly within what could be a very large graph of data. However, the algorithms presented in this work will rectify that problem by not only indicating a graph contains an anomaly, but more importantly, they will discover the specific anomaly and its pertinent structure within the graph.

2.4 Graph-Based Pattern Discovery

While not specific to anomaly detection, there are several approaches to handling just the discovery of the normative pattern in data that is represented as a graph. One approach called gSpan returns all *frequent* substructures in a database that is represented as a graph [Yan and Han 2002]. Using a depth-first search on the input graphs, the algorithm constructs a hierarchical search tree based upon the DFS code assigned to each graph. Then, from its canonical tree structure, the algorithm performs a pre-order traversal of the tree in order to discover the frequent subgraphs.

Another approach is found in FSG, which is similar to gSpan in that it returns all of the frequent subgraphs (substructures) in a database of transactions that have been represented as a graph [Kuramochi and Karypis 2004]. However, unlike gSpan, FSG uses an Apriori-style breadth-first search. The algorithm takes the input graphs and performs a level-by-level search, growing patterns one edge at a time. The core of the FSG algorithm lies in its candidate generation and counting that are used to determine the frequent subgraphs.

In order to mine large graphs for frequent subgraphs, Huan et al. proposed a maximal frequent subgraphs approach called SPIN as an improvement to gSpan [Huan et al. 2004]. By mining only subgraphs that are not part of any other frequent subgraphs, they are able to reduce the number of mined patterns by orders of magnitude. This is accomplished by first mining all frequent trees from a graph, and then reconstructing all maximal subgraphs from the mined trees. Zeng et al. looked at the problem of dense graphs by mining the properties of quasi-cliques [Zeng et al. 2006]. Using a system called Cocain, they propose several optimization techniques for pruning the unpromising and redundant search spaces. To help combat the subgraph isomorphism issue, Gudes et al. proposed a new Apriori-based algorithm using disjoint paths [Gudes et al. 2006]. Following a breadth-first enumeration and what they called an “admissible support measure”, they are able to prune candidate patterns without checking their support, significantly reducing the search space. MARGIN is another maximal subgraph mining algorithm that focuses on the more promising nodes in a graph [Thomas et al. 2006]. This is accomplished by searching for promising nodes in the search space along the “border” of frequent and infrequent subgraphs, thus reducing the number of candidate patterns.

The goal of SUBDUE is to return the substructures that compress the graph the best [Holder et al. 1994]. Using a beam search (a limited length queue of the best few patterns that have been found so far), the algorithm grows patterns one edge at a time, continually discovering what substructures best compress the description length of the input graph. The core of the SUBDUE algorithm is in its compression strategy. After extending each substructure by one edge, it evaluates each extended substructure based upon its compression value (the higher the better). A list is maintained of the best substructures, and this process is continually repeated until either there are no more substructures to compress or a user-specified limit is reached.

While each of these approaches is successful at pattern discovery, we will use the SUBDUE compression evaluation technique as the basis for our underlying discovery of the normative patterns. While the gSpan application is not publicly available, there are a few reasons why we found FSG to not be an ideal candidate for our implementation. One reason for our choice of pattern learner lies with the format expected by the FSG application. SUBDUE can effectively discover normative patterns whether it is given all transactions or data as one entire graph, or if each transaction is defined as individual subgraphs. As a graph data miner, FSG shows the frequency of a pattern based upon the number of transactions defined in the graph input file. So, if a graph is not delineated by individual transactions, the frequency of every pattern is 1, and thus very difficult to determine which pattern is the most frequent. However, in some later work by Kuromachi and Karypis, they improve upon this with an approach called Grew that is able to better handle large graphs that consist of connected subgraphs [Kuromachi and Karypis 2004]. Another reason lies in the FSG approach to determining the normative pattern based upon frequency. While tests on various graphs showed SUBDUE and FSG returned the same normative pattern, when the tests involved a graph where the normative pattern is not found across all transactions (e.g., noise), the frequent pattern is not found unless the FSG support percentile is reduced. The issue then is knowing what support percentile should be used for a specific run. Specifying 100% support will result in the normative pattern being lost if the pattern is not found in at least one transaction, while using a lower percentile may result in other (smaller) normative patterns being found. In short, SUBDUE allows us to find the normative pattern in data that may be less regular or contain some noise. As will be shown in the following section, this is critical to the success of discovering anomalies.

CHAPTER 3

GRAPH-BASED ANOMALIES

Webster defines an anomaly as "a deviation from the common rule, type, or form." [Webster 1989]. Webster's Thesaurus states that the word "anomaly" can also be termed "peculiarity, unconformity, exception,…, irregularity" [Webster 1971].

3.1 Definition of Graph-Based Anomaly

Setting up fraudulent web-sites, "phishing" for credit cards, stealing calling cards, and creating bogus bank accounts are just some of the countless examples of scams that have succumb everyone from the individual investor to large corporations. In every case, the fraudster has attempted to swindle their victim and hide their dealings within a morass of data that has become proverbially known as the "needle in the haystack". Yet, even when the data is not relatively large in size, the ability to discover the nefarious actions is still ultimately difficult due to the *mimicry* of the perpetrator.

Before we lay the groundwork for our definition of a graph-based anomaly, we need to put forth a framework for the definition of a graph. In general, a graph is a set of nodes and a set of links, where each link connects either two nodes or a node to itself. More formally, we use the following definitions:

Definition: A graph *G* = (*V*,*E*,*L*) is a mathematical structure consisting of three sets *V*, *E* and *L*. The elements of *V* are called *vertices* (or nodes), the elements of *E* are the *edges* (or links) between the vertices, and the elements of *L* are the string *labels* assigned to each of the elements of *V* and *E*.

Definition: A vertex (or node) is an entity (or item) in a graph. For each vertex there is a labeled vertex pair (*v*,*l*) where *v* is a vertex in the set *V* of vertices and *l* is a string label in the set *L* of labels.

Definition: An edge (or link) is a labeled relation between two vertices called its endpoints. For each edge there is a labeled edge pair (*e*, *l*) where *e* is an edge in the set *E* of edges and *l* is a string label in the set *L* of labels.

Definition: An edge can be directed or undirected. A *directed* edge is an edge, one of whose endpoints is designated as the *tail*, and whose other endpoint is designated as the *head*. An *undirected* edge is an edge with two unordered endpoints. A multi-edge is a collection of two or more edges having identical endpoints.

[Gross and Yellen 1999] [West 2001]

Much research has been done recently using *graph*-based representations of data. Using *vertices* to represent entities such as people, places and things, and *edges* to represent the relationships between the entities, such as friend, lives and owns, allows for a much richer expression of data than is present in the standard textual or tabular representation of information. Representing various data sets like telecommunications call records, financial information and social networks in a graph form allow us to discover *structural* properties in data that are not evident using traditional data mining methods.

The idea behind the approach presented in this work is to find anomalies in graph-based data where the anomalous substructure (at least one edge or vertex) in a graph is part of (or attached to or missing from) a non-anomalous substructure, or the *normative pattern*. This definition of an anomaly is unique in the arena of graph-based anomaly detection, as well as non-graph-based anomaly detection. The concept of finding a pattern that is "similar" to frequent, or good, patterns, is different from most approaches that are looking for unusual or “bad” patterns. While other non-graph-based data mining approaches may aide in this respect, there does not appear to be any existing approaches that directly deal with this scenario.

Definition: Given a graph *G* with a normative substructure *S*, a substructure *S’* and a *d* that is the difference between *S* and *S’*, let *C*(*d*) be the cost of the difference and *P*(*d*) be the probability of the difference. Then the graph G is considered anomalous if 0 < *A(S’)* <= *X*, where *X* is a user-defined threshold and *A(S’)* = C(d) * P(d) is the anomaly score.

The importance of this definition lies in its relationship to fraud detection (i.e., any sort of deceptive practices that are intended to illegally obtain or hide information). If a person or entity is attempting to commit fraud, they will do all they can to hide their illicit behavior. To that end, their approach would be to convey their actions as close to legitimate actions as possible. That makes this definition of an anomaly extremely relevant.

3.2 Anomaly Types

For a graph-based anomaly, there are several situations that might occur:

1. *The label on a vertex is different than was expected.*
2. *The label on an edge is different than was expected.*
3. *A vertex exists that is unexpected.*
4. *An edge exists that is unexpected.*
5. *An expected vertex is absent.*
6. *An expected edge between two vertices (or a self-edge to a vertex) is absent.*

It is also evident that these same situations can be applied to a substructure (i.e., multiple vertices and edges), and will be addressed as such. In essence, there are three general *categories of anomalies*: modification, insertions and deletions. Modifications would constitute the first two situations; insertions would consist of the third and fourth situation; and deletions would categorize the last two situations.

In this work, we will not be addressing the possible scenario of a change in the directedness of an edge. While directed edges are allowed, we will not introduce any anomalies that consist of change in an edge's direction. There is also the situation where a graph changes over time. [Coble et al. 2005] addressed this dimension in their paper that proposed an iterative approach to substructure discovery, as well as how it related to anomaly detection. [Cortes et al. 2003] examined the temporal evolution of large dynamic graphs at it related to telecommunication's fraud detection. Also, [Ide and Kashima 2005] addressed the problem of anomaly detection from a time sequence of graphs using the principle eigenvector of the eigenclusters of the graph. The dimension of analyzing graphs over time is beyond the scope of this work, but the approaches presented here can be applied to graphs changing over time as well.

3.3 Assumptions

Many of the graph-based anomaly detection (or intrusion detection) approaches up to now have assumed that the data exhibits a power-law distribution. For example, much of the data that has been used in previous analysis has used items like the world-wide web, social networks, or other sources that convey a power-law behavior [Faloutsos et al. 1999]. The advantage of the approaches presented in this work is that it does not assume the data consists of a power-law behavior. In fact, no standard distribution model is assumed to exist. All that is required is that the data is *regular,* which in general means that the data is “predictable”. While there are many data sets that are not regular in nature, many of the real-world data sets that are examined for fraudulent activity, such as telecommunications call traffic, financial transactions and shipping manifests, consist of user transactions that exhibit regular patterns of behavior.

In order to address our definition of an anomaly, we make the following assumptions about the data:

Assumption 1: *The majority of a graph consists of a normative pattern.*

In general, the more regular the data (or graph), the more predictable it is to discover anomalies. If a graph were irregular, the ability to distinguish between anomalies and noise would be prohibitive.

Assumption 2: *No more than X% of the normative pattern is altered in the case of an anomaly.*

Since our definition implies that an anomaly constitutes a minor change to the prevalent substructure, we can choose a small percentage (e.g., 10%) to represent the most a substructure would be changed in a fraudulent action.

Assumption 3: *Anomalies consist of one or more modifications, insertions or deletions.*

As was described in Section 3.2, there are only three types of changes that can be made to a graph. Therefore, anomalies that consist of structural changes to a graph must consist of one of these types.

Assumption 4: *The normative pattern is connected.*

In a real-world scenario, we would apply this approach to data such as cargo shipments, telecommunications traffic, financial transactions or terrorist networks. In all cases, the data consists of a series of nodes and links that share common nodes and links. Certainly, graphs could contain potential anomalies across disconnected substructures, but at this point, we are constraining our research to only connected anomalies.

CHAPTER 4

GRAPH-BASED ANOMALY DETECTION ALGORITHMS

Most anomaly detection methods use a supervised approach, which requires some sort of baseline of information from which comparisons or training can be performed. In general, if we have an idea what is normal behavior, deviations from that behavior could constitute an anomaly. However, the issues with these approaches are that one has to have the data in advance in order to train the system, and the data has to already be labeled (i.e., fraudulent versus legitimate).

Our work has resulted in the development of three algorithms, which we have implemented using a tool called GBAD (Graph-Based Anomaly Detection). GBAD is an *unsupervised* approach, based upon the SUBDUE graph-based knowledge discovery system [Cook and Holder 1998]. Using a breadth-first search and Minimum Description Length (MDL) heuristic, each of the three anomaly detection algorithms uses GBAD to provide the normative pattern in an input graph. In our implementation, the MDL approach is used to determine the best substructure(s) as the one that minimizes the following:

$$M(S,G) = DL(G \mid S) + DL(S)$$

where *G* is the entire graph and *S* is the substructure. *M(S,G)* is the message length of the graph *G* with respect to the substructure *S* where *DL(G|S)* is the description length of *G* after compressing it using *S*, and *DL(S)* is the description length of the substructure.

Using GBAD as the tool for our implementation, we have developed three separate algorithms: GBAD-MDL, GBAD-P and GBAD-MPS. Each of these approaches is intended to discover all of the possible graph-based anomaly types as set forth earlier.

4.1 Information Theoretic Algorithm (GBAD-MDL)

The GBAD-MDL algorithm uses a Minimum Description Length (MDL) heuristic to discover the best substructure in a graph (i.e., the substructure that compresses the graph the most), and then subsequently examines all of the instances of that substructure that "look similar" to that pattern.

The high-level approach for the GBAD-MDL algorithm is, for a graph *G*:

- Find the best substructure *S* that minimizes the description length of *G*.
- Find all instances I_k, whose cost of transformation is less than a specified threshold, where the threshold is a user-defined parameter.
- Output all I_k whose (cost * frequency) is minimum.

"Cost of transformation" (or *matchcost*) is the cost of transforming subgraph A into an isomorphism of subgraph B. We calculate this by adding 1.0 for each vertex, edge and label that would need to be changed in order to make A isomorphic to B.

4.1.1 Algorithm

In Algorithm 1, the threshold *T* is the user-defined level of inexact matching that can occur. That is to say, it is the amount of change that one is willing to accept between an instance (or subgraph of the graph *G*) and the normative substructure. The detailed GBAD-MDL algorithm is as follows:

Algorithm 1: proc **GBAD-MDL** (graph *G*, threshold *T*)

```
1. Find normative substructure $S$ minimizing $DL(S)+DL(G|S)$
2. Identify instances $I_k$ of $S$ in $G$ having matchcost($I_k$,$S$) < ($T$ * size(|$S$|))
3. For each instance $I_k$ such that matchcost($I_k$,$S$) > 0
    a. freq($I_k$) = number of instances of $S$ that exactly match $I_k$
    b. anomalyScore($I_k$) = freq($I_k$) * matchcost($I_k$,$S$)
4. Return all instances $I_k$ having the minimal anomalyScore
```

With the inexact matching that occurs in the GBAD-MDL algorithm, the result will be those instances that are the "closest" (without matching exactly) in structure to the best structure (i.e., compresses the graph the most), where there is a tradeoff in the cost of transforming the instance to match the structure, as well as the frequency with which the instance occurs. Since cost of transformation and frequency are independent variables, multiplying their values together results in a combinatory value: the lower the value, the more anomalous the structure. It is these inexact matching instances that will be analyzed for anomalousness.

Note that we are only interested in the top substructure (i.e., the one that minimizes the description length of the graph), so *k* will always be 1. However, for extensibility, we can work from the top k substructures if it is felt that anomalous behavior is not found in the top normative pattern.

4.1.2 Implementation

The goal of SUBDUE is to discover the best pattern in a graph using a Minimum Description Length (MDL) evaluation of how much a substructure compresses the graph. This is fundamentally different from GBAD-MDL where the approach is to use the same MDL principle but instead look for those substructures that do *not* compress the graph the best, but are structurally closest to the best substructure.

In order to implement the GBAD-MDL algorithm, we first used SUBDUE to discover the best substructure. In addition to providing the normative pattern using an MDL evaluation, SUBDUE also provides two other features: the ability to specify inexact matching as a percentage of the normative substructure, and a list of all instances that match the best substructure. SUBDUE terminates processing when there are no more extensions to candidate substructures, whereas the GBAD-MDL algorithm continues processing the best substructure by analyzing its instances for the one that is closest in transformation cost to the normative pattern.

First, the algorithm modifies the best substructure list by determining which substructure is actually the true normative pattern. Since an inexact matching was used, it is possible that the top substructure specified in SUBDUE (i.e., the best substructure), may not be the true normative pattern. So, a search is performed on the list of instances, finding the pattern that is the most frequent, and replacing the previously specified best substructure with its structure.

Second, the new list of instances is compared to the new best substructure, and each instance is given an anomalous score equal to its cost of transformation (for transforming the instance into the best substructure). Then, for each instance in the list that matches this instance (i.e., isomorphic), the anomalous score is increased by the value of the cost of transformation – in essence, creating an anomalous score that is equal to the cost of transformation times frequency.

For the last step, our GBAD-MDL implementation finds the anomalous instance (or instances, if their anomalous scores are equal), and flags the individual vertices and edges that are anomalous. This is accomplished by comparing the structure of the anomalous instance with the normative substructure, and for each vertex and edge in the anomalous instance that does not have a match in the normative pattern, a flag is set. So, in the end, when the anomalous instance is output by this implementation, there is an indicator next to each individual anomaly. An example of the textual output is shown in Figure 4.1.

Anomalous Instance(s):

```
v 58 v1  < -- anomaly
v 59 v2
v 60 v1
d 58 60 e2  < --  anomaly
d 59 58 e2
d 60 59 e1
(information_theoretic anomalous value = 2.000000)
```

Figure 4.1 Example output of GBAD-MDL showing anomalies and score.

4.1.3 Examples

The following are some simple examples of results obtained using our implementation of the GBAD-MDL algorithm described above.

First, take the fairly regular example shown in Figure 4.2.

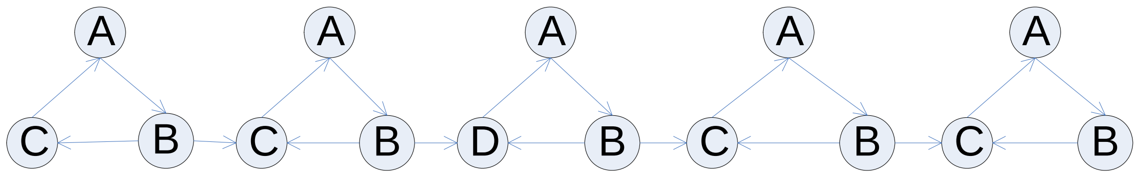

Figure 4.2 Simple graph for GBAD-MDL example.

Running the GBAD-MDL algorithm, the anomalous substructure, as shown in Figure 4.3, is:

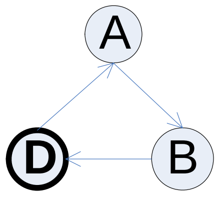

Figure 4.3 Anomalous substructure from simple graph using GBAD-MDL.

which is exactly the desired result. (The individual anomaly is in **bold**.) It should also be noted that no other substructures were reported as anomalous.

The above is similar to the example that was presented in the paper by Noble and Cook [Noble and Cook 2003], as shown in Figure 4.4.

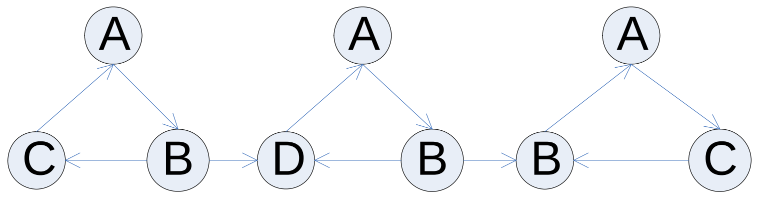

Figure 4.4 Noble and Cook example.

Running the GBAD-MDL algorithm on this example also results in the same anomalous substructure as shown in Figure 4.3. In Noble and Cook's approach, the **D** vertex is shown to be the anomaly, as they use a combination of size and number of instances to determine the anomalousness of a substructure. Similarly, GBAD-MDL also indicates that **D** is anomalous. The importance of this new approach is that a larger picture is provided regarding its associated substructure. In other words, not only are we providing the anomaly, but we are also presenting the context of that anomaly within the graph.

Another common real-world graph structure is a star-cluster configuration (e.g., shipping manifests), like the example shown in Figure 4.5.

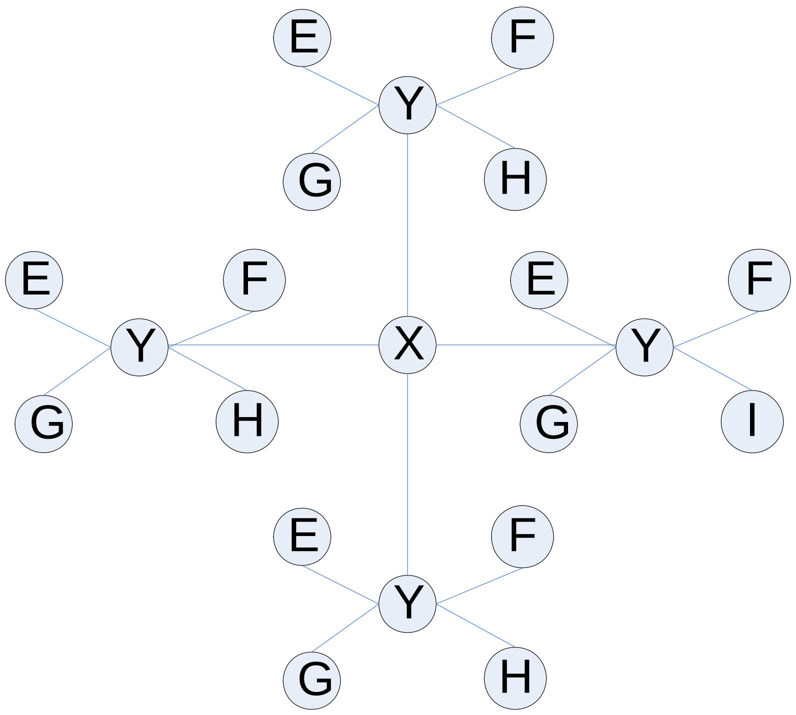

Figure 4.5 Star example.

Again, for this simple example, GBAD-MDL is correctly able to find the anomaly and its associated substructure, as shown in Figure 4.6.

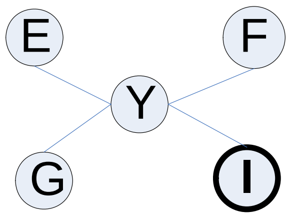

Figure 4.6 Anomalous structure from star example using GBAD-MDL.

4.2 Probabilistic Algorithm (GBAD-P)

The GBAD-P algorithm also uses the MDL evaluation technique to discover the best substructure in a graph, but instead of examining all instances for similarity, this approach examines all extensions to the normative substructure (pattern), looking for extensions with the lowest probability. The subtle difference between the two algorithms is that GBAD-MDL is looking at instances of substructures with the same characteristics (i.e., size, degree, etc.), whereas GBAD-P is examining the probability of extensions to the normative pattern to determine if there is an instance that when extended beyond its normative structure is traversing edges and vertices that are probabilistically less than other extended instances.

The high-level approach for the GBAD-P algorithm is:

- For a graph *G*, find the best substructure *S* that minimizes the description length of *G*.
- Compress *G* using *S*.
- For the newly compressed graph *G*
    - Find the single edge and vertex extension *E* that has the lowest probability *P* of existence from instances *I* of *S*.
    - Output instance I_n and *E* whose *P* is minimum.
    - Set *S'* to instance I_n's substructure.
- Compress *G* using *S'*, and repeat the above steps if there are still other extensions of the normative pattern to consider.

At each iteration, the result will be the instance that consists of the best substructure pattern and an extension with the lowest probability of existence. The value associated with this instance represents the lowest frequency, where the lower the value, the more anomalous the structure.

4.2.1 Algorithm

In Algorithm 2, the probability *P* allows the user to specify how much probability of existence they are willing to accept for a substructure to be considered anomalous. The number of iterations *N* allows the user to specify how far beyond the normative pattern they want to extend to look for anomalies. The detailed GBAD-P algorithm is as follows:

Algorithm 2: proc **GBAD-P** (graph *G*, probability *P*, iterations *N*)

```
1. Find normative substructure $S$ minimizing $DL(S)+DL(G|S)$; where $I$ are instances
   of $S$ in $G$.
2. Compress $G$ by $S$, where all instances $I$ of $S$ in $G$ are each replaced by a new
   vertex $V$.
3. Iterate over each new vertex $V$, extending each vertex $V$ by all possible single
   edges $E$.
4. For all instances $I'$, where each instance of $I'$ consists of $V$ and a unique extension,
   a substructure $S'$ consists of all matching instances $A$ from instances $I'$.
5. For each instance $A_k$ , anomalyScore($A_k$) = number of instances of $S'$ / $|I'|$
6. Return each instance $A_k$ with minimal anomalyScore $< P$.
7. Set $S$ to substructure definition of the $A_k$ with minimal anomaly score.
8. If current iteration $< N$, start next iteration at step 2.
```

anomalyScore(I_k) is the *probability* that a given instance should exist given the existence of all of the extended instances. Again, the lower the value, the more anomalous the instance. Given that $|I_n|$ is the total number of possible extended instances, freq(I_k) can never be greater, and thus the value of anomalyScore(I_k) will never be greater than 1.0.

4.2.2 Implementation

SUBDUE provides the ability to continually compress a graph, searching for the pattern that compresses the graph the most, and thus is ultimately the best substructure within the graph. GBAD-P takes a different approach by compressing the graph a single extension at a time, using the extensions that have the *least* probability of being part of the best substructure.

In order to implement the GBAD-P algorithm, we again used SUBDUE to discover the best substructure. In addition, we also used two other features provided by SUBDUE: maintaining a list of all instances that match the best substructure; iterating multiple times, compressing the graph by the best substructure at each iteration. When enough iterations are specified, SUBDUE terminates processing when any more attempts at compressing the graph would not result in a further reduction in its MDL. After the first iteration, where the graph is compressed by the normative pattern, the GBAD-P algorithm analyzes extensions from each instance of the best substructure at each iteration, looking for the ones with the lowest probability of occurring.

First, SUBDUE's logic for extensions is modified to only extend one edge at each iteration. While the first iteration works as-is in terms of performing extensions in order to find the best substructure, subsequent iterations only process single edge extensions from the newly compressed substructure. This allows the GBAD-P algorithm to evaluate the probability of individual extensions.

Second, the algorithm modifies the best substructure list by finding the best substructure that contains the compressed normative pattern from the first iteration. This is done to ensure that at each iteration we are still working from the normative pattern. The first substructure in the list that contains the compressed normative pattern is moved to the top of the list as the best substructure (since the list is already in order by value).

Third, for the newly defined best substructure, all of its instances are evaluated in terms of their probability among themselves. For each instance, a simple evaluation is calculated where the probability of the instance is the number of matching instances divided by the total number of instances, all within the list of instances for the best substructure. This value is then set as the anomalous score for the corresponding instance.

After each iteration, our GBAD-P implementation prints the anomalous instance (or instances, if their anomalous scores are equal). The output is similar to what is produced by the GBAD-MDL algorithm, except that the score is a value from 0.0-1.0, and it is done after each iteration (except for the first). By doing this over each iteration, it allows one to view the growth of the anomaly, one edge at a time.

4.2.3 Examples

The following are some simple examples of results obtained using our implementation of the GBAD-P algorithm described above.

First, take the fairly regular example shown in Figure 4.7.

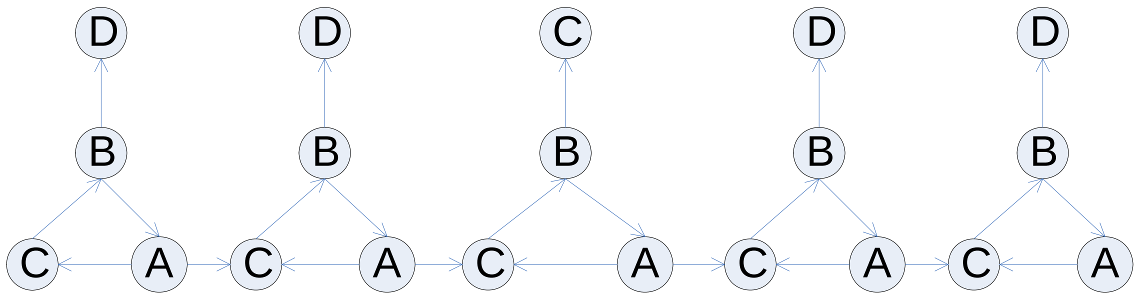

Figure 4.7 Simple graph for GBAD-P example.

After one iteration, the best substructure is shown in Figure 4.8.

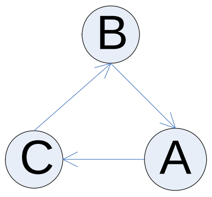

Figure 4.8 Best substructure from simple graph for GBAD-P example.

Then, on the second iteration, this substructure is compressed to a single vertex, extensions are evaluated, and the resulting anomalous substructure is shown in Figure 4.9 (with the compressed substructure expanded just for this visualization).

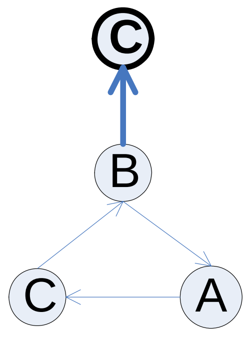

Figure 4.9 Anomalous substructure from simple graph using GBAD-P.

Clearly the anomalous substructure shown in Figure 4.9 is the desired result as the extension to **C** is less probable than the other **D** extensions.

Let us take another example, this time of a more network-type structure, as shown in Figure 4.10.

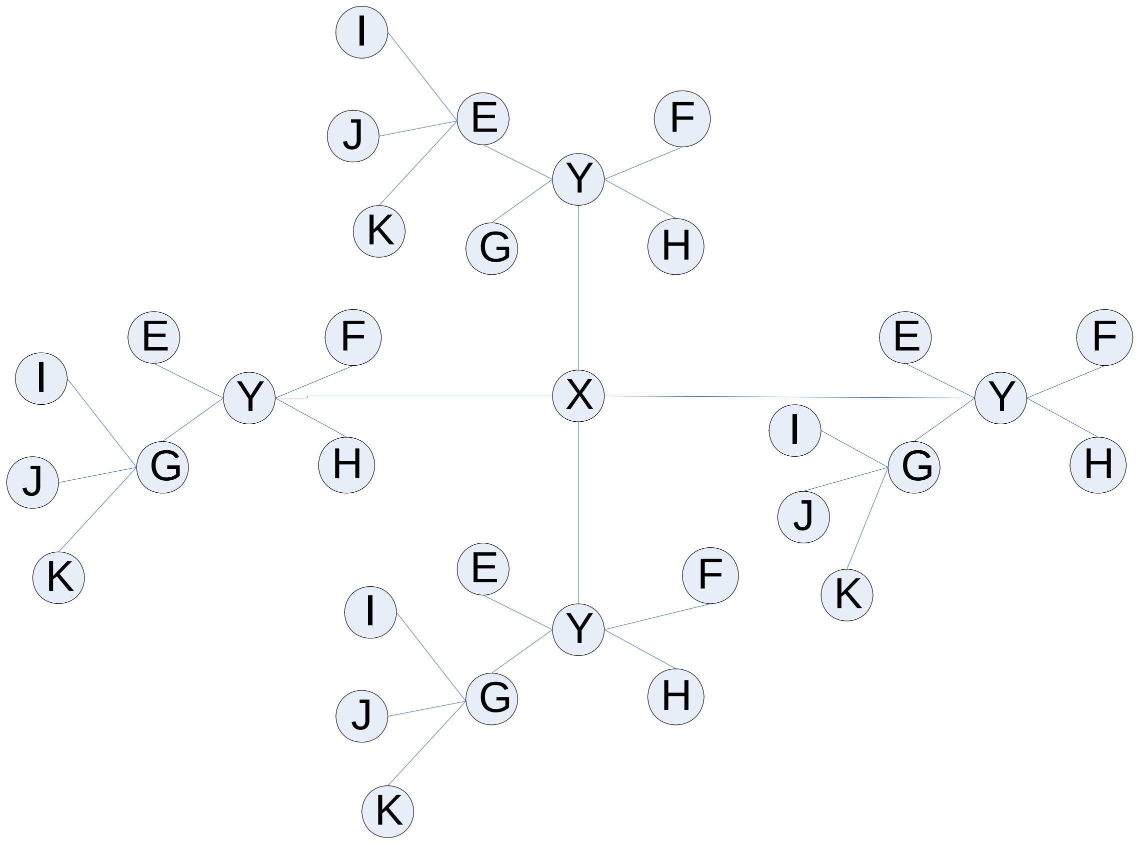

Figure 4.10 Network-type graph.

In Figure 4.10**,** there is a central node (labeled **X**) with four connected identical star structures (each with a center node labeled **Y**). Each of these star structures has an identical smaller substructure (made up of vertices labeled **I**, **J** and **K**) connected to it. However, one of the star structures has the **IJK** substructure connected to its vertex labeled **E**, while the others have it connected to their vertex labeled **G**.

Running the GBAD-P algorithm on this graph results in the following three structures labeled as anomalous, as shown in Figure 4.11 (after the second iteration).

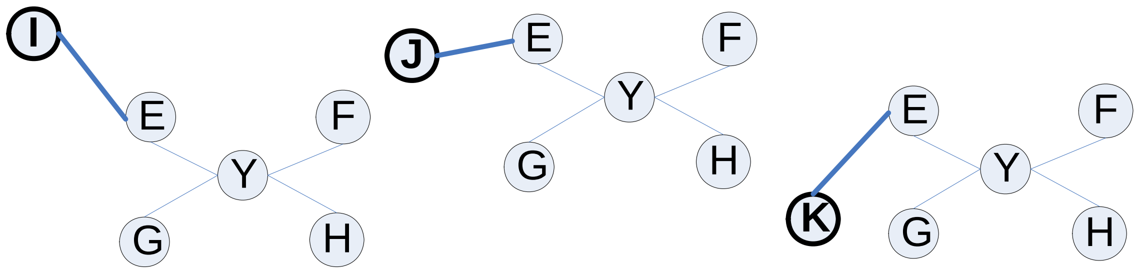

Figure 4.11 Anomalous insertions on network-type example.

So, in essence, while it did report the anomaly as three different substructures (all equal in probability), the complete anomaly is discovered. It should also be noted that on subsequent iterations, no more anomalous substructures are found. (All of the subsequent candidates have a probability of 100%.) This is because on the following iteration, the instances of the best substructure are compressed to a single vertex, and the other vertices (**I**, **J** and **K**), are linked to that single vertex, with no former knowledge of where they linked (i.e., whether they linked to **E** or **G**). Possible future work could include a modification to this approach to keep track of the original connections for further evaluation.

4.3 Maximum Partial Substructure Algorithm (GBAD-MPS)

The GBAD-MPS algorithm uses the MDL approach to discover the best substructure in a graph, and then it examines all of the instances of ancestral substructures that are missing various edges and vertices.

The high-level approach for the GBAD-MPS algorithm is:

- For a graph *G*, find the best substructure *S* that minimizes the description length of *G*.
- Find the instances of ancestor substructures of *S* (we will refer to the ancestor substructures S_n*)*.
- Output all instances I_k of S_n that are not part of any instances of *S*, and that minimize (cost * frequency).

The result will be those instances that are the maximum possible partial substructures to the normative (or best) substructure. The value associated with the instances represents the cost of transformation (i.e., how much change would have to take place for the instance to match the best substructure). Thus, the instance with the lowest cost transformation (if more than one instance have the same value, the frequency of the instance's structure will be used to break the tie if possible) is considered the anomaly, as it is closest to the best substructure without being included on the best substructure's instance list.

4.3.1 Algorithm

The key to Algorithm 3 is the building of the substructure list S_n, which are all substructures that are ancestral substructures to the normative substructure *S*, meaning they share common vertices and edges. The detailed GBAD-MPS algorithm is as follows:

Algorithm 3: proc **GBAD-MPS** (graph G, cost C)

```
1. Find normative substructure $S$ minimizing $DL(S)+DL(G|S)$, where $I$ is the set of
   its instances.
2. For each $S_n$, in the set of previously-generated substructures, where $S_n \subseteq S$, let $I_n$
   be the set of instances of $S_n$.
3. For each instance $I_k$ in the set of instances $I_n$, where $0 <$ matchcost($I_k$,$S$) $<$ C,
   $I_k \not\subset I$ and $I_m$ are all instances in $I_n$ that are isomorphic to $I_k$
   a. anomalyScore($I_k$) = $| I_m |$ * matchcost($I_k$,$S$).
4. Return all instances $I_k$ having minimal anomalyScore.
```

By allowing the user to specify the cost threshold C, we can control the amount of “anomalousness” that we are willing to accept. By our definition of an anomaly, we are expecting low transformation costs (i.e., few changes for the anomalous instance to match the best substructure). With the GBAD-MPS algorithm, while the user-definable threshold is a value based upon the cost of transformation, the product of the number of instances and the cost is the final anomalous score. For instance, if there are two substructures X and Y that have the same minimal cost of transformation to matching the best substructure, but there are 3 instances of X and 2 instances of Y, the instances of the Y substructure would be output as the most anomalous.

Throughout this work, whenever we indicate a relationship between substructures as $x \subseteq y$, we are referring to the fact that x is a *subgraph* of y, rather than x is a subset of y. It should also be pointed out that $x \not\subset y$ is referring to the case where x is not a complete sub-instance of y.

4.3.2 Implementation

While building substructures, and ultimately searching for the best pattern, SUBDUE maintains a list of the instances that match the current best substructure. GBAD-MPS takes a different approach by constructing a history of the substructures, allowing for the eventual analysis of ancestral substructures, and thus looking for those patterns that are structurally closest to the normative pattern, but are missing some structure.

In order to implement the GBAD-MPS algorithm, again we used SUBDUE to discover the best substructure. In addition, we also used another feature provided by SUBDUE: specifying the beam width of the search. By default, SUBDUE uses a beam width of 4 which signifies that it will only keep the top 4 substructures after evaluating each extension. While this heuristic has proven to be successful in SUBDUE's ability to discover the normative pattern, in order to be able to analyze substructures that never extended to the normative pattern, which is necessary for this algorithm, we need to extend the beam width so that other substructures can be evaluated for anomalies. This allows for us to keep track of those instances that are not direct ancestors of the normative pattern. In the end, SUBDUE terminates processing when there are no more extensions to candidate substructures, while the GBAD-MPS algorithm continues processing all of the ancestral substructures, looking for the one that is closest in transformation cost to the normative pattern

First, a list of substructures is maintained that consists of substructures (and their instances) that at some point during SUBDUE processing were used in evaluating their potential for being the normative pattern. Even if a substructure fails to make the "best" list at some point, it is still maintained on this list as a possible anomalous substructure. While this list can be rather large (and deserves some future memory-saving analysis), since the normative pattern is not known at this point, it has to be maintained until the final evaluation.

Second, the algorithm takes this list of substructures and compares each substructure to the normative pattern. If a substructure matches within the user specified anomalous threshold (cost of transformation), each of its instances is compared to the instances of the normative pattern. If an instance overlaps one of the normative pattern's instances (i.e., all of its edges and vertices are found in one of the normative instances), the instance is thrown out because it is considered one that could eventually extend to the normative pattern.

Third, each instance in the candidate list of instances is given an anomalous score equal to its cost of transformation (for transforming the instance into the normative pattern). Then, for each instance in the list that is isomorphic to another instance in the list, its anomalous score is increased by the value of its cost of transformation (i.e., cost of transformation * frequency).

In the end, our GBAD-MPS implementation prints the anomalous instance (or instances, if their anomalous scores match). This output is a little different from the other two algorithms in that no anomalous vertices and edges are indicated, just the entire anomalous instance. Since what is anomalous is the lack of structure, a comparison of the normative substructure to the anomalous substructure yields the anomalous differences.

4.3.3 Examples

The following are some simple examples of results obtained using our implementation of the GBAD-MDL algorithm described above.

First, take the example shown in Figure 4.12.

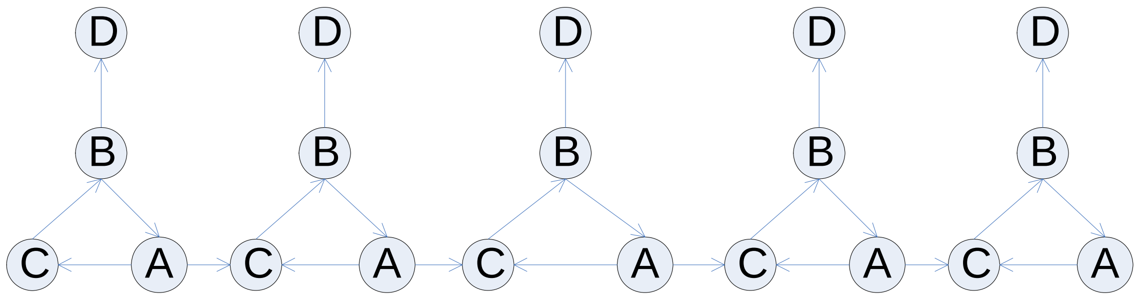

Figure 4.12 Simple graph for GBAD-MPS example.

The normative pattern (best substructure) from this graph is shown in Figure 4.13.

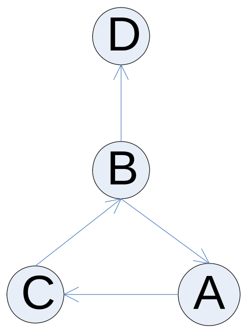

Figure 4.13 Normative pattern from simple graph for GBAD-MPS example.

Now, suppose we remove one of the edges and its associated vertex, from one of the instances of this normative pattern, creating the graph shown in Figure 4.14.

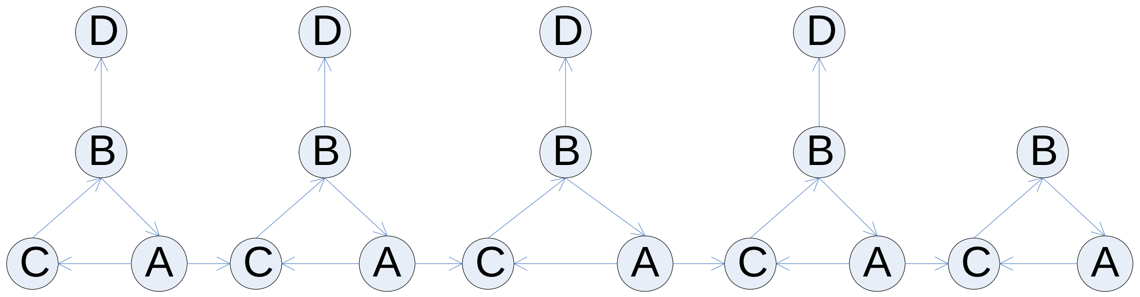

Figure 4.14 Simple graph for GBAD-MPS example with deleted vertex and edge.

In other words, we removed one of the **D** vertices and its associated edge. Running the maximum partial substructure approach on this modified graph, results in the anomalous instance shown in Figure 4.15.

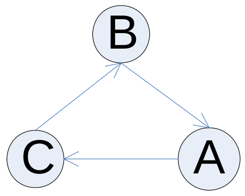

Figure 4.15 Anomalous instance from deletion example when using GBAD-MPS.

However, this pattern is common to all of the normative instances. So, for usefulness, our implementation reports the actual anomalous graph instance as specified in the input graph file (for example):

```
v 17 C
v 18 B
v 19 A
d 17 18 label
```

d 19 17 label

In this example, the actual vertex and edge numbers, as well as their labels, are output, allowing the user to go directly to the specific anomalous instance.

Now, take the example shown in Figure 4.16.

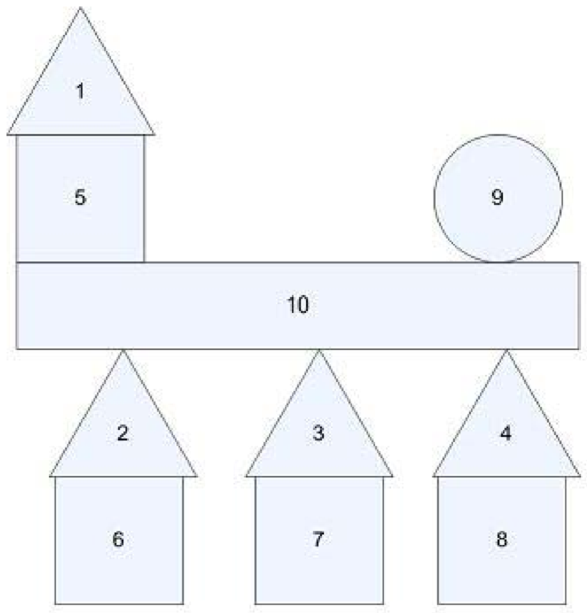

Figure 4.16 Geometric representation of example.

We can convert the geometric representation of Figure 4.16 into the graph representation shown in Figure 4.17.

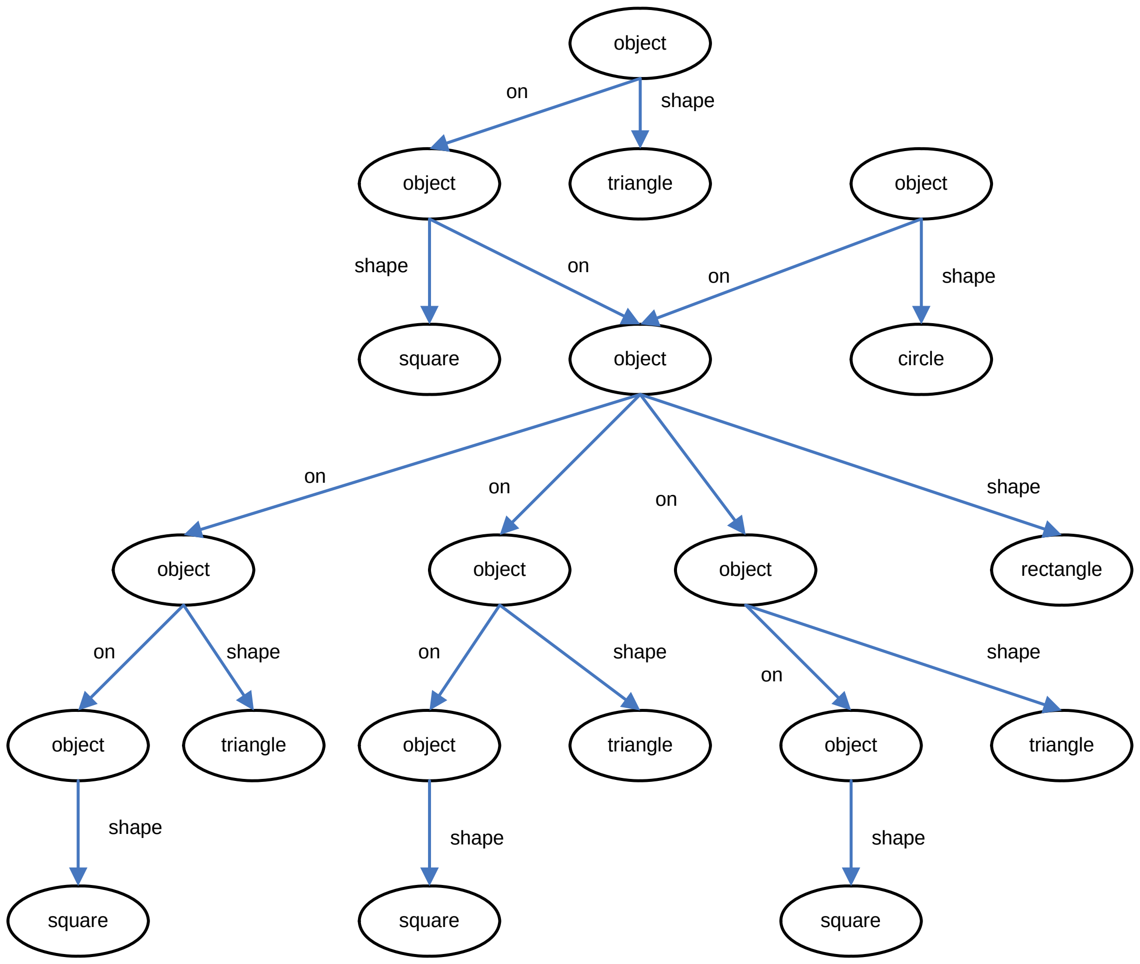

Figure 4.17 Larger graph for GBAD-MPS example.

The normative pattern from this graph is shown in Figure 4.18.

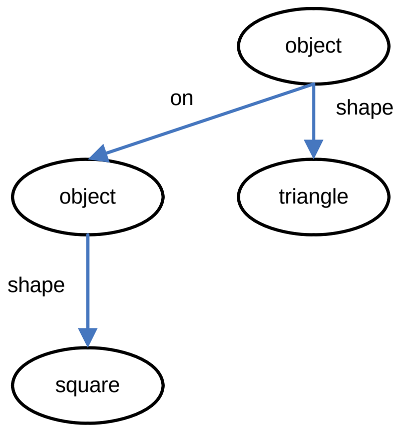

Figure 4.18 Normative pattern from larger graph used for GBAD-MPS example.

Suppose we remove one of the edges and its associated vertex from one of the instances of this normative pattern, creating the graph shown in Figure 4.19.

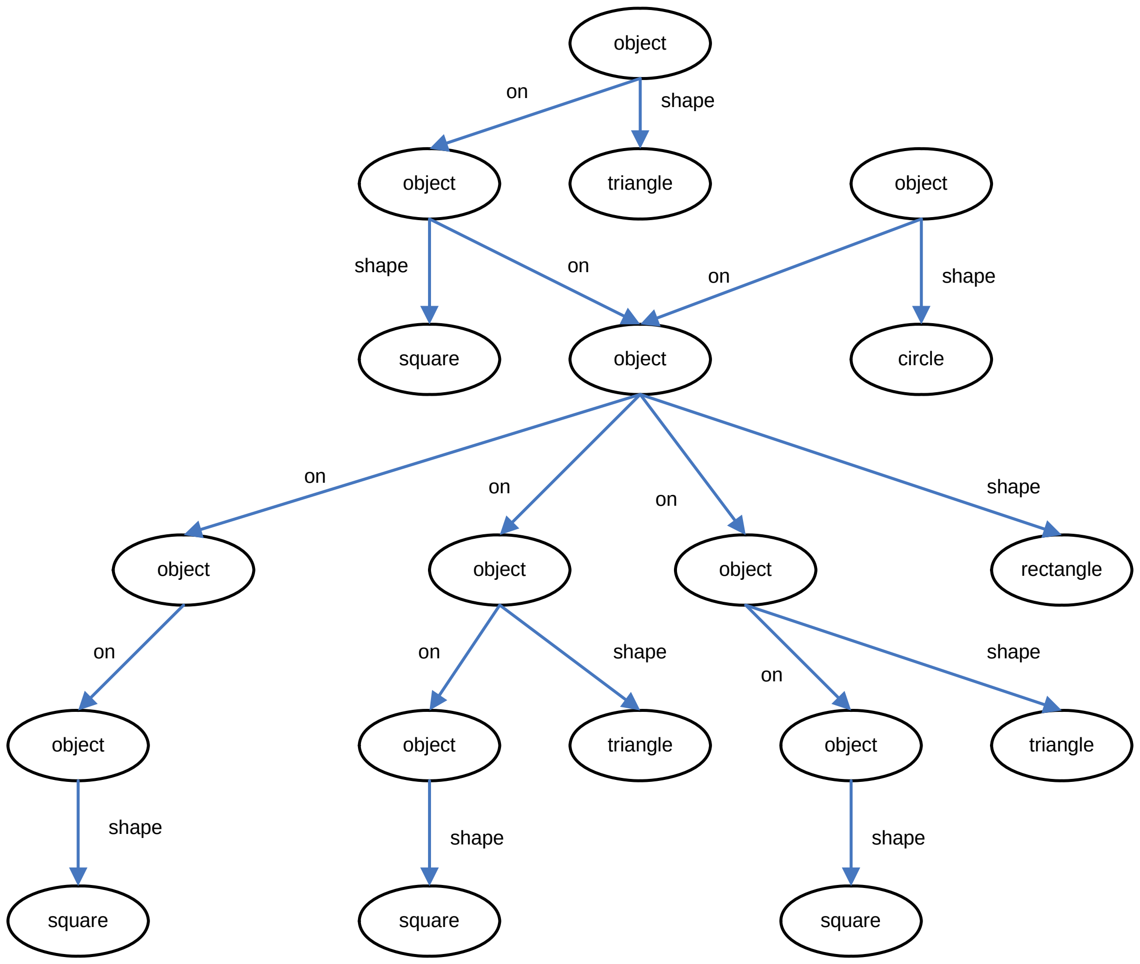

Figure 4.19 Larger graph, used for GBAD-MPS example, with deleted vertex and edge.

Notice that even on this simple example, a quick eye-ball comparison of the difference between the two graphs is not easy. In this case, the parts shown in Figure 4.20 were removed.

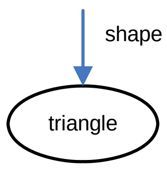

Figure 4.20 Parts removed from larger graph in GBAD-MPS example.

Running the GBAD-MPS algorithm on this modified graph results in the anomalous instance shown in Figure 4.21 being discovered.

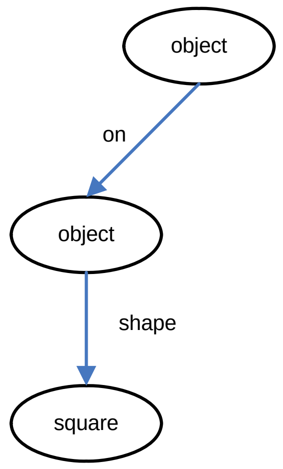

Figure 4.21 Anomalous instance from larger graph when using GBAD-MPS.

So, in this case, the instance was anomalous because it did not contain the “triangle” node or its ”shape” edge.

4.4 Summary

Each of these algorithms was designed to address one of three types of graph alterations. Knowing that each of the types of anomalies consists of one of these changes, we know that if we address the three possible graph changes, we will be able to address all of the possible structural anomalies in a graph. The GBAD-MDL algorithm, using the minimum description length principle, is designed to uncover label modifications; the GBAD-P algorithm, with a probabilistic approach, is setup to discover additional edges and vertices; and the GBAD-MPS algorithm, which looks for substructures that almost extended to the best pattern, is built to retrieve instances that are missing graph parts. While there was nothing algorithmically prohibiting us from building a single algorithm for handling the discovery of anomalous modifications, insertions and deletions, we would have to significantly open the search space so that we are examining substructures consisting of all three anomalous types, and thus a wider deviation from the normative pattern. Whereas, if we maintain these algorithms separately, running each of them on a targeted data set, less memory and processing will be used in each approach, as each algorithm would only need a search space for its particular anomalous deviation to the normative pattern.

One of the advantages of these algorithms lies in the fact that they are not implementation dependent. While we are using GBAD as a tool to run these algorithms, they can be implemented with any graph-based tool, as long as that tool maintains a list of substructures and instances that are being evaluated. Another advantage is that these algorithms do not just return the pattern of the anomaly – they also return the actual anomalous instances within the data. In a real-world scenario, that can be invaluable to an analyst who may need to act upon a fraud situation before the losses are too great. The disadvantage of these algorithms is that they are focused on specific anomalies: modifications, insertions and deletions. Thus, in a real-world setup, it would require that all three algorithms be used in conjunction, as the type of anomaly would most likely be unknown and possibly a combination of the different anomaly types.

CHAPTER 5

EMPIRICAL EVALUATIONS ON SYNTHETIC DATA

The following sections contain the results when applying the algorithms to randomly-generated synthetic data. Creating graphs of varying sizes, with normative patterns and anomalies of different sizes, our algorithms are empirically evaluated as to their effectiveness. Later in this chapter we present results using multiple anomalous types, normative patterns of different shapes and overlapping instances. We then conclude with some of the known limitations of the algorithms, as well as the running-time performances of each of the approaches.

5.1 Graph Generation

Synthetic graphs were created using a tool called *subgen* that generates random graphs based upon user-specified parameters, including:

- total number of vertices and edges
- list of possible vertex and edge labels and their probabilities
- substructure pattern
- amount of connectivity
- amount of overlap

Using these parameters, *subgen* computes the number of instances that need to be generated by calculating the size of the graph and dividing by the size of the substructure pattern. If an overlap percentage is specified, the amount of common structure is calculated for each instance, and instances are randomly merged until the specified overlap is achieved. After the graph is built from these instances, random vertices (based upon their probabilistic ratios) are added in order to achieve the desired graph size. Then random edges (again based upon their probabilistic ratios) are added in order to achieve the specified connectivity level. Finally, any additional edges are added in order to achieve the desired graph size. As a result of running *subgen*, two output files are created: the graph file and a file containing the list of substructure instances.

5.2 Synthetic Data

In order to create our synthetic graphs, we use the *subgen* tool to randomly generate graphs, and use a tool we created to generate new modified graphs from these graphs where:

- AV is the number of anomalous vertices in an anomalous substructure
- AE is the number of anomalous edges in an anomalous substructure
- V is the number of vertices in the normative pattern
- E is the number of edges in the normative pattern

Each synthetic graph consists of substructures containing the normative pattern (with V number of vertices and E number of edges), connected to each other by one or more random connections, and each synthetic experiment consists of AV number of vertices and AE number of edges altered.

For *modification* anomalies: an AV number of vertices and AE number of edges, from the same randomly chosen normative instance, have their labels modified to randomly chosen, non-duplicating labels (e.g., we do not replace a vertex labeled “X” with another vertex also labeled “X”).

For *insertion* anomalies: randomly inserted AV vertices and AE edges, where the initial connection of one of the AE edges is connected to either an existing vertex in a randomly chosen normative instance, or to one of the already inserted AV vertices.

For *deletion* anomalies: randomly chosen AV vertices and AV edges, from a randomly chosen normative instance, are deleted along with any possible “dangling” edges (i.e., if a vertex is deleted, all adjacent edges are also deleted).

In order to test graphs and substructures of varying sizes, we will need to adjust the total number of vertices and edges, as well as the size of the substructure pattern. Also, in order to be consistent across all of our regular tests, we will choose a connectivity setting of 1.0, signifying that the graph is connected. The ability of the algorithms to discover anomalous instances in graphs that are not connected will be tested in a subsequent section on two real-world data sets: cargo shipments and network traffic. We have also chosen to have no overlapping instances, except for the special overlap tests, in Section 5.11, where we specifically examine the effectiveness of the algorithms on substructures whose instances share vertices and edges.

In addition, in order to better exemplify our definition of an anomaly, the distribution of randomly generated connections versus the anomalies is not the same. In other words, if randomly generated connections have the same frequency as the anomalies, it would be difficult to distinguish between noise and anomalous behavior. Since our definition explicitly states that an anomaly is a slight unexpected deviation from the normal (to better hide the true nature of the action), its existence can not be as common as normal behaviors. Now, that does not mean that anomalies are completely different from noise, just that their frequency is not as prevalent. So, in order to achieve our goal, we will define in the *subgen* tool a distribution of 10-to-1 (i.e., random normal connections are ten times more likely than an anomaly). Because the number of vertices and edges will vary among test sets, their label probabilities (i.e., the probability that a label will be used when vertices and edges are added in order to achieve the proper graph size and connectivity) are adjusted in order to reflect the desired ratio of noise to anomaly.

It should also be noted that our tool for generating random graph edge/vertex insertions does not create self-edges. This was done because compression would result in an edge pointing from the compressed substructure to itself, and would not be considered in future extensions. In other words, we are not considering a normative pattern connected to itself.

Also, due to our definition of an anomaly, all tests will be limited to changes that constitute less than 10% of the normative pattern. Again, since anomalies are supposed to represent slight deviations in normal patterns, an excessive change to a pattern is irrelevant. However, in order to analyze the upper bounds effectiveness of these algorithms, we will also perform some tests at deviations above 10%.

Each of the above is repeated for each algorithm, varying sizes of graphs, normative patterns, thresholds, iterations and sizes of anomalies (where the size of the anomaly is $|AV| + |AE|$). Also, due to the random nature in which structures are modified, each test will be repeated multiple times to verify its consistency.

5.3 Metrics

Each test consists of a single graph from which 30 randomly altered graphs are generated. The output results consist of running the algorithms against those 30 graphs for the specified settings. The primary three metrics calculated are:

1. Percentage of runs where the *complete* anomalous substructure was discovered.
2. Percentage of runs where at least *some* of the anomalous substructure was discovered.
3. Percentage of runs containing *false positives*.

After the algorithm has completed running, the first metric represents the percentage of success when comparing the results to the known anomalies that were injected into the data. If all of the anomalies are discovered for a particular run, that is counted as a success for that run. For example, if 27 out of the 30 runs found all of their anomalies, the value for this metric would be 90.0.

The second metric represents the percentage of runs where at least some part of the injected anomaly was discovered. For example, if the anomaly consisted of 3 vertices and 2 edges that had their labels changed, and the run reported only one of the anomalous vertices, then that run would be considered a success for this measurement. Obviously, this metric will always be at least as high as the first metric.

The last metric represents the percentage of runs that reported at least one anomaly that was not one of the injected anomalies. Since it is possible that multiple reported anomalous instances could have the same anomalous value, some runs may contain both correct anomalies and false ones. Further tuning of these algorithms may enable us to discover other measurements by which we could “break the tie” when it comes to calculating an anomalous score.

There will also be some run-time statistics showing the performance of the algorithms.

5.4 Shapes

In addition to the types of anomalous changes, the “shape” of the normative pattern may affect the algorithms’ abilities to discover the anomalies. The types of normative patterns that we are going to consider are:

- single star
- multiple connected stars (*cluster*)
- single strand
- network (*cycle*)

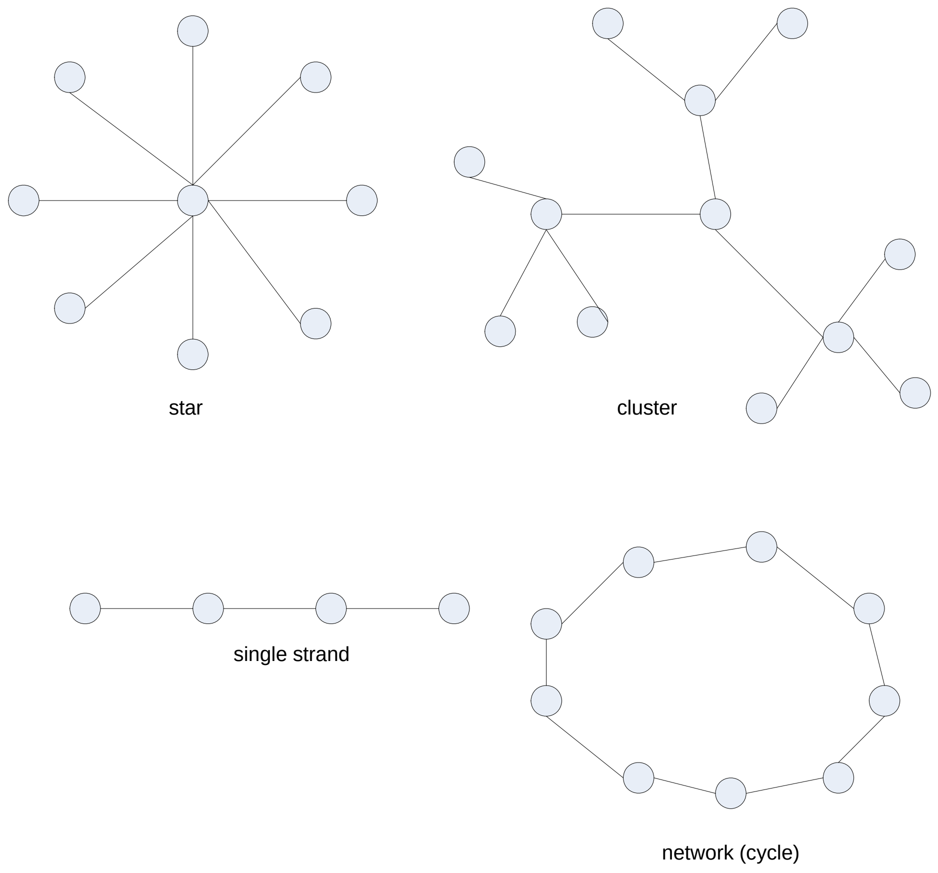

Figure 5.1 Example shapes.

In the following tests, we chose to use the *cluster* pattern as the baseline for the tests. While we could have chosen any of these patterns, we felt that this pattern was the most representative of the types of real-world data that these approaches would be used for discovering anomalies. For example (which will be shown in later tests), cargo shipments consist of ships, manifests, ports, etc… An appropriate representation of this type of data would be where certain entities are hubs for common information (e.g., shipper, port, etc.), such that multiple manifests could share identical information.

There are many other types of pertinent information, such as telecommunication call records, terrorist networks, financial transactions, etc. which share this same cluster type of topology.

We will include another section later in this chapter that will test for each of the patterns identified in Figure 5.1.

5.5 Information Theoretic Results

In this section, and the subsequent two sections, tables and figures will represent a summary of the results obtained for the different tests. For each approach, the synthetic experiments will represent *modifications* (i.e., vertices and edges modified within an instance of the normative pattern), *insertions* (i.e., additional vertices and edges connected to the normative pattern), and *deletions* (i.e., vertices and edges missing from what could have been an instance of the normative pattern). While each of the algorithms was designed to handle different types of anomalies, each approach will be evaluated as to their effectiveness across all types.

It should be noted that for each of the following GBAD-MDL tests, to improve performance, we will prune substructures whose values are less than their parent's. While this could result in a loss of accuracy, previous testing has shown this to not be the case in most situations.

5.5.1 Modifications

Figure 5.2 and Figure 5.3 show the effectiveness of the GBAD-MDL approach on graphs of varying sizes, with a normative pattern of 10 vertices and 10 edges, with random anomalous modifications. In these figures (and subsequent figures), the X-axis represents the thresholds, the Y-axis is the percentage of anomalies discovered, and the Z-axis indicates the sizes of the normative patterns, graphs and anomalies (i.e., <number of vertices in the normative pattern>/<number of edges in the normative pattern>/<number of graph vertices>/<number of graph edges>-<# of anomalous changes>).

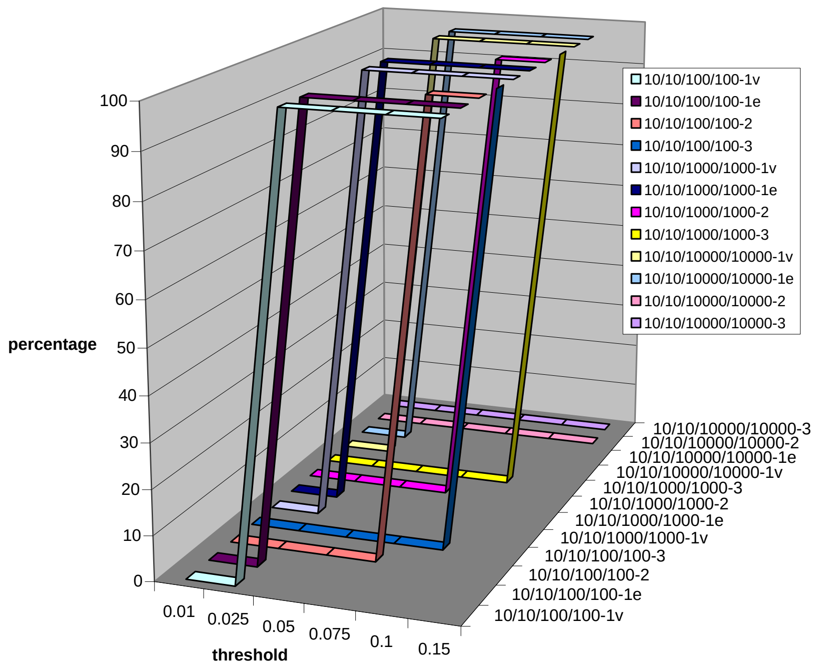

Figure 5.2 Percentage of GBAD-MDL runs where all anomalies discovered.

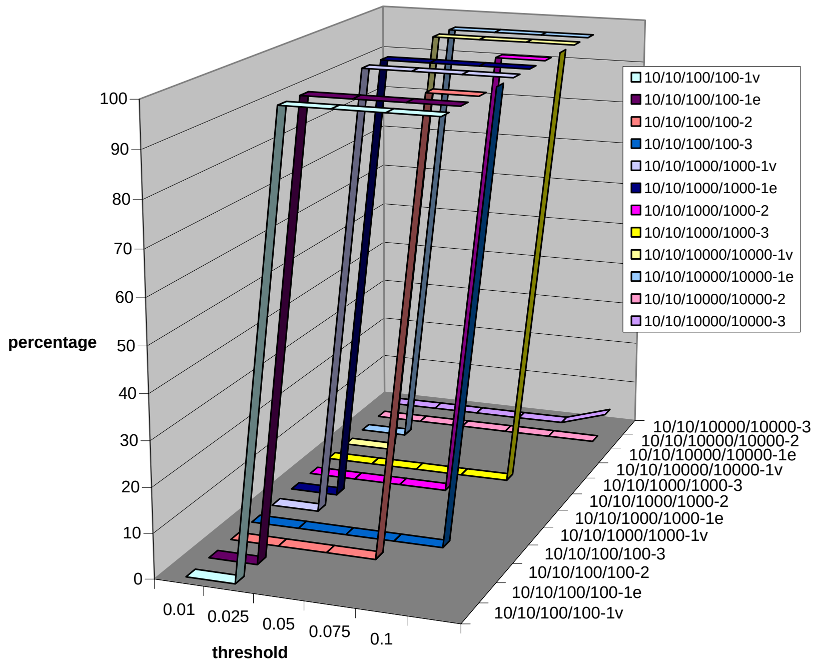

Figure 5.3 Percentage of GBAD-MDL runs where at least one anomaly discovered.

For the very small synthetic tests (not shown in Figures 5.2 and 5.3), when the threshold is high enough (i.e., the threshold is equal to or higher than the percentage of change), this approach is able to find all of the anomalies, and at no point were any false positives registered. For example, when the normative pattern is of size 6 (3 vertices and 3 edges), we have to set the threshold higher than 0.1. This is done because with a normative pattern of size 6, even just a change of a single vertex and a single edge would require a threshold of at least 0.33 in order to discover such an anomaly. Thus, when we set the threshold to 0.035, the algorithm is able to find 100% of the anomalies for a change of size 2. Similarly, when we set the threshold to 0.2, GBAD-MDL is able to find all of the single anomalous modifications (as 0.167% of the normative pattern is changed).

For all tests where the threshold is 0.1 or less, no false positives are reported. However, when we increase the threshold to 0.2, a few false positives are reported. For a threshold of 0.2, we are basically saying that we want to analyze patterns that are up to 20% different. Such a huge window results in some noise being considered (along with the actual anomalies, as all of the anomalous instances are discovered). Fortunately, our definition of what is truly an anomaly would steer us towards observing runs with lower thresholds.

In Figure 5.2 and Figure 5.3, we observe that no complete anomalies are discovered in the graphs of 10,000 vertices and 10,000 edges when the size of the anomaly is either 2 or 3. This is due to the fact that we get 100% false positives in both of those situations (actually, 87.88% in the case where the size of the anomaly is 3, but that is just coincidental), because every graph contains at least one substructure (and usually a few more) that consists of smaller changes (i.e., either a single vertex or a single edge). As the graph grows in size, more substructure connections are made in the random generation of the graph, leading to the likelihood that a substructure will be created that is similar to the normative pattern (i.e., within the inexact matching threshold) but consists of a smaller modification than the induced anomaly. This issue accentuates one of the drawbacks of the GBAD-MDL algorithm, in that this algorithm assumes that the anomaly is the *smallest* modification to the normative substructure. Further testing of this observation has shown that the anomaly is discovered, but the anomalous value is always higher than the noisy substructure (or substructures) that are able to score lower because they require fewer changes to match the normative pattern.

When the size of the normative pattern is larger, smaller thresholds can be used in order to uncover small changes to the structure. Figure 5.4, Figure 5.5 and Figure 5.6 represent the use of finer thresholds for the discovery of very small anomalies (1 to 3 changes) to a normative pattern of 30 vertices and 30 edges.

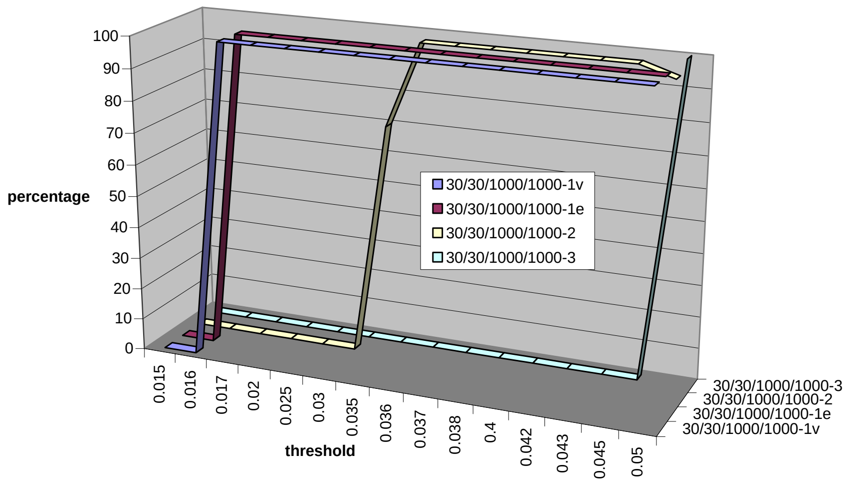

Figure 5.4 Percentage of GBAD-MDL runs where all anomalies are discovered (finer granularity).

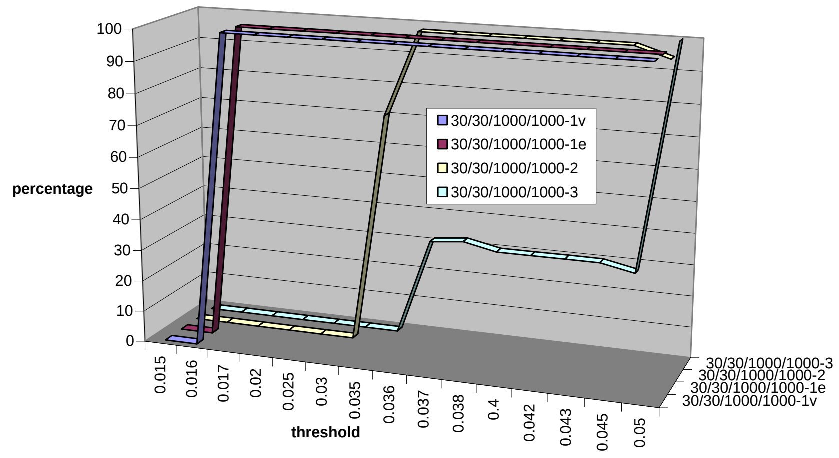

Figure 5.5 Percentage of GBAD-MDL runs where at least one anomaly is discovered (finer granularity).

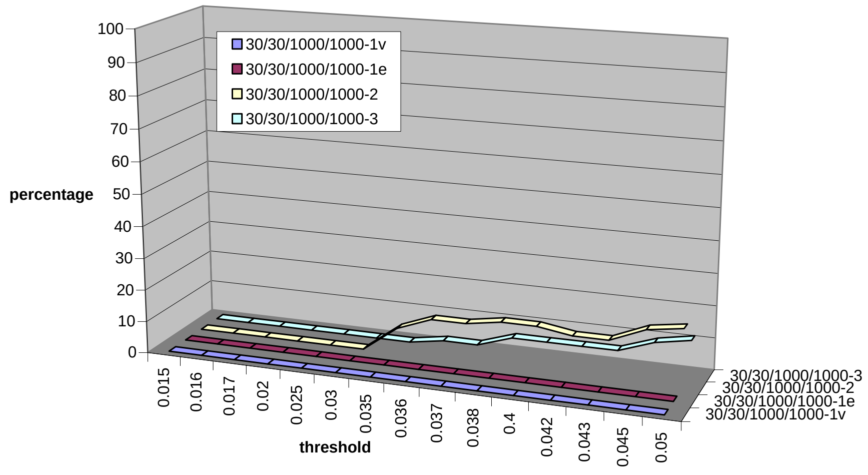

Figure 5.6 Percentage of GBAD-MDL runs with false positives (finer granularity).

5.5.2 Insertions

Figure 5.7, Figure 5.8 and Figure 5.9 show the effectiveness of the GBAD-MDL approach on graphs of varying sizes with random anomalous insertions.

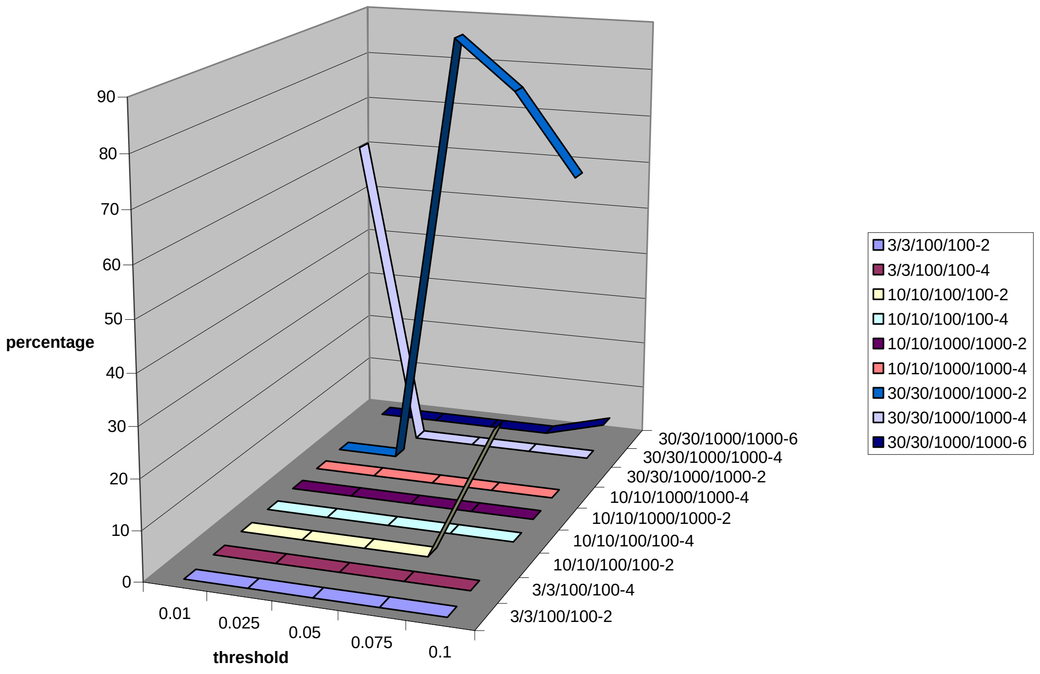

Figure 5.7 Percentage of GBAD-MDL runs where all anomalous insertions are discovered.

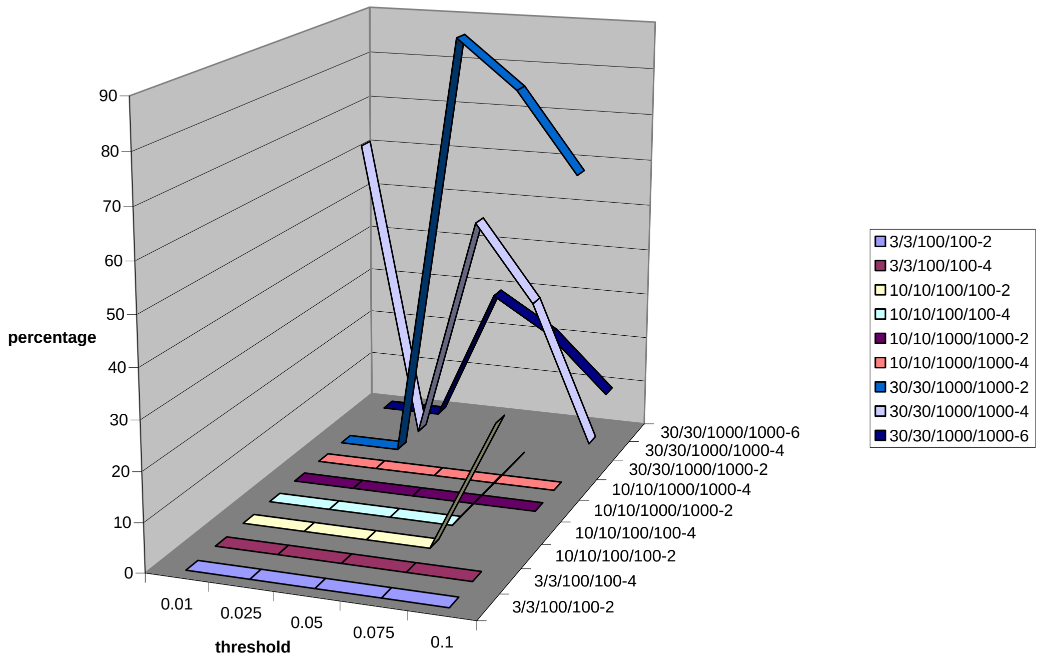

Figure 5.8 Percentage of GBAD-MDL runs where at least one anomalous insertion is discovered.

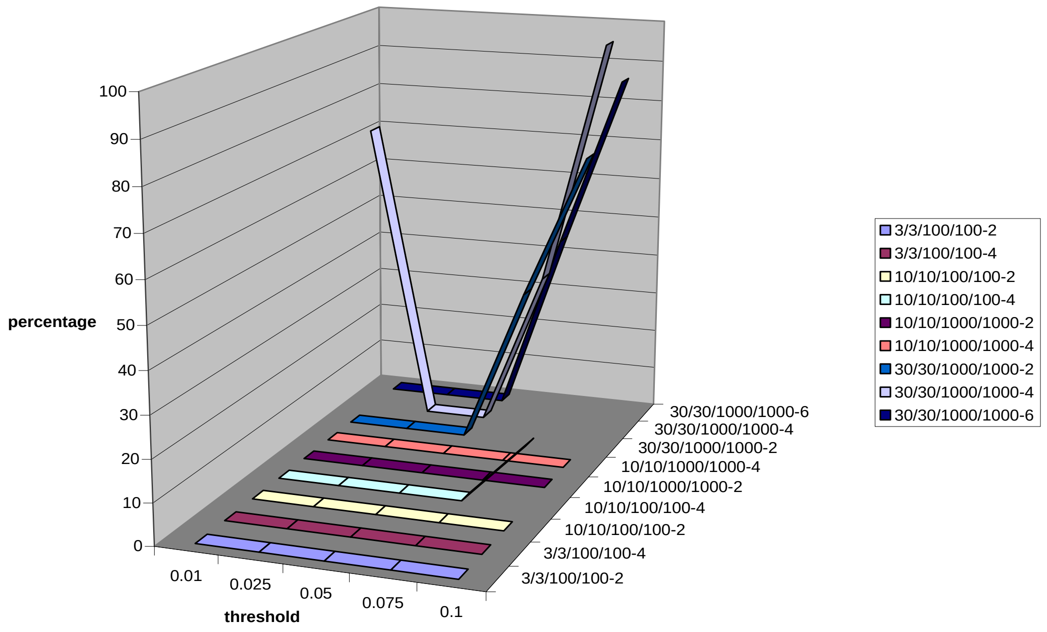

Figure 5.9 Percentage of GBAD-MDL runs on anomalous insertions containing false positives.

On the smaller graphs, no matter how large the anomaly or the size of the threshold, none of the anomalous insertions are discovered. Even when unrealistically large thresholds (i.e., above 0.1) are used, we are unable to find any of the insertion anomalies.

Two observations about the effectiveness of this algorithm on anomalous insertions are evident in these results. First, in the situation where the anomaly is a single edge and vertex, there is some success because the insertion is “close” to the normative pattern (a direct connection). Second, the effectiveness clearly drops off as the insertions get further away from the normative pattern, and the few successes can be attributed to the random insertions being close to the normative pattern (e.g., an anomaly of size 4 that consists of two edges and two vertices that are directly connected to the normative pattern via different vertices).

This is clearly not an effective solution for anomalous insertions.

5.5.3 Deletions

The following tables show the effectiveness of the GBAD-MDL approach on graphs of varying sizes with random anomalous deletions. It should be noted that we chose to represent these results as a table because of the scarcity of non-zero results and the relatively few differences in values.

Table 5.1 Percentage of GBAD-MDL runs where all anomalous deletions are discovered.

Graph Size (Norm Pattern) <anomaly size>	0.01	0.025	0.05	0.075	0.1	0.2	0.3	0.35
100 vertices/100 edges (3 vertices/3 edges)								
<1 vertex/1 edge>	---	---	---	---	---	0	0	0
<2 vertices/2 edges>	---	---	---	---	---	0	0	0
100 vertices/100 edges (10 vertices/10 edges)								
<1 vertex>	---	---	0	0	0	0	---	---
<1 edge>	---	---	0	0	0	20.0	---	---
<1 vertex/1 edge>	---	---	0	0	0	0	---	---
<2 vertices/1 edge>	---	---	0	0	0	0	---	---
1000 vertices/1000 edges (10 vertices/10 edges)								
<1 vertex>	---	0	0	0	0	0	---	---
<1 edge>	---	0	0	0	0	0	---	---
<1 vertex/1 edge>	---	0	0	0	0	0	---	---
<2 vertices/1 edge>	---	0	0	0	0	0	---	---
1000 vertices/1000 edges (30 vertices/30 edges)								
<1 vertex>	0	0	3.33	0	0	---	---	---
<1 edge>	0	0	0	3.33	0	---	---	---
<1 vertex/1 edge>	0	0	0	0	0	---	---	---
<2 vertices/1 edge>	0	0	0	0	0	---	---	---
<2 vertices/2 edges>	0	0	0	0	0	---	---	---
<2 vertices/3 edges>	0	0	0	0	0	---	---	---
<3 vertices/3 edges>	0	0	0	0	0	---	---	---
<4 vertices/3 edges>	0	0	0	0	0	---	---	---

Table 5.2 Percentage of GBAD-MDL runs where at least one anomalous deletion is discovered.

Graph Size (Norm Pattern) <anomaly size>	**0.01**	**0.025**	**0.05**	**0.075**	**0.1**	**0.2**	**0.3**	**0.35**
100 vertices/100 edges (3 vertices/3 edges)								
<1 vertex/1 edge>	---	---	---	---	---	0	0	0
<2 vertices/2 edges>	---	---	---	---	---	0	0	0
100 vertices/100 edges (10 vertices/10 edges)								
<1 vertex>	---	---	0	0	0	30.0	---	---
<1 edge>	---	---	0	6.67	43.33	93.33	---	---
<1 vertex/1 edge>	---	---	0	0	0	25.0	---	---
<2 vertices/1 edge>	---	---	0	0	0	13.33	---	---
1000 vertices/1000 edges (10 vertices/10 edges)								
<1 vertex>	---	0	0	0	0	0	---	---
<1 edge>	---	0	0	0	0	0	---	---
<1 vertex/1 edge>	---	0	0	0	0	0	---	---
<2 vertices/1 edge>	---	0	0	0	0	1.11	---	---
1000 vertices/1000 edges (30 vertices/30 edges)								
<1 vertex>	0	0	33.33	0	6.67	---	---	---
<1 edge>	0	0	0	6.67	0	---	---	---
<1 vertex/1 edge>	0	0	8.33	0	0	---	---	---
<2 vertices/1 edge>	0	0	0	0	0	---	---	---
<2 vertices/2 edges>	0	0	0	0	0	---	---	---
<2 vertices/3 edges>	0	0	0	0	0	---	---	---
<3 vertices/3 edges>	0	0	0	0	0	---	---	---
<4 vertices/3 edges>	0	0	0	0	0	---	---	---

Table 5.3 Percentage of GBAD-MDL runs with anomalous deletions that return false positives.

Graph Size (Norm Pattern) <anomaly size>	**0.01**	**0.025**	**0.05**	**0.075**	**0.1**	**0.2**	**0.3**	**0.35**
100 vertices/100 edges (3 vertices/3 edges)								
<1 vertex/1 edge>	---	---	---	---	---	0	0	0
<2 vertices/2 edges>	---	---	---	---	---	0	0	0
100 vertices/100 edges (10 vertices/10 edges)								
<1 vertex>	---	---	0	0	0	61.7	---	---
<1 edge>	---	---	0	0	7.14	21.43	---	---
<1 vertex/1 edge>	---	---	0	0	0	53.12	---	---
<2 vertices/1 edge>	---	---	0	0	0	48.94	---	---
1000 vertices/1000 edges (10 vertices/10 edges)								
<1 vertex>	---	0	0	0	0	0	---	---
<1 edge>	---	0	0	0	0	0	---	---
<1 vertex/1 edge>	---	0	0	0	0	0	---	---
<2 vertices/1 edge>	---	0	0	0	0	0	---	---
1000 vertices/1000 edges (30 vertices/30 edges)								
<1 vertex>	0	0	0	0	96.97	---	---	---
<1 edge>	0	0	100	93.55	100	---	---	---
<1 vertex/1 edge>	0	0	16.67	0	0	---	---	---
<2 vertices/1 edge>	0	0	100	100	100	---	---	---
<2 vertices/2 edges>	0	0	100	100	100	---	---	---
<2 vertices/3 edges>	0	0	100	100	100	---	---	---
<3 vertices/3 edges>	0	0	100	100	100	---	---	---
<4 vertices/3 edges>	0	0	100	100	100	---	---	---

Due to the scarcity of anomalies being discovered and the high false-positive rates, it is clear that the GBAD-MDL is not useful in discovering anomalous deletions.

5.5.4 Denser Graphs

For the following tests, we increased the ratio of edges to vertices so as to create a denser graph. In this case, each vertex in the normative structure has a degree of 5 (i.e., five edges), as opposed to what was created for the previous tests where the number of edges matched the number of vertices, and the average degree was 2. The size of the source graph is approximately 1000 vertices and 3000 edges, with a normative pattern of 10 vertices and 30 edges.

Table 5.4 Percentage of GBAD-MDL runs on dense graphs where all anomalous modifications are discovered.

Size of Anomaly	0.01	0.025	0.05	0.075	0.1	0.125
1 vertex	0	100	100	100	100	36.67
1 edge	0	100	100	100	100	96.67
1 vertex/1 edge	0	0	100	100	100	13.33
2 vertices/1 edge	0	0	0	100	100	100
2 vertices/2 edges	0	0	0	0	80.0	80.0
2 vertices/3 edges	0	0	0	0	0	100

Table 5.5 Percentage of GBAD-MDL runs on dense graphs where at least one anomalous modification is discovered.

Size of Anomaly	0.01	0.025	0.05	0.075	0.1	0.125
1 vertex	0	100	100	100	100	73.33
1 edge	0	100	100	100	100	100
1 vertex/1 edge	0	0	100	100	100	13.33
2 vertices/1 edge	0	0	0	100	100	100
2 vertices/2 edges	0	0	0	0	95.0	95.0
2 vertices/3 edges	0	0	0	0	60.0	100

Table 5.6 Percentage of GBAD-MDL runs on dense graphs that return false positives.

Size of Anomaly	0.01	0.025	0.05	0.075	0.1	0.125
1 vertex	0	0	0	0	0	46.34
1 edge	0	0	0	0	0	34.83
1 vertex/1 edge	0	0	0	0	0	71.43
2 vertices/1 edge	0	0	0	0	0	0
2 vertices/2 edges	0	0	0	0	0	0
2 vertices/3 edges	0	0	0	0	0	0

With a 100% discovery rate and no false positives, when the threshold is at 10% or less, these results confirm that even when the graphs are dense, the effectiveness of GBAD-MDL is not compromised. Due to the wide deviations in its discovery rate at a threshold above 10%, the effectiveness becomes sporadic as more noise is accepted into the evaluations, particularly when the anomalous change is minimal.

5.5.5 Extremes

In a few of the examples shown in the previous sections, we used thresholds and/or anomalies that were greater than the 10% which begins to violate our definition of an anomaly. This produced some interesting results that showed the effectiveness of the GBAD-MDL algorithm beyond the intended upper bounds.

In the following tables, notice the effectiveness of the information theoretic approach when the threshold used is greater than 0.1 for a graph of approximately 100 vertices and 100 edges, with a normative pattern of 10 vertices and 10 edges.

Table 5.7 Percentage of GBAD-MDL extreme runs where all anomalous modifications are discovered.

Size of Anomaly (% of normative pattern)	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**
2 (10%)	100	100	83.33	93.33	96.67
3 (15%)	0	96.67	30.0	100	93.33
4 (20%)	0	86.67	30.0	33.33	83.33
5 (25%)	0	0	16.67	13.33	40.0
6 (30%)	0	0	53.33	16.67	36.67
7 (35%)	0	0	0	3.33	20.0
8 (40%)	0	0	0	16.67	16.67

Table 5.8 Percentage of GBAD-MDL extreme runs where at least one anomalous modification is discovered.

Size of Anomaly (% of normative pattern)	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**
2 (10%)	100	100	83.33	93.33	98.33
3 (15%)	0	96.67	70.0	100	97.78
4 (20%)	0	89.17	40.0	37.50	90.83
5 (25%)	0	46.67	26.67	26.0	50.67
6 (30%)	0	16.67	70.0	23.33	58.89
7 (35%)	0	4.76	55.24	9.05	33.81
8 (40%)	0	2.5	15.0	38.75	30.42

Table 5.9 Percentage of GBAD-MDL extreme runs that return false positives.

Size of Anomaly (% of normative pattern)	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**
2 (10%)	100	3.23	45.65	3.45	3.23
3 (15%)	0	3.33	50.0	0	0
4 (20%)	0	10.0	51.61	62.5	0
5 (25%)	0	21.88	67.74	60	39.29
6 (30%)	0	59.38	26.67	63.64	25.0
7 (35%)	0	80.65	20.0	86.84	65.91
8 (40%)	0	86.92	61.76	58.06	64.82

For the small anomalies, the results are effective at all thresholds, with some degradation at the higher thresholds. However, despite some occasional positive indications of effectiveness, the results are sporadic the larger the anomaly and the higher the threshold. If we were to institute a different definition of anomaly, it is possible that our current algorithms would not be effective.

5.6 Probabilistic Results

For each of the following tests, we will implement the following GBAD settings:

- prune substructures whose value is less than their parent's (for performance)
- only analyze extensions found in the top 10 best substructures (for performance)
- allow the program to iterate (i.e., compress and run again) until there is nothing left to compress

For these experiments we will not perform any single vertex tests because one can not insert a new vertex without also inserting a new edge.

5.6.1 Insertions

The following table shows the effectiveness of the GBAD-P approach on graphs of varying sizes with random anomalous insertions.

Table 5.10 Percentage of discovery for GBAD-P runs on anomalous insertions.

Graph Size (Norm Pattern) <anomaly size>	All Anomalies	Partial Anomalies	False Positives
100 vertices/100 edges (3 vertices/3 edges)			
<1 vertex/ 1edge>	100	100	0
<2 vertices/2 edges>	100	100	0
100 vertices/100 edges (10 vertices/10 edges)			
<1 vertex/ 1edge>	100	100	0
<2 vertices/ 2 edges>	100	100	0
<3 vertices/3 edges>	100	100	0
<4 vertices/ 4 edges>	100	100	0
<5 vertices/5 edges>	93.33	93.33	0
1000 vertices/1000 edges (10 vertices/10 edges)			
<1 vertex/ 1edge>	100.0	100.0	0
<2 vertices/ 2 edges>	100.0	100.0	0
<3 vertices/3 edges>	93.33	95.56	0
1000 vertices/1000 edges (30 vertices/30 edges)			
<1 vertex/ 1edge>	93.33	93.33	0
<2 vertices/ 2 edges>	96.55	96.55	0
<3 vertices/3 edges>	93.33	93.33	0
10000 vertices/10000 edges (10 vertices/10 edges)			
<1 vertex/ 1edge>	96.67	96.67	0
<2 vertices/ 2 edges>	100	100	0
<3 vertices/3 edges>	93.33	98.33	0
10000 vertices/10000 edges (30 vertices/30 edges)			
<1 vertex/ 1edge>	100	100	0
<2 vertices/ 2 edges>	93.10	95.69	0
<3 vertices/3 edges>	90.0	95.56	0
<4 vertices/ 4 edges>	96.0	96.0	0
<5 vertices/5 edges>	96.43	99.64	0

In the example with 1000 vertices/1000 edges and a normative pattern of 10 vertices/10 edges, even though some unrealistic anomaly sizes were used (representing

20-30% of the normative pattern), this approach is still effective. This same behavior can be observed in larger graphs as well.

As a further experiment, we also tried this approach on different distributions, varying the number of vertices versus the number of edges (e.g., adding more edges than vertices by creating more edges between existing vertices), and also reducing the distribution difference between noise and anomalies. In all cases, the results were relatively the same, with never less than 96.67% of the anomalous instances being found for anomalies of size 8 (or 40% of the normative pattern) or less, with the lowest discovery rate being 90% for an anomaly of size 10 (or 50% of the normative pattern).

There are several parameters that can be adjusted in these algorithms, two of which are the *beam width* of the search and the *limit* on the number of substructures to consider in each iteration. Each of these values is an adjustable parameter so as to aid in the issue of performance primarily due to the number of calls to the graph isomorphism procedure. By default, the beam is set to a value of 4. The beam specifies at each level of the search the number of paths being considered. So, the search beam width can be adjusted when we are looking for anomalies. As a result, there is a tradeoff between time and memory. With each increase in the size of the beam, not only is there an increase in CPU time, but the amount of memory required to store all of the possibilities is prohibitive. By default, the number of substructures considered in each iteration is based upon the size of the input graph as (number of edges)/2. While many years of experimentation with SUBDUE have shown that the normative pattern (or best substructure) should be discovered with these default settings, because the

GBAD algorithms need to examine substructures beyond the normative, the limit may need to be increased in order to consider other possible substructures. The limit parameter controls the extent of SUBDUE's search by limiting the number of different substructures SUBDUE considers for expansion, i.e., it is an upper bound on the portion of the search space considered by SUBDUE. The *limit* defaults to half the number of edges in the graph. This default value tends to be enough (and higher than necessary in some cases), as SUBDUE typically finds the best substructure early on. However, again, as with the increasing of the beam width, there is a tradeoff associated with achieving 100% accuracy versus time and memory usage. To demonstrate this point, for each of the tests shown in Table 5.10 where the detection rate is less than 100%, we incrementally increased the beam width and limit until all anomalies were discovered. (It should be noted that from the original settings until we reached a 100% discovery rate, no false positives were reported.) In these tests, the larger the graphs and the larger the normative pattern, the larger the beam and limit had to be set. In order to get a 100% discovery rate on the largest graphs with the largest normative patterns (in this case, 10,000 vertices/10,000 edges, with a normative pattern of 30 vertices/30 edges), we had to increase the beam width to 20 with a limit of 200,000. The limit is based upon SUBDUE's limit on substructures considered, which to allow consideration of a pattern with V vertices, we have to set the limit equal to the (number of initial substructures) + (V * beam width), where the number of initial substructures is the number of unique vertex labels that appear more than once in the graph. The second term derives from the fact that SUBDUE considers each substructure in the beam before considering extensions to these substructures. Thus, ((V-1) * beam width) substructures will be considered before a substructure of size V is considered. In the case of this experiment, this increase also results in approximately an 11% increase in execution time, with a minimal increase in memory usage.

Also, as was mentioned earlier, this approach is incrementally discovering single extensions from compressed substructures. Since the random generation of the anomalies could introduce multiple edges between the same two vertices, compressing the normative pattern at an iteration could result in a second duplicate edge being missed or ignored, as it would just appear to be a self-edge to a compressed substructure.

5.6.2 Modifications

Running modification tests on graphs of 100 vertices/100 edges and 1000 vertices/1000 edges results in no anomalies, partial or complete, being discovered, and no false positives reported. The issue with trying to find modifications using the Probabilistic approach lies in the way the algorithm examines extensions. Modifications are changes to the normative pattern, while the Probabilistic approach is examining edges and vertices that are connected to a normative pattern. Since modified anomalies are connected to substructures that are not the best substructure (or normative pattern), examination of extensions would not consider these anomalies. While we could look for smaller patterns (i.e., a substructure of the true normative pattern that does not include the modified vertices and edges), that would go against our principle of what is the normative (and best) pattern.

5.6.3 Deletions

Running deletion tests on graphs of 100 vertices/100 edges and 1000 vertices/1000 edges results in no anomalies, partial or complete, being discovered, and no false positives reported.

Again, the Probabilistic approach is examining extensions from the normative pattern. Intuitively, it makes sense that anomalous deletions would be difficult to discover with this approach, as they could almost be considered the opposite of what we are hoping to uncover.

5.6.4 Denser Graphs

As was done when we tested the GBAD-MDL algorithm, the following results represent tests using graphs where we increased the ratio of edges to vertices so as to create a denser graph.

Table 5.11 Percentage of discovery for GBAD-P runs on denser graphs with anomalous insertions.

Size of Anomaly (% of normative)	**All Anomalies**	**Partial Anomalies**	**False Positives**
2 (5%)	96.67	96.67	0
4 (10%)	83.33	89.17	1.79
6 (15%)	93.33	98.89	0

When compared to the results shown in Table 5.10, it is clear that the density of the graph does not have much of an effect on the results. For instance, when compared to the graphs of 1000 vertices/1000 edges or larger, the results are both around the 90% discovery rate with almost 0% false positives.

5.7 Maximum Partial Substructure Results

For each of the following tests, we will implement the following GBAD settings:

- prune substructures whose value is less than their parent's (for performance)
- only analyze the top 25 ancestral substructures (for performance)
- a cost of transformation (or anomalous) threshold of 8.0, so as to ignore substructures that would have too many changes to be considered an anomaly (based upon our definition of an anomaly)

The threshold value of 8.0 was chosen based upon our definition of an anomaly and the maximum size of the normative patterns. For each of the following experiments, we will use an anomalous ratio of 10% or less to normative patterns with a maximum of 30 vertices and 30 edges. By choosing a value of 8.0, we are guaranteed to consider an anomalous instance with at most 8 changes. While we can use a threshold greater than 8.0, there is always a tradeoff between noise and actual anomalies, so with the expectation that the anomalies will always be around 10% or less of a deviation from the norm, we have chosen a value that will discover the targeted anomalies with minimal false positives. Note that "partial-anomalies" do not apply for this approach.

Either an instance is anomalous because it is missing some edges and vertices that exist in the normative pattern, or it is not considered anomalous.

5.7.1 Deletions

The following table shows the effectiveness of the GBAD-MPS approach on graphs of varying sizes with random anomalous deletions.

Table 5.12 Percentage of discovery for GBAD-MPS runs on anomalous deletions.

Graph Size (Norm Pattern) <anomaly size>	**All Anomalies**	**False Positives**
100 vertices/100 edges (3 vertices/3 edges)		
<1 vertex and associated edges>	100	0
<1 edge>	100	0
<1 vertex/1 edge>	100	0
100 vertices/100 edges (10 vertices/10 edges)		
<1 vertex and associated edges>	100	0
<1 edge>	100	0
<1 vertex/1 edge>	100	0
<2 vertices/1 edge>	100	0
1000 vertices/1000 edges (10 vertices/10 edges)		
<1 vertex>	100	0
<1 edge>	100	0
<1 vertex/1 edge>	100	0
<2 vertices/1 edge>	100	0
1000 vertices/1000 edges (30 vertices/30 edges)		
<1 vertex>	100	0
<1 edge>	100	0
<1 vertex/1 edge>	100	0
<2 vertices/1 edge>	100	0
<2 vertices/2 edges>	100	0
<2 vertices/3 edges>	100	0
<3 vertices/3 edges>	100	0
10000 vertices/10000 edges (10 vertices/10 edges)		
<1 vertex>	100	0
<1 edge>	100	0
<1 vertex/1 edge>	100	0
<2 vertices/1 edge>	100	0
10000 vertices/10000 edges (30 vertices/30 edges)		
<1 vertex>	100	0
<1 edge>	100	0
<1 vertex/1 edge>	100	0
<2 vertices/1 edge>	100	0
<2 vertices/2 edges>	100	0
<2 vertices/3 edges>	100	0
<3 vertices/3 edges>	100	0

Initially, the results from the runs on the graph with 1000 vertices and 1000 edges, where the normative pattern consists of 30 vertices and 30 edges resulted in very low detection rates. However, when we increase the number of ancestral substructures (to analyze) to 100, and increase the anomalous threshold (i.e., cost of transformation * frequency) to 50.0, the results improve to what is shown. The reason that the number of best substructures and the threshold has to be increased is that as the size of the anomaly grows (i.e., the number of vertices and edges deleted increases), the further away the cost of transformation for the anomalous instance is from the normative pattern. So, a good rule of thumb is to choose an anomalous threshold based upon the size of the normative pattern. For instance, GBAD could be run first to determine the normative pattern, then based upon the size of the normative pattern, we can determine the maximum size of an anomaly (e.g., around 10%), choose a cost of transformation that would allow for the discovery of an anomaly that size, and then rerun the algorithm with the new threshold.

5.7.2 Modifications

Running tests on graphs of 100 vertices/100 edges and 1000 vertices/1000 edges results in no anomalies, partial or complete, being discovered, and a 100% false positive rate.

Examination of the output reveals that while it does not find any of the true anomalies, it appears that every false positive is actually "around" the anomaly. For instance, if the anomaly is vertex **X** (of vertices **X**, **Y** and **Z**), it might display **Y** as the anomaly. Or, if the anomaly is vertex **X** and the edge between **X** and **Y**, it might display **Z** and the edge between **Y** and **Z** as the anomaly. In other words, the reported anomaly is adjacent to the actual anomaly.

For example, in the case where the graph contains (before the anomalous modification is made) the normative pattern shown in Figure 5.10.

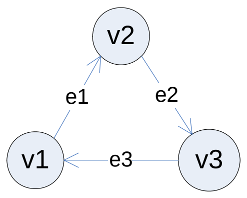

Figure 5.10 Example normative pattern.

When we change the label on the vertex labeled v3 to the new label of v1, and the label on the edge labeled e3 to the new label of e4, the GBAD-MPS algorithm reports the instance shown in Figure 5.11 as anomalous.

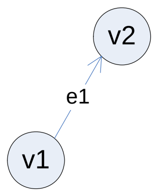

Figure 5.11 Anomalous instance from example.

As one can see, it does not report the actual anomalous vertex and/or edge, but it does report the rest of the vertices and edge (except for the other edge that was connected to the anomalous vertex) that were part of the instance before it was modified. While not ideal, a user might find that useful in that they could then examine vertices and edges surrounding this reported anomaly.

It should also be noted that while the output of false positives can be reduced by decreasing either the anomalous threshold or the number of substructures to be considered, it does not result in any anomalous modifications being discovered.

5.7.3 Insertions

When running tests on 100 vertices/100 edges and 1000 vertices/1000 edges the GBAD-MPS algorithm does not discover any anomalies. As with the previous tests, since no anomalies are discovered in any of the tests up to this point (also there are no false positives), there is no reason to continue with these experiments. If the GBAD-MPS algorithm is unable to find anomalous insertions on small graphs where the normative pattern is known, it will be unable to find insertions on much larger graphs.

5.7.4 Denser and Larger Graphs

Identical to the previous two approaches, we increased the ratio of edges to vertices so as to create a denser graph.

Table 5.13 Percentage of discovery for GBAD-MPS runs on denser graphs with anomalous deletions.

Size of Anomaly	**All Anomalies**	**False Positives**
1 (edge)	100	0
1 (vertex AND associated edges)	100	0
2 (vertex and edge)	100	0
3	100	0
4	100	0
5	100	0

Again, the results from the denser graphs mirror those of the previous runs, as all of the anomalies are discovered with no false positives reported.

We also experimented on larger graphs, on the order of 20,000 vertices and 20,000 edges, with anomalies of varying sizes. While the running times and amount of memory required increased significantly, the results are the same. The GBAD-MPS algorithm is able to discover all of the anomalies with no false positives.

5.8 Handling Multiple Types

In the real world, it is more than likely that the type of anomaly will be unknown and/or the anomalous situation will consist of more than one type of anomaly. In the case where the type of anomaly is unknown, that will require us to run all three algorithms on the data and observe which algorithms reports anomalies. In actual fraud detection, companies employ many different types of algorithms and tools, each with a purpose of finding different types of fraud. For example, telecommunications companies use algorithms such as call volumes over a specified time period, calls from particular hot numbers, or increased volume from a single calling card as part of a suite of tools for detecting fraud. Each approach is focused on a specific type of anomaly, and will raise an alarm to an analyst if a flag is set or a threshold is exceeded. Similar to our approach, each algorithm is focused on detecting a specific type of anomaly, and together they can discover all possible types of graph-based structural anomalies.

However, there is also the scenario where the anomalous situation consists of multiple anomalous types. In other words, an anomalous substructure could be composed of a modification, insertion and deletion. An example of this is a fraudulent financial transaction where the perpetrator has generated an unexpected transaction type (modification), routed the money through another financial institution (insertion), and omitted some identification information (deletion). Another actual example of multiple anomalous types will be shown in the next chapter.

In order to determine the effectiveness of our algorithms in this scenario, we have to modify the approach we originally used for setting the normative pattern. However, we fortunately do not have to make any modifications to the algorithms themselves. Each algorithm can still be focused on discovering a particular type of anomaly, but now they must base their analysis on the pattern discovered by one of the other algorithms. To understand how this works, take the situation where the anomalous substructure that we wish to discover contains all three types of anomalies – modifications, insertions and deletions – as shown in Figure 5.12, where the modified label is an **X**, the inserted vertex (labeled Y) and edge are in bold, and the deleted vertex (labeled Z) and edge (which would be missing from the anomalous instance) are separated from the rest of the substructure.

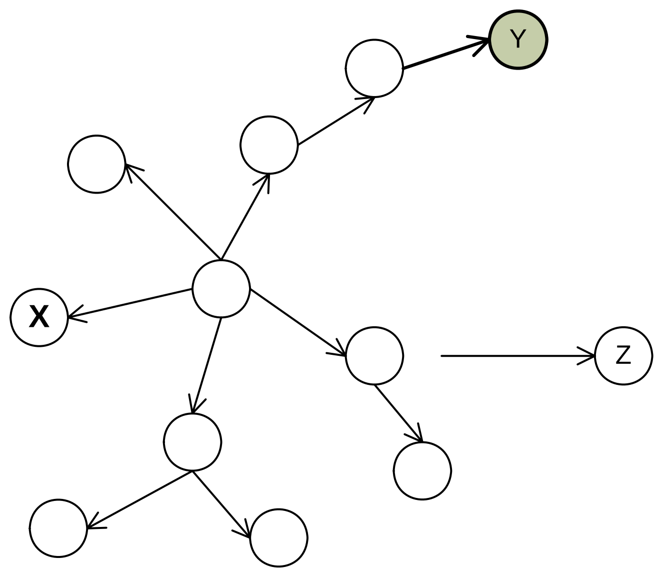

Figure 5.12 Substructure containing a modification, insertion and deletion.

In this example, suppose the normative pattern consists of all of the vertices and edges shown in this substructure excluding the inserted edge and vertex (shown darkened). When we run all three GBAD algorithms on the graph containing this normative pattern and this anomalous substructure, two steps are involved. On the first iteration, we run the GBAD-MDL and GBAD-MPS algorithms to determine the normative patterns and the base anomalous substructure. Using an inexact matching threshold of 0.15 (to allow for the modified vertex and the deleted vertex and edge), the GBAD-MDL and GBAD-MPS algorithms produce the anomalous substructure shown in Figure 5.13, with the anomaly (so far) labeled.

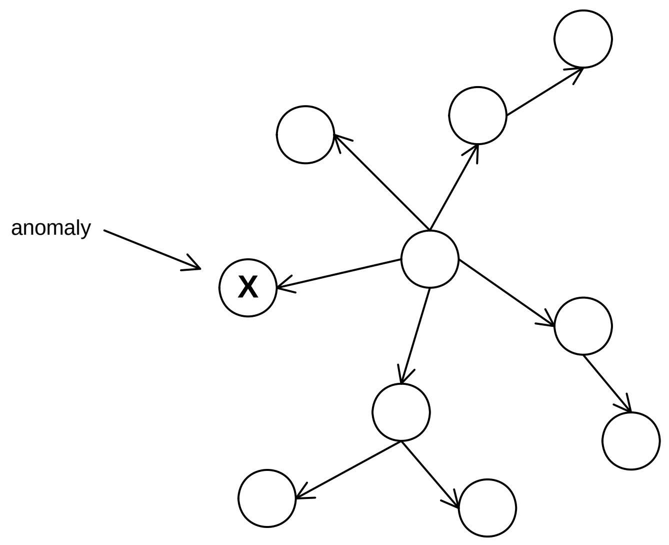

Figure 5.13 Anomalous substructure after one iteration.

On the second and subsequent iterations, the GBAD-P algorithm uses the anomalous substructure as its base substructure for evaluating extensions. The difference in this implementation of the algorithm is that it is initially using the anomalous substructure as the normative pattern, instead of the true normative substructure that is used when GBAD-P is used by itself, where the list of instances of the true normative substructure is maintained so that extensions can be evaluated to find which of the extensions of the anomalous substructure have the lowest probability. The result (in this example, after just one more iteration), is the anomalous substructure in Figure 5.14 with each of the individual anomalies labeled.

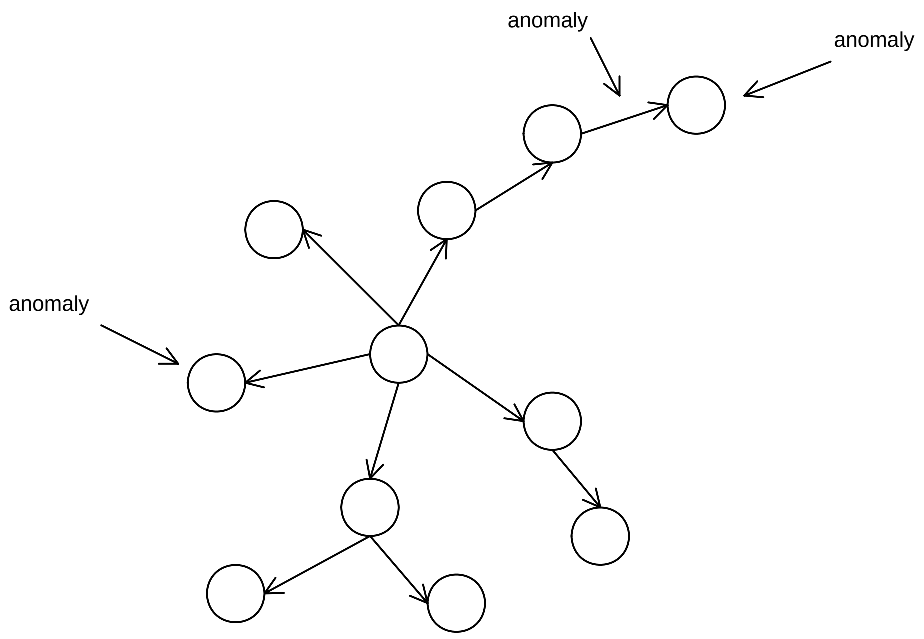

Figure 5.14 Anomalous substructure after two iterations using all GBAD algorithms.

And, of course, when compared to the normative pattern, the missing vertex and edge will be seen as the anomalous deletion.

In order to thoroughly test our approach, we will generate random graphs of 100 vertices and 100 edges, with a normative pattern of 10 vertices and 9 edges, with tests that cover each of the following possible random changes:

1. Modification and insertion
2. Modification and deletion
3. Insertion and deletion
4. Modification, insertion and deletion

Using the default parameters, our results are less than ideal, with around 90% effectiveness for the first change scenario and a less than 30% discovery rate for the remaining possibilities. However, similar to what we discovered during our previously reported synthetic experiments, the GBAD-P and GBAD-MPS algorithms may require a much larger beam width and limit in order to evaluate a larger set of substructures. In addition, due to the nature of the size of the normative pattern in relation to the amount of change, the inexact matching threshold will need to be increased to compensate for the scenarios when the anomaly involves modifications and deletions – both changes that require a larger match cost.

For tests consisting of five or fewer changes for each type (e.g., one modified vertex label, one deleted vertex and one deleted edge), when we increase the beam width to 100 and the limit to 11,000 (again we just kept increasing the beam width until an anomaly was reported, and subsequently based our limit on the size of the beam, the number of pattern vertices and the number of initial single-vertex substructures), and set the inexact matching threshold to 0.27 (based upon the amount of deviation we are willing to accept), we are able to discover 100% of the anomalies in each of the above scenarios. We also notice, as we did in other tests, that as the amount of change increases in proportion to the size of the substructure, the ability to discover the anomalies decreases. The issue again lies in the ability to distinguish expected deviations (e.g., noise) from the unexpected deviations (i.e., anomalies). If the expected change consists of a smaller deviation than the unexpected anomaly, the algorithms will report the non-anomalous deviation as the anomaly.

5.9 Other Types of Normative Patterns

Each of the above tests is run using the same type of normative pattern: a star cluster. However, it is possible that the shape of the normative pattern could have an effect on the effectiveness of the algorithms. So, the following tests are devised in order to test other patterns that are common, particularly as they might be used for real-world data. (Refer to Section 5.4 for examples of each of these patterns.)

Each of the following tests are executed with a graph size of approximately 500 vertices and 500 edges, a normative pattern of 10 vertices and 10 edges, and an anomalous change of 10% . All of the same GBAD parameter settings that were used in the previous tests are implemented here, and a threshold of 0.1 is used for the GBAD-MDL algorithm.

Table 5.14 Percentage of complete anomalous instances found on runs with different patterns.

Normative Pattern / **Algorithm (Anomaly Type)**	**Star**	**Strand**	**Cycle**
GBAD-MDL (Modification)	100	100	100
GBAD-P (Insertion)	40.0	100	100
GBAD-MPS (Deletion)	100	100	100

Table 5.15 Percentage of runs with anomalous parts found on runs with different patterns.

Normative Pattern / Algorithm (Anomaly Type)	Star	Strand	Cycle
GBAD-MDL (Modification)	100	100	100
GBAD-P (Insertion)	40.0	100	100
GBAD-MPS (Deletion)	n/a	n/a	n/a

Table 5.16 Percentage of runs containing false positives on runs with different patterns.

Normative Pattern / Algorithm (Anomaly Type)	Star	Strand	Cycle
GBAD-MDL (Modification)	0	0	0
GBAD-P (Insertion)	44.19	0	0
GBAD-MPS (Deletion)	0	0	0

From these results we notice that except for the star normative pattern when running the GBAD-P algorithm, the results are the same across each algorithm for each of the different shapes. However, if we completely distinguish the anomalies from the noise (i.e., use different labels for the anomalies as opposed to the random connections), 93.3% of the complete anomalous instances and 93.3% of the partial anomalies are discovered – closer to the results found when the normative patterns are the other pattern types. Because of the connectivity introduced when generating a connected graph of star patterns, the ability to distinguish between noise and actual anomalies becomes more difficult.

It should also be noted that the placement of the anomaly plays into the less than perfect discovery rate for GBAD-P algorithm when the normative pattern is a star. In all cases where the anomalous edge is attached to a *non-center* vertex, the anomaly is discovered, as in the example shown on the left in Figure 5.15.

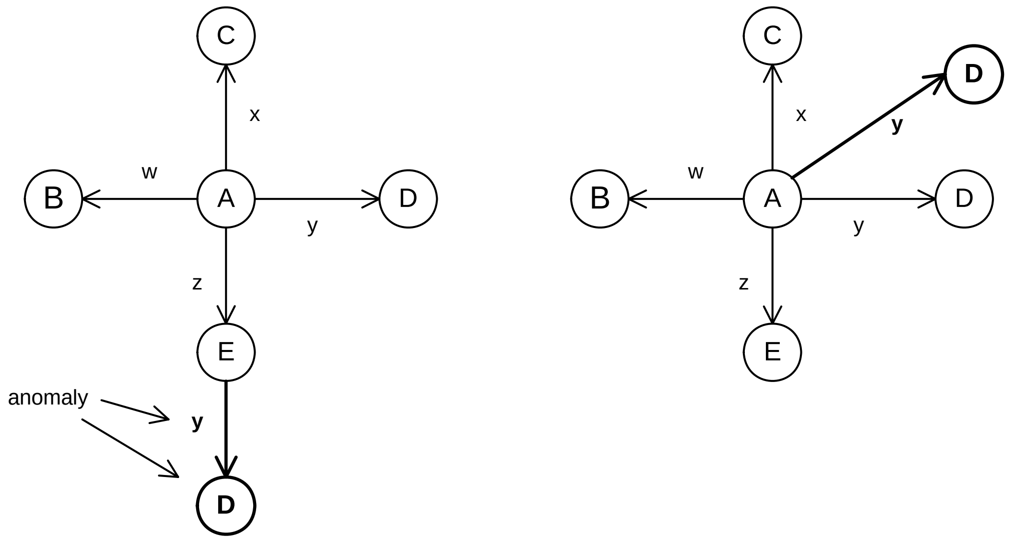

Figure 5.15 Examples of anomalous insertions to a star shaped pattern.

However, when the anomalous insertion is like the one shown on the right in Figure 5.15, the anomaly is not always discovered by the GBAD-P algorithm. This is really more of an issue with how the normative pattern is discovered. Not that the discovery is incorrect, as the best pattern is discovered, but that it chose the anomalous edge (and vertex) rather than the non-anomalous one. So, again taking the example shown in Figure 5.15, the edge and vertex shown in bold in the substructure on the right, are used in the building of that instance, and the other identical edge and vertex (not in bold) are reported as the anomaly. Of course, one of the advantages of these algorithms is that not just the anomaly is reported but the entire anomalous instance. In reality, as shown in this example, what has been reported is correct – there is an anomalous edge labeled *y* and an anomalous vertex labeled *D*.

5.10 Limitations

As was mentioned and demonstrated earlier with the GBAD-P experiments, the ability to discover the anomalies is sometimes limited by the *resources* allocated to the algorithm. This is true not only for GBAD-P, but for GBAD-MDL and GBAD-MPS as well. In other words, given a graph where the anomalous substructure consists of the *minimal* deviation from the normative pattern, if a sufficient amount of processing time and memory is provided, all of these algorithms will discover the anomalous substructure with no false positives.

However, the ability to discover anomalies (per our definition) is also hampered by the amount of *noise* present in the graph. The issue is that if noise is a smaller deviation from the normative pattern than the actual anomaly, it may score higher than the targeted anomaly (depending upon the frequency of the noise). Of course, one might say that noise is an anomaly in that it is not normal; however, it is probably not fraudulent activity, which is the goal of these approaches.

Now, the presence of noise does not eliminate the algorithms' abilities to discover the anomalous substructure. It only results in more false positives being detected if the anomalous score of the noisy structure is better than the desired anomalous substructure. That is where another trade-off is necessary that can be found in most fraud detection systems: adjusting thresholds to find a balance of false-positives versus true anomalies. This is something to be considered in future work where ideally the approach would not involve parametric values that may need to be adjusted based upon a user's experiences.

5.11 Overlapping Instances

Another possible scenario to consider is the case of overlapping instances. This situation occurs when instances of the normative pattern overlap with other instances of the normative pattern. For example, suppose you have the situation of two star instances that share a common vertex, as shown in Figure 5.16.

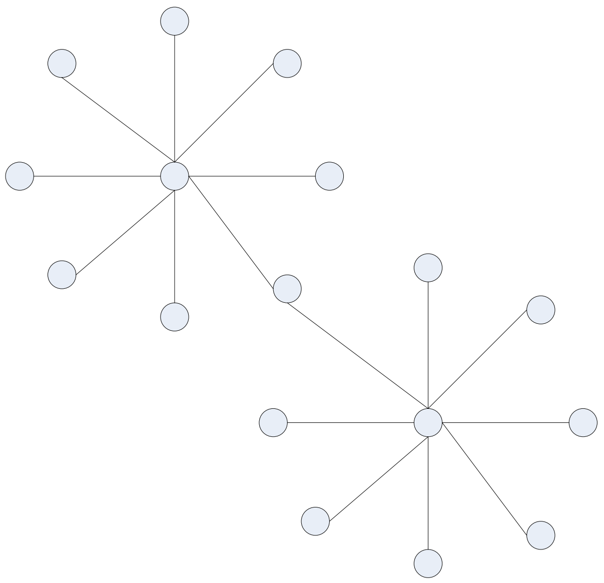

Figure 5.16 Two star patterns sharing a common vertex.

Taking this example, if the normative pattern consists of a vertex connected to eight other vertices, by default, only one of the star instances will be counted as an instance of the normative pattern. However, when we allow for instances to overlap, thus counting (in this example) each star as a separate complete instance, both instances get full consideration when analyzing the substructure for anomalies (as well as discovering the normative pattern). As was mentioned earlier, we can override the default setting of not allowing overlapping instances.

First, just to observe what happens when we do not allow overlap discovery mode on graphs with overlapping instances, as the amount of overlap increases, the ability to discover the anomalies significantly decreases. This is to be expected, as we are relying on the anomalous instance to be among the discovered instances, and not to be one of the structures discarded when it is found to be overlapping with another previously discovered structure. In other words, if a portion of the anomalous instance (e.g., a vertex and edge) is already included in another instance being considered, the remaining parts of the anomalous instance will not be analyzed any further because it is not an instance of the normative substructure (or within an acceptable threshold of change to the normative substructure). Thus, the more instances overlap, the greater the probability that the anomaly will be lost.

For each of our algorithms, the key to discovering the anomalous substructure lies initially with the ability to determine the normative pattern in a graph. Since our definition of what is an anomaly is based upon small deviations from the norm, if we are unable to find the normative pattern, our ability to uncover anomalies with these approaches will be coincidental.

As was discussed in Chapter 4, GBAD's heuristic for evaluating substructures is based upon the Minimum Description Length (MDL) principle, where the description length is an approximation of the minimum number of bits needed to encode a graph. As such, the MDL of a graph consists of the description length of the graph *compressed* by the best substructure, plus the description length of that substructure. Thus, in our GBAD implementation, substructures are evaluated, and the one that results in the lowest description length is determined to be the best substructure – or the normative pattern. Computationally, we replace all instances of the best substructure with a single vertex, representing the best substructure, and calculate the new size of the graph. Figure 5.17 shows a pictorial representation of this replacement for evaluation. The graph on the left consists of three instances of the normative substructure a->b->c. Compressing the graph by replacing each of the three instances with a single vertex, results in a smaller graph that can be described by only 3 vertices, as shown on the right.

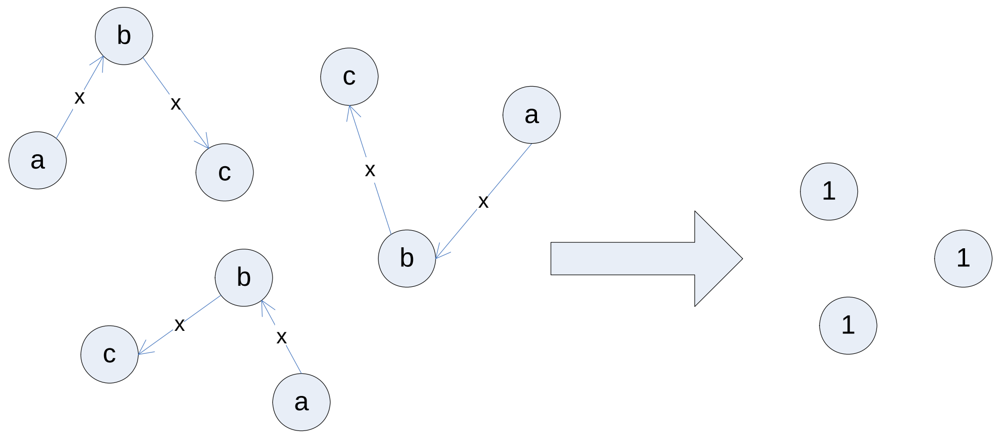

Figure 5.17 Example of compressing a graph by its normative pattern.

As mentioned earlier, when a graph consists of overlapping instances, there is the possibility that not all instances of the normative pattern will be discovered. However, there are several ways to handle the evaluation of substructures that consist of overlapping substructure instances. In addition to the normal way of evaluating substructures based upon their compression (as shown in Figure 5.17), another way is to count the number of substructure instances that match a specified substructure definition. In this case, we would consider substructures with the most matching instances to be the best substructure. Take the example in Figure 5.18.

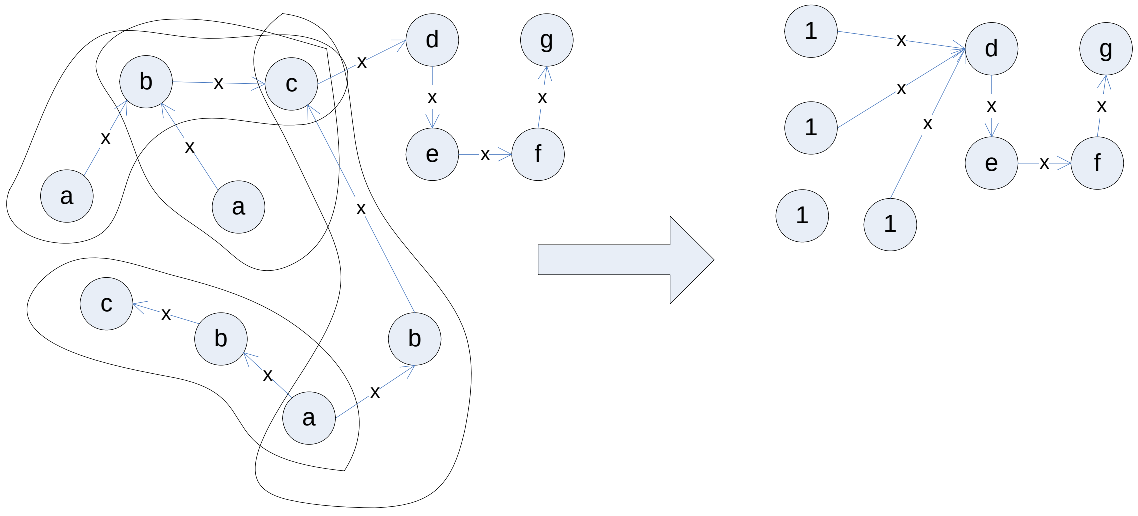

Figure 5.18 Example of compression with overlapping instances based on the number of instances.

In this example, using an evaluation based upon the number of instances, the normative pattern consists of several overlapping a->b->c substructure instances. The result after compression is a graph consisting of eight vertices and six edges, with four of the vertices not being compressed.

Another option, and the one we have chosen, is to base our evaluation on the *coverage* that is achieved during compression. Taking the same example from Figure 5.18, if we base our evaluation of the best substructure on the pattern that includes the most vertices and edges, we achieve the result shown in Figure 5.19.

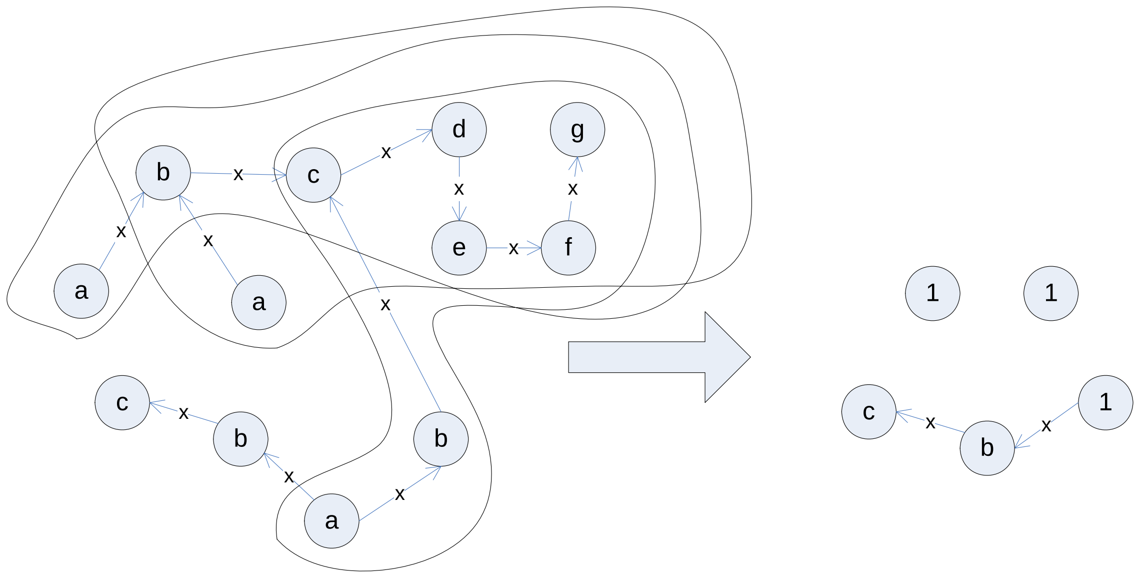

Figure 5.19 Example of compression with overlapping instances based on the amount of coverage.

In this example, the result is not only a smaller representation of the graph, represented by five vertices and two edges, but also one that covers more of the original graph, as only two original vertices are not covered by this choice of normative pattern.

Clearly, the latter approach includes more of the graph's vertices and edges than the approach that uses the number of instances. However, the reason that we chose this approach over the maximum compression approach is due to the somewhat deceiving description length of the graph when substructure instances are significantly overlapping. Because every instance of the supposedly best substructure is replaced by a single vertex, substructures consisting of overlapping instances are penalized by the fact that the compressed graph is represented by a vertex for each of the overlapping instances. Thus, a graph whose normative substructure instances overlap with multiple vertices and edges, may report a different normative substructure that has resulted in fewer compressed vertices because a compressed vertex is inserted for *each* instance, no matter how many are overlapping. So, while a substructure using the compression evaluation may represent a graph with the fewest number of vertices, with the coverage approach, we can ensure the maximum number of vertices and edges are represented by the instances of what we have determined to be the best substructure.

In order to test our algorithms on overlapping instances, we have to increase the *beam width* of the search and the *limit* on the number of substructures to consider. In earlier experiments we increased the beam and limit so that we could evaluate larger graphs. For graphs with overlapping instances, we have to increase these parameters due to the increased number of possible substructures.

Figure 5.20 shows the results when running the GBAD-MDL algorithm (with a threshold of 0.19 to allow for up to 2 changes on a substructure of size 11) on a graph of 100 vertices and 100 edges, for a normative pattern of 6 vertices and 5 edges, with varying amounts of overlap between the instances, and anomalies consisting of random modifications to a vertex label and an edge label.

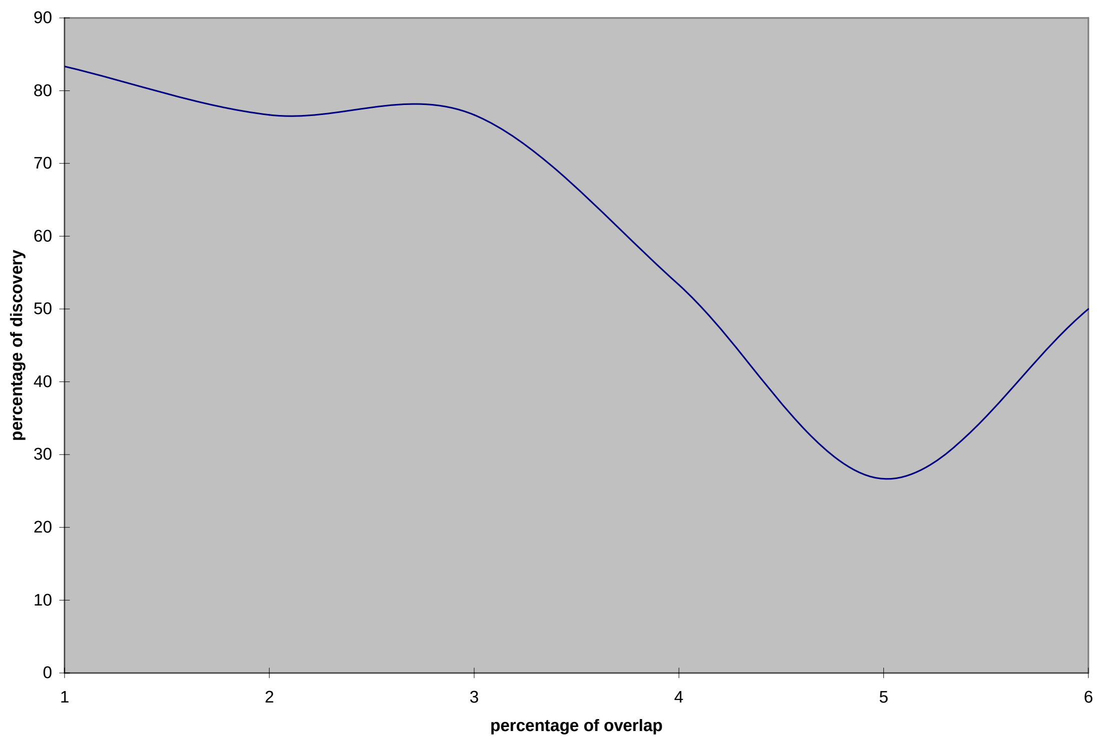

Figure 5.20 Discovery percentages for overlapping instances.

Similar results are obtained with the GBAD-P and GBAD-MPS algorithms.

In every test case, the correct normative pattern is properly discovered. However, due to the overlapping nature of the normative instances, in many cases, the anomalous substructure is not discovered when the random anomalous modification is larger than other random noise. Because we have to use an inexact matching threshold of 0.19 in order to find an anomaly of size 2 for a substructure of size 11, other substructure instances, as a result of overlapping, are discovered that have lower cost transformations because they consist of just a single vertex or just a single edge modification. But, in experiments where the anomaly consists of either a single vertex or a single edge, the anomalous substructure is discovered 100% of the time, with small levels of false positives that increase as the amount of overlap increases.

While it appears that our algorithms may have issues with graphs that consist of overlapping instances, these experiments further accentuate our definition of an anomaly. In every case, when the anomaly was the smallest deviation to the normative pattern, our approaches never failed in discovering the complete anomalous substructure. These results are consistent with previous results, and should be addressed as part of any future work for distinguishing expected deviations (or noise) from the true anomalies.

5.12 Performance

One of the factors to consider in evaluating these algorithms is their respective performances. The following table represents the average running times (in seconds) for each of the algorithms against each of the graph sizes for the anomaly types that were the most effectively discovered (i.e., the types of anomalies that each algorithm was targeted to discover). Overall, the running times were slightly shorter for the non-targeted anomalies, as the algorithms for the most part did not have any anomalies to process.

Table 5.17 Running-times of algorithms (in seconds).

Graph Size (Normative Pattern Size) / Algorithm (Anomaly Type)	100v 100e (6)	100v 100e (20)	1,000v 1,000e (20)	1,000v 1,000e (60)	10,000v 10,000e (20)	10,000v 10,000e (60)
GBAD-MDL (Modification)	0.05-0.08	0.26-15.80	20.25-55.20	31.02-5770.58	1342.58-15109.58	1647.89-45727.09
GBAD-P (Insertion)	1.33	0.95	30.61	18.52	745.45	2118.99
GBAD-MPS (Deletion)	0.14	0.07	4.97	75.59	242.65	813.46

Because the GBAD-MDL algorithm uses a matching threshold, the performance of the algorithm is dependent upon the threshold chosen. The higher the threshold, the longer the algorithm takes to execute, so there is a trade-off associated with the threshold choice. Even on graphs of 10,000 vertices and 10,000 edges, the running times varied anywhere from 1342 seconds to 45,727 seconds, depending upon the threshold chosen. In short, the GBAD-MDL algorithm is exponential in the worst case, but tractable given the parameters.

5.13 Summary

The ability to discover anomalous modifications, insertions and deletions is evident when applying all three algorithms. With a 100% or almost 100% discovery rate for each algorithm on most of the graphs with varying sizes of normative patterns and anomalies, each approach has clearly shown to be useful at discovering a specific type of anomaly. While the algorithms do not appear to be useful outside of their intended targets, no graphs of any size and any anomaly went undetected by all three approaches. Even graphs of higher density had no effect on the accuracy of the algorithms. While there were some differences in detection rates for the probabilistic approach when the normative pattern was a star, overall, the shapes of the normative pattern did not have any effect on the abilities of the algorithms to discover the anomalies. In every test, when the normative pattern was discovered, and the anomaly was the smallest deviation of an instance of the normative substructure, the instance was reported as anomalous. The running times of GBAD-P and GBAD-MPS are very acceptable, and the performance of GBAD-MDL is clearly affected by the size of the threshold specified.

CHAPTER 6

EMPIRICAL EVALUATIONS OF REAL-WORLD SCENARIOS

The following sections contain the results when applying the algorithms to real-world data sets. First, we present results using shipping manifests of cargo imports into the U.S., and compare our algorithms to other non-graph-based approaches. Then, we compare our algorithms against another graph-based approach using the KDD Cup ’99 network intrusion data. Finally, we conclude with some experiments using other real-world data sets.

6.1 Cargo Shipments

One area that has garnered much attention recently is the analysis and search of imports into the United States. The largest number of imports into the U.S. arrive via ships at ports of entry along the coast-lines. Thousands of suspicious cargo, whether they be illegal or dangerous, are examined by port authorities every day. Due to the volume, strategic decisions must be made as to which cargo should be inspected, and which cargo will pass customs without incident. This is a daunting task that requires advanced analytical capabilities that will maximize effectiveness and minimize false searches.

The Customs and Border Protection (CBP) agency maintains shipping manifests in a system called PIERS (Port Import Export Reporting Service). This database is used for tasks such as reporting and data mining. Each entry (or row) in the PIERS tables consists of various information from a shipping manifest. An example of a PIERS record (and its header) are as follows:

VDATE,BOL_NBR,CONTAINER,FPORT,USPORT,COUNTRY,SLINE,VESSEL,VOYAGE,NAME,FNAME,COMMODITY,HARM_DESC,HSCODE,HAZMAT_FLA,CONSIZE,TEUS,MTONS,VALUE

020601,00434100,TOLU4972933,YOKOHAMA,SEATTLE,JAPAN,CSCO,LING YUN HE,36,AMERICAN TRI NET EXPRESS,TRI NET,EMPTY RACK,CONTAINERS FOR ONE OR MORE MODES OF TRANSPORT,860900,,,0.00,5.60,27579

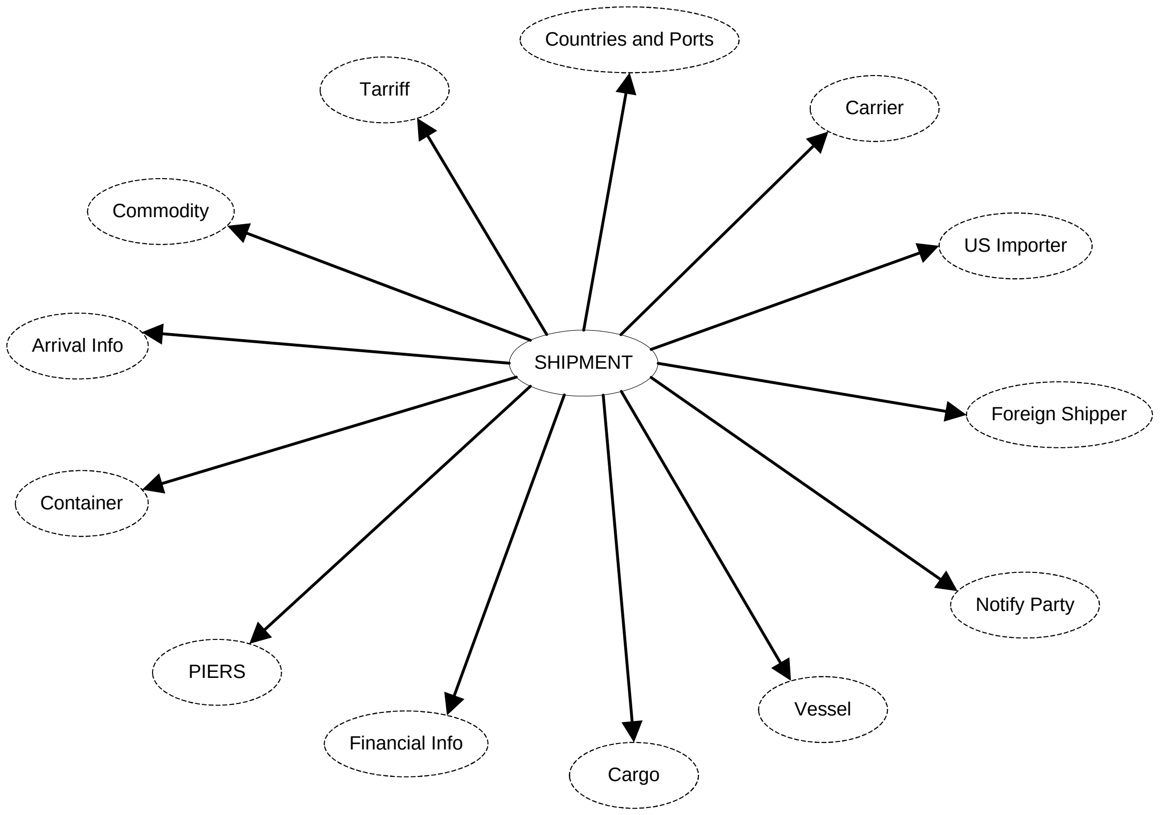

Figure 6.1 High-level pictorial representation of cargo information.

Using shipping data obtained from the CBP (http://www.cbp.gov/), we are able to create a graph-based representation of the cargo information where row/column entries are represented as vertices, and labeled edges convey their relationships as edges. Figure 6.1 shows the high-level structure of the ontology we use to represent the cargo data as a graph. Figure 6.2 shows a portion of the actual graph that we will use in our anomaly detection experiments.

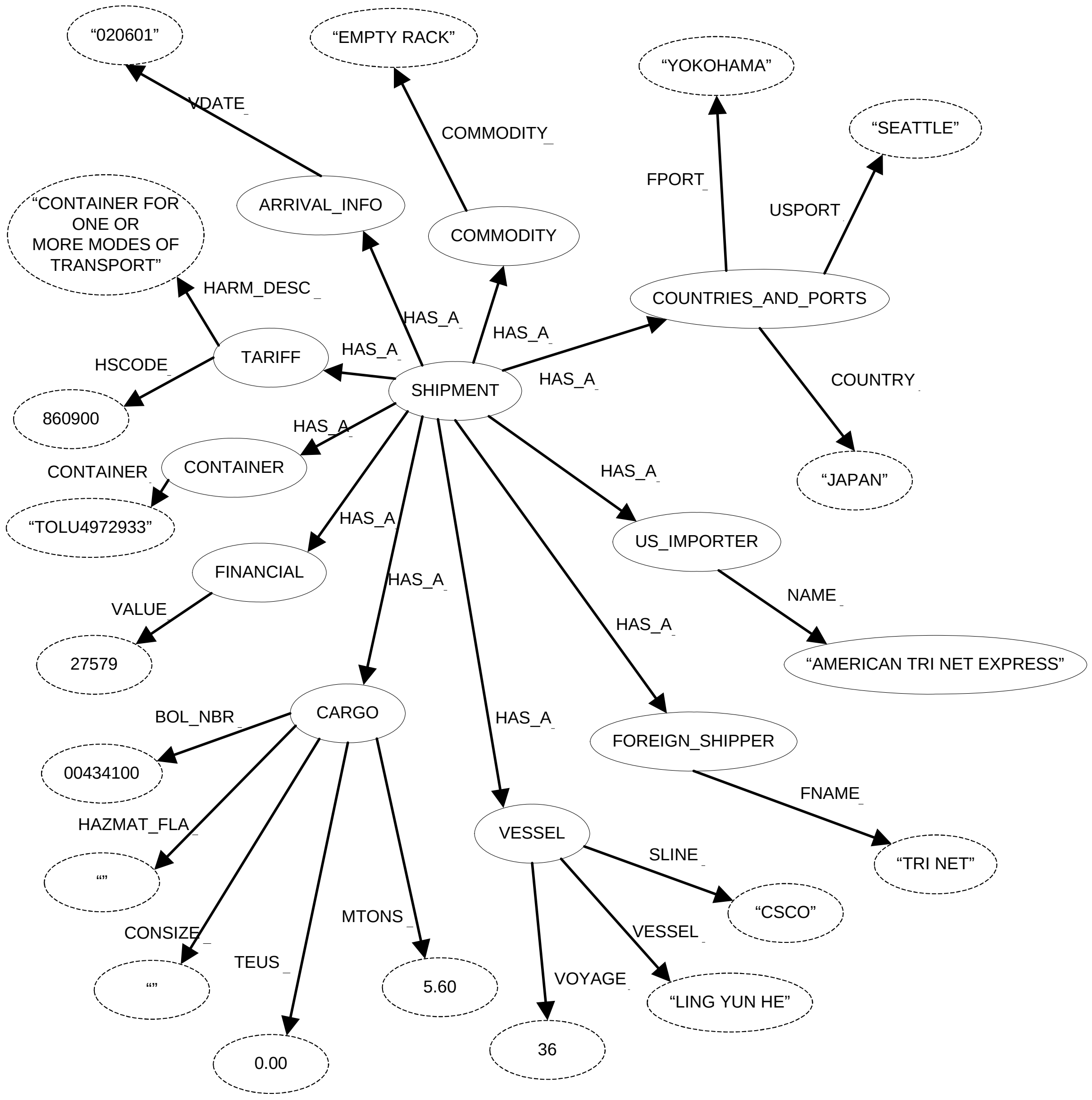

Figure 6.2 Example of cargo information represented as a graph.

While we were not given any labeled data from the CBP (i.e., which shipments were illegal, or anomalous, and which ones were not), we can draw some results from simulations of publicized incidents.

6.1.1 Random Changes

Similar to what we did for the synthetic tests, we randomly modified, inserted and deleted small portions of the graph for randomly selected shipping entries. However, even though this data is not as regular as the synthetic data generated for the earlier tests, all three algorithms were able to successfully find all of their intended target anomalies with no false positives reported. This helps us validate further the usefulness of this approach when the anomaly consists of small modifications to the normative pattern.

It should also be noted that when the algorithms are run on the original cargo data with no injected anomalies, the GBAD-P and GBAD-MPS algorithms do not find any anomalies. However, the GBAD-MDL algorithm does report anomalies when the threshold is greater than 0.08 (or 8%). Albeit, while the reported anomalies are merely different US_IMPORTER and FOREIGN_SHIPPER names (as opposed to possible fraud situations), this algorithm does successfully report anomalies in the data.

6.1.2 Real-world Scenarios

In [Eberle and Holder 2006], real-world cargo shipment occurrences were generated so as to show how graph properties can be used to determine structural anomalies in graphs. While that approach was successful in discovering graphs that contained anomalies, the exact anomalies were not part of the output. Using the GBAD algorithms on these same data sets, we can display the actual anomalies.

One example is from a press release issued by the U.S. Customs Service. The situation is that almost a ton of marijuana is seized at a port in Florida [U.S. Customers Service 2000]. In this drug smuggling scenario, the perpetrators attempt to smuggle contraband into the U.S. without disclosing some financial information about the shipment. In addition, an extra port is traversed by the vessel during the voyage. For the most part, the cargo looks like a normal shipment from Jamaica. Figure 6.3 shows a graphical representation of a portion of the graph (for space reasons) containing the anomaly.

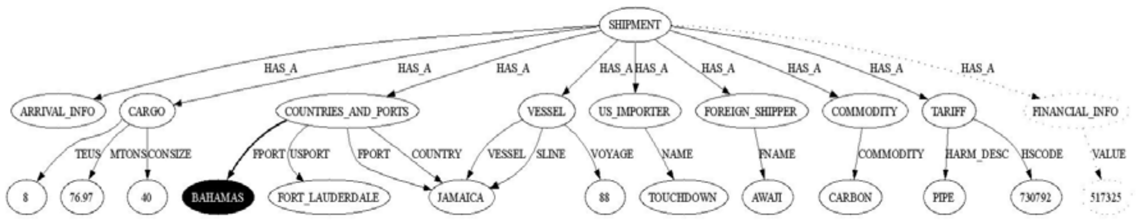

Figure 6.3 Graph representation of cargo shipment containing the anomaly, with an insertion in bold and removals represented as dotted lines.

When we run the three algorithms individually on this graph, GBAD-MDL is unable to find any anomalies, which makes sense considering none of the anomalies are modifications. When the graph contains the anomalous insertion of the extra traversed port (shown as the bold edge and darkened vertex in Figure 6.3), the GBAD-P algorithm is able to successfully discover the anomaly. Similarly, when the shipment instance in the graph is missing some financial information (the dotted and dashed edges and vertices in Figure 6.3), GBAD-MPS reports the instance as anomalous. When we run all three algorithms on the graph simultaneously, as was shown in Section 5.8, both anomalies are reported in the same run. This again further validates the effectiveness of running these algorithms individually or combined.

According to the CBP, an estimated $2 billion in illegal textiles enter the U.S. every year [Customs and Border Protection 2003]. One of the more common methods of eluding authorities is accomplished using what is called transshipment. The CBP defines transshipment as "A false declaration or information given in order to circumvent existing trade laws for the purpose of avoiding quotas, embargoes or prohibitions, or to obtain preferential duty treatment." In order to circumvent quotas, the fraudster will change the country of origin of their goods. For example, they may ship the goods into Canada or Mexico, change the country-of-origin, and ship into the U.S. free from tariffs under the North American Free Trade Agreement (NAFTA).

In order to simulate this real-world example, we randomly changed the country of origin on one of the shipments to "CANADA". While the GBAD-P and GBAD-MPS algorithms were unsuccessful in discovering this anomaly (as was expected), the GBAD-MDL algorithm was able to clearly mark the instance that contained the anomaly. At first it was surprising that just a change in the country of origin would have that effect, and given perhaps a different set of data, this would not have been as effective. But, in this case, all of the shipments had a normative pattern that included Asian shippers. So, by altering the originating country to Canada, the GBAD-MDL was able to clearly notice the structural anomaly.

6.1.3 Comparison to Non-Graph-Based Approaches

There are many different non-graph-based machine learning approaches to anomaly detection. The *Bayesian* classification approach is a probabilistic method for determining an appropriate classification hypothesis. More succinctly, it is the probability of a particular class for a given instance. This approach has been commonly used in intrusion detection, where instances are classified as possible anomalies [Kruegel et al. 2003]. *Neural Networks* are a non-linear approach that has shown to be a very effective at classifying real-valued or discrete-valued targets [Mitchell 1997]. Widely used as an approach to detect misuse, particularly for intrusion detection systems and fraud detection, it has also shown to be applicable to anomaly detection. *Support vector machines*, or SVMs, sometimes referred to as kernel-based learning algorithms, are a set of supervised learning methods that apply linear classification techniques to non-linear classification problems [Muller et al. 2001]. This approach has also been shown useful for detecting anomalies [Hu et al. 2002]. Another set of classifiers is known as *lazy learners*, or *instance-based* learning. This is perhaps the simplest form of learning as the instances themselves represent the knowledge. Training instances are searched for the instance that most closely resembles the new instance. Anomaly detection using this technique has been applied to computer security and issues on network routers [Aha et al. 1991]. Another class of learning methods is *meta-learners*. Meta-learning is an area of predictive data mining that combines predictions from multiple models. Again, meta-learners have been widely used in terms of intrusion detection systems [Lee et al. 1999]. *Decision tree induction* is one of the more popular approaches to machine learning classification. A decision tree is a tree-structured plan of a set of attributes to test in order to predict the output. In [Stein et al. 2004] a genetic algorithm is used with a decision tree classifier towards intrusion detection. Classification *rules* are a popular alternative to decision trees. The rules consist of pre-condition tests that are then logically ANDed together to produce a rule that assigns a class, or set of classes, and possibly a probability distribution. Many of the commercial intrusion detection systems in the nineties consisted of classification rules.

Perhaps the most popular approach to anomaly detection involves the class of approaches known as *clustering*. The idea behind clustering is the grouping of similar objects, or data. This is an unsupervised approach whose goal is to find all objects that are similar where the class of the example is unknown [Frank and Witten 2005]. From an anomaly detection perspective, those objects that fall outside a cluster (*outliers*), perhaps within a specified deviation, are candidate anomalies. Due to the popularity of this approach, and the fact that it is an unsupervised approach (like GBAD), we evaluate the effectiveness of the simple k-Means clustering approach on the cargo shipment data using the WEKA tool [WEKA]. For the simple k-Means approach (*SimpleKMeans*), given a set of n data points in d-dimensional space R^d, and an integer k, the problem is to determine a set of k points in R^d, called centers, so as to minimize the mean squared distance from each data point to its nearest center [Kanungo et al. 2000].

First, we randomly select 200 cargo records and generate a graph from the chosen records. Second, we run the graph through SUBDUE to determine the normative pattern in the graph. Then, we generate multiple anomalies of each of the anomaly types (modifications, insertions and deletions), where each of the induced anomalies is similar to the real-world examples mentioned earlier (e.g., change in the name of a port), and the anomaly is part of a normative pattern. Only choosing anomalies that are small deviations (relative to the size of the graph), all three of the GBAD algorithms are able to successfully find all of the anomalies, with only one GBAD-MDL test and one GBAD-P test reporting false positives.

Using these same 200 records, we then convert the data into the appropriate WEKA format for the k-Means algorithm. For each of the tests involving what were vertex modifications in the graph, and are now value or field modifications in the text file, the k-Means algorithm is able to successfully find all of the anomalies. Similar to how we have to adjust GBAD parameters, we have to increase the default WEKA settings for the number of clusters and the seed, as the defaults do not produce any anomalous clusters on the cargo test set which consists of 12 attributes for 200 records. By increasing the number of clusters to 8 and the seed to 31, we are able to discover the anomalous modifications, as shown in the example in Figure 6.4.

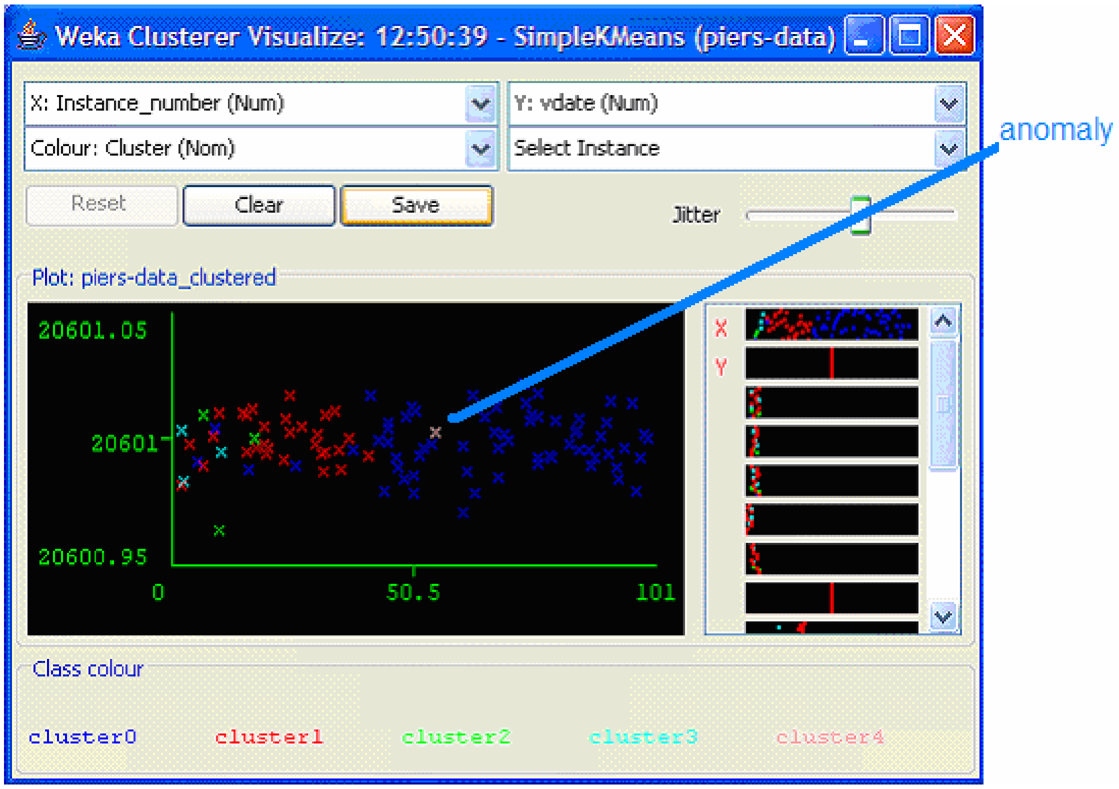

Figure 6.4 Results from k-Means clustering algorithm on cargo data with anomalous modification.

In these tests, a cluster is considered anomalous if it contains only a single instance.

However, for insertions and deletions, the k-Means approach is not effective, but in some ways, that is to be expected. The k-Means algorithm assumes that every specified record is of the same length. So, in order to simulate anomalous insertions, extra attributes must be added to represent the extra vertices, where the values for those attributes are NULL, unless an anomalous insertion is present. Yet, despite the additional non-NULL attribute when the anomaly is present, the k-Means algorithm never reports an anomalous cluster for any of the tests. When we increase the number of clusters and the seed, it only increases the number of false positives (i.e., clusters of single instances that are not the anomalous instances). This is surprising in that we would have assumed that the unique value for an individual attribute would have been discovered. However, again, we are attempting to simulate a structural change, which is something that the k-Means algorithm (or other traditional clustering algorithms) is not intended to discover.

Similarly, we can only simulate an anomalous deletion by replacing one of the record's attributed values with a NULL value. Again, at first, we would have considered this to be the same as a modification, and clearly identifiable by the k-Means algorithm. But, the algorithm was unable to find the anomalous deletion in any of the tests, which leads us to believe that the presence of a NULL value has an effect on the functionality of the k-Means algorithm. The importance of these tests is to show that for some anomalies (specifically modifications) traditional machine learning approaches like clustering are also effective, and at the same time, the inability to discover anomalous insertions and deletions further justifies the use of an approach like GBAD for structural anomalies. In addition, approaches like k-Means are only able to report the anomalous record – not the specific anomaly within the record.

The use of the k-Means clustering algorithm for anomaly detection and intrusion detection has been reported in other research efforts [Portnoy 1999][Caruso and Malerba 2004].For more information on how the WEKA tools work, please refer to the WEKA website [WEKA].

We also compared our algorithms against a traditional non-graph-based anomaly detection approach found in the commercially available application called

Gritbot, from a company called RuleQuest (http://www.RuleQuest.com/). The objective of the Gritbot tool is to look for anomalous values that would compromise the integrity of data that might be further analyzed by other data modeling tools.

There are two required input files for Gritbot: a .names file that specifies the attributes to be analyzed, and a .data file that supplies the corresponding comma-delimited data. There are several optional parameters for running Gritbot, of which the most important is the "filter level". By default, the filter level is set at 50%. The lower the filter level percentage, the less filtering that occurs, resulting in more possible anomalies being considered.

In order to compare Gritbot to our GBAD algorithms, we gave Gritbot the same cargo data files used in the previous experiments (formatted to the Gritbot specifications). Using the default parameters, no anomalies were reported. We then lowered the filter level to 0 (which specifies that all anomalies are requested). In every case, anomalies were reported, but none of the anomalies reported were the ones we had injected into the data set. So, we increased the number of samples from 200 shipments to ~1000 shipments, so that Gritbot could infer more of a statistical pattern, and then randomly injected a single modification to the country-of-origin attribute. In the cargo data files, all of the country-of-origins were "JAPAN", except for the randomly selected records where the value was changed to "CHINA". Again, Gritbot did not report this anomaly (i.e. 1020 cases of "JAPAN" and one case of "CHINA"), and instead reported a couple of other cases as anomalous.

While we consider the existence of a record with "CHINA" as anomalous, Gritbot does not view that as an anomaly. The issue is that Gritbot (and this is similar to other outlier-based approaches), does not treat discrete attributes the same as numeric attributes. This is because Gritbot views continuous distributions (such as "age") as a much easier attribute to analyze because the distribution of values leads to certain expectations. While discrete distributions are more difficult because there is not a referential norm (statistically), it limits the tool's ability to provide its user with a comprehensive list of anomalies. That is not to say that Gritbot will not discover anomalous discrete values - it will if it can determine a statistical significance. For example, we found (when examining by hand) records that contained a significant number of identical attribute values (e.g., COUNTRY, FPORT, SLINE, VESSEL). In our data set, approximately 250 out of the approximately 1,000 records had identical SLINE values. When we arbitrarily modified the SLINE value of one of these cases from "KLIN" to "PONL" (i.e., another one of the possible SLINE values from this data set), Gritbot did not report the anomaly. When we changed it to "MLSL", Gritbot still did not report it. However, when we changed it to "CSCO", Gritbot reported that case as being anomalous (albeit, not the most anomalous). Why? This behavior is based on what Gritbot can determine to be statistically significant. Of all of the ~1,000 records, only 1 has an SLINE value of "MLSL", and only 3 have a value of "PONL". However, there are 123 records with an SLINE value of "CSCO". Thus, Gritbot was able to determine that a value of "CSCO" among those ~250 records is anomalous because it had enough other records containing the value "CSCO" to determine that its existence in these other records was significant. In short, the behavior depends upon the definition of what is an anomaly.

Gritbot's approach to anomaly detection is common among many other outlier-based data mining approaches. However, in terms of finding what we would consider to be anomalous (small deviations from the norm), Gritbot's approach typically does not find the anomaly.

6.2 Intrusion Detection

One of the more applied areas of research when it comes to anomaly detection can be found in the multiple approaches to intrusion detection. The reasons for this are its relevance to the real world problem of networks and systems being attacked, and the ability of researchers to gather actual data for testing their models. Perhaps the most used data set for this area of research and experimentation is the 1999 KDD Cup network intrusion dataset [KDD Cup 1999].

In 1998, MIT Lincoln Labs managed the DARPA Intrusion Detection Evaluation Program. The objective was to survey and evaluate research in intrusion detection. The standard data set consisted of a wide variety of intrusions simulated in a military network environment. The 1999 KDD Cup intrusion detection dataset consists of a version of this data. For nine weeks, they simulated a typical U.S. Air Force local-area network, initiated multiple attacks, and dumped the raw TCP data for the competition.

The KDD Cup data consists of connection records, where a connection is a sequence of TCP packets. Each connection record is labeled as either “normal”, or one of 37 different attack types. Each record consists of 31 different features (or fields), with features being either continuous (real values) or discrete. In the 1999 competition, the data was split into two parts: one for training and the other for testing. Groups were then allowed to train their solutions using the training data, and were then judged based upon their performance on the test data.

6.2.1 GBAD

Since the GBAD approach uses unsupervised learning, we will run the algorithms on the test data so that we can judge our performance versus other approaches. Also, because we do not know the possible structural graph changes associated with network intrusions, we will have to run all three algorithms to determine which algorithms are most effective for this type of data. Each test contains 50 essentially random records, where 49 are normal records and 1 is an attack record, where the only controlled aspect of the test is that there is only one attack record per data set. This is done because the test data is comprised of mostly attack records, which does not fit our definition of an anomaly, where we are assuming that anomalous substructures are rare. Fortunately, this again is a reasonable assumption, as attacks would be uncommon in most networks.

Not surprisingly, each of the algorithms has a different level of effectiveness when it comes to discovering anomalies in intrusion detection data. Using GBAD-MDL, our ability to discover the attacks is relatively successful. Across all data sets, 100% of the attacks are discovered. However, all but the *apache2* and *worm* attacks produce some false positives. 42.2% of the test runs do not produce any false positives, while runs containing *snmpgetattack*, *snmpguess, teardrop* and *udpstorm* attacks contribute the most false positives. False positives are even higher for the GBAD-P algorithm, and the discovery rate of actual attacks decreases to 55.8%. GBAD-MPS shows a similarly bad false positive rate at 67.2%, and an even worse discovery rate at 47.8%.

It is not surprising that GBAD-MDL is the most effective of the algorithms, as the data consists of TCP packets that are structurally similar in size across all records. Thus, the inclusion of additional structure, or the removal of structure, is not as relevant for this type of data, and any structural changes, if they exist, would consist of value modifications.

6.2.2 Comparison to Other Graph-based Approaches

In order to better understand the effectiveness of the GBAD algorithms on intrusion detection data, we will compare our results with the graph-based approaches of Noble and Cook [Noble and Cook 2003]. As was presented earlier in this work, they proposed two approaches to discovering anomalies in data represented as a graph. Their *anomalous substructure detection* method attempts to find unusual substructures within a graph by finding those substructures that compress the graph the least, compared to our GBAD-MDL approach which uses compression to determine which substructures are closest to the best substructure (i.e., the one that compresses the graph the most). In their results, they use the inverse of the ratio of true anomalies found over the total number of anomalies reported, where the lower the value (i.e., a greater percentage of the reported anomalies are the network attacks), the more effective the approach for discovering anomalies.

The other approach presented is what they call *anomalous subgraph detection*. The objective with this method is to compare separate structures (subgraphs) and determine how anomalous each subgraph is compared to the others. This is similar to our approaches in that the minimum description length is used as a measurement of a substructure's likelihood of existence within a graph. However, in order to implement this approach, the graph must be divided into clearly defined subgraphs so that a proper comparison can be performed. The basic idea is that every subgraph is assigned a value of 1, and the value decreases as portions of the subgraph are compressed away. In the end, the subgraphs are ranked highest to lowest, with the higher the value (i.e., closest to, or equal to, 1), the more anomalous the subgraph. While this works well on intrusion detection data, their approach is restricted to domains where a clear delineation (i.e., subgraphs) must be defined. In other words, the delineation occurs when a graph can be sub-divided into distinct subgraphs, with each subgraph representing a common entity. For example, in domains like terrorist or social networks, this type of delineation may be difficult and subjective.

Using the same set of KDD Cup intrusion detection data as set forth previously, we can compare GBAD-MDL (since it performed the best on this set of data) to both of these approaches, using the same anomalous attack ratios used by Noble and Cook. The ratio used in their work is an inverse fraction of correctly identifying the attacks among all of the attacks reported. For example, if 10 anomalies are reported, but only 1 of them is the actual attack, then the fraction is 1/10, and the inverse is the score of 10, where obviously the lower the score the better. Their anomalous substructure detection method achieves an average anomalous ratio of 8.64, excluding the *snmpgetattack* and *snmpguess* attacks, while using the same scoring ratio, GBAD-MDL generates an average of 7.22 *with snmpgetattack* and *snmpguess* included. In Noble and Cook's paper, both the *snmpgetattack* and *snmpguess* attacks were excluded from the anomalous substructure detection approach results because they had high average attack values of around 2211 and 126 respectively (i.e., too many false positives). However, GBAD-MDL is much more successful at discovering these two attack types, as their respective averages are 8.55 and 7.21. Then, for their anomalous subgraph detection approach, they get an average ranking of 4.75, whereas the GBAD-MDL algorithm is able to achieve a better average ranking of 3.02. Figure 6.5 shows the ranking results for each of the different attack types.

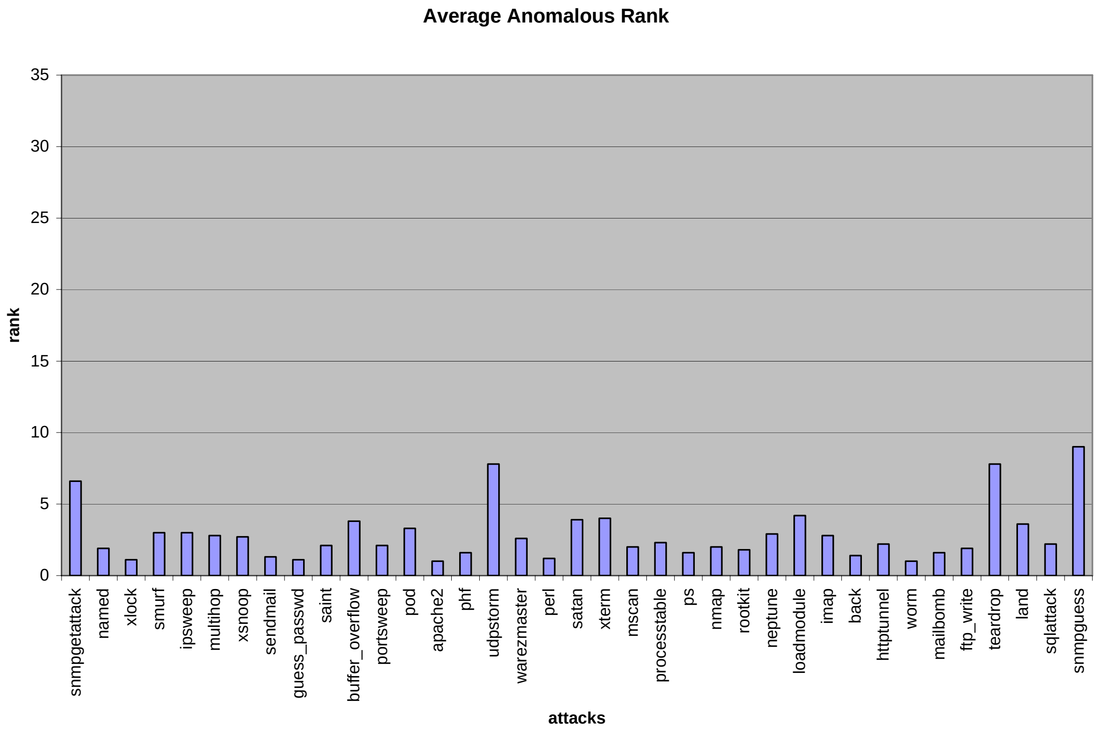

Figure 6.5 Average anomalous ranking using GBAD-MDL on KDD intrusion detection data.

These results, when compared to the ones presented in Noble and Cook's paper [Noble and Cook 2003], not only show an overall average improvement, but also again show a significant improvement when it comes to effectively discovering the *snmpgetattack* and *snmpguess* attacks, which both had values over 20 using the anomalous subgraph detection approach, whereas the GBAD-MDL algorithm was under 10 for both attack types. It should also be noted that the false positives are mostly due to the fact that we have to increase the anomalous threshold in order to detect some of the anomalous patterns. Unlike our assumption that anomalies are small deviations from the normative pattern, several of the attack records are actually large deviations from the norm.

6.3 Other Data Sets

We have also tried our algorithms on other publicly available data sets in which we would like to identify anomalous behaviors.

6.3.1 Enron E-mail

One of the more recent domains that has become publicly available is the data set of e-mails between management from the Enron corporation. The Enron e-mail dataset was made public by the Federal Energy Regulatory Commission during its investigation. After subsequent data integrity resolutions, as well as the removal of some e-mails due to requests from affected employees, William Cohen at CMU put the dataset on the web for researchers. From that dataset, Shetty and Adibi further cleaned the dataset by removing duplicate e-mails, and putting the final set into a publicly available MySql database (http://www.isi.edu/~adibi/Enron/Enron.htm). This dataset contains 252,759 messages from 151 employees distributed in approximately 3000 user-defined folders.

In Priebe et al's work, they used what are called "scan statistics" on a graph of the Enron data that is represented as a time series [Priebe et al, 2005]. While their approach detects statistically significant events (excessive activity), without further analysis, they are unable to determine whether the events are relevant (like insider trading). In Shetty and Adibi's paper, they propose analyzing the entropy of the Enron data represented as a graph [Shetty and Adibi, 2005]. However, with their approach of "event based graph entropy", the objective was to discover the more interesting nodes in the graph, which does not necessarily mean they are anomalous. Huang and Zeng proposed using several link prediction approaches for analyzing the Enron data for anomalous e-mails [Huang and Zeng, 2006]. Again, they report what they consider to be anomalous e-mails, but they state that without knowing which e-mails are truly anomalous, they are just making conjectures. They do introduce some fake e-mails into the data with the purpose of showing that their approaches do indicate their induced fake e-mails as either the most anomalous or equal to other e-mails that scored high.

This Enron e-mail database consists of messages not only between employees but also from employees to external contacts. In addition to providing the e-mails, the database also consists of employee information such as their name and e-mail address. However, since we do not have information about their external contacts, we decided to limit our graph to the Enron employees and just their correspondences, allowing us to create a more complete “social” structure. In addition, since the body of e-mails consist of many words (and typos), we limited the textual nature of the e-mails to just the subject headers. From these decisions, we created a graph consisting of the structure shown in Figure 6.6.

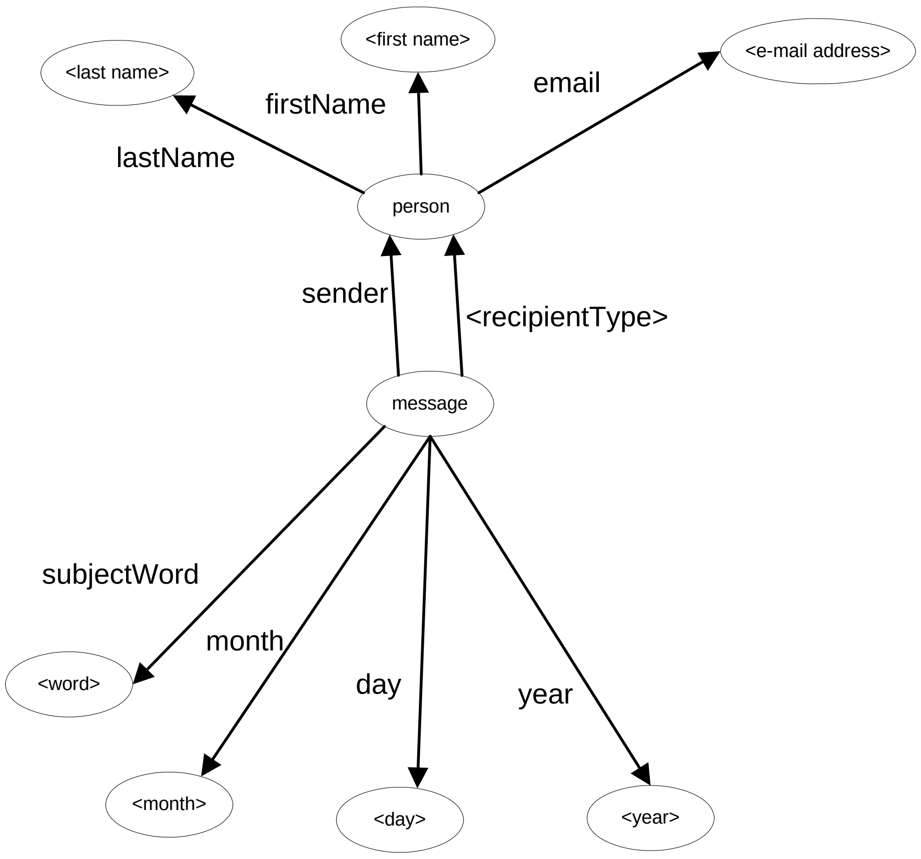

Figure 6.6 Graphical representation of Enron e-mail.

In Figure 6.6, the message vertex can have multiple edges to multiple subject words, and multiple recipient type edges (i.e., TO, CC and BCC) to multiple persons.

Running the GBAD algorithms on this data set produced the small normative pattern shown in Figure 6.7.

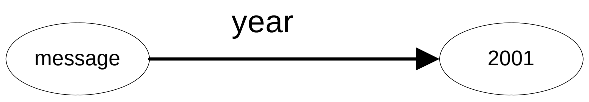

Figure 6.7 Normative pattern from Enron e-mail data set.

This was an expected normative pattern because most of the e-mail was from the year 2001, with little regularity beyond the fact that messages were sent.

So, considering the small size of the normative pattern, we did not run the GBAD-MDL and GBAD-MPS algorithms, as clearly nothing of importance would be derived from a modification or deletion to this normative pattern. However, running the GBAD-P algorithm resulted in the substructure shown in Figure 6.8.

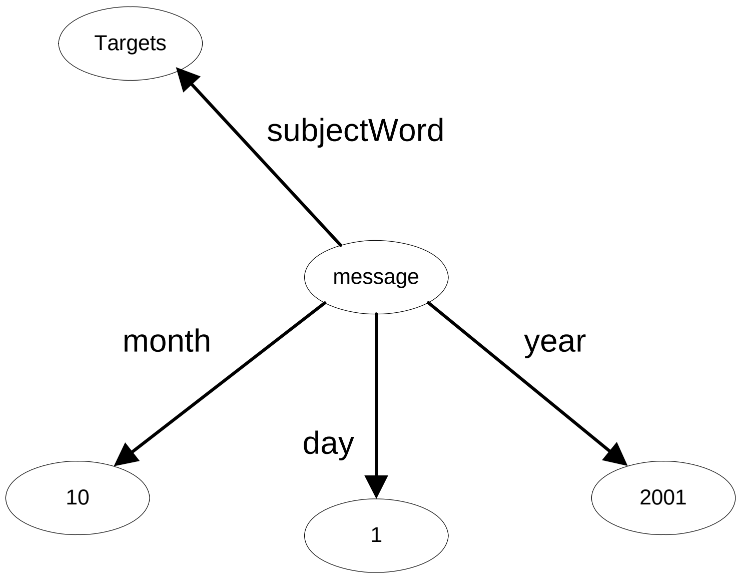

Figure 6.8 Results from running GBAD-P on Enron e-mail data set.

It is interesting to point out that 859 messages were sent on October 1, 2001, and of all of the messages in the data set, this was the only one with a subject of “Targets”, and 1 of only 36 messages in the entire dataset that had the word “Targets” anywhere in its subject line, and no messages anywhere that were a response to this message.

Unfortunately, just as Huang and Zeng pointed out, without actually knowing which e-mails are anomalous, we are all just making conjectures.

6.3.2 Internet Movie Database (IMDB)

Another common source of data mining research is the Internet Movie Data Base (http://www.imdb.com/). The database consists of hundreds of thousands of movies and television shows, with all of the credited information such as directors, actors, writer, producers, etc. In their work on semantic graphs, Barthelemy et al. proposed a statistical measure for semantic graphs and illustrated these semantic measures on graphs constructed from terrorism data and data from the IMDB [Barthelemy et al, 2005]. While they were not directly looking for anomalies, their research presented a way to measure useful relationships so that a better ontology could be created. As was mentioned previously, using bipartite graphs, Sun et al. presented a model for scoring the normality of nodes as they related to the other nodes [Sun et al, 2005]. Using the IMDB database as one of their datasets, they analyzed the graph for just anomalous nodes.

In order to run our algorithms on the data, due to the voluminous amount of information, we chose to create a graph of the key information (title, director, producer, writer, actor, actress and genre) for the movies from 2006. Running the GBAD algorithms on this data set produced the normative pattern shown in Figure 6.9.

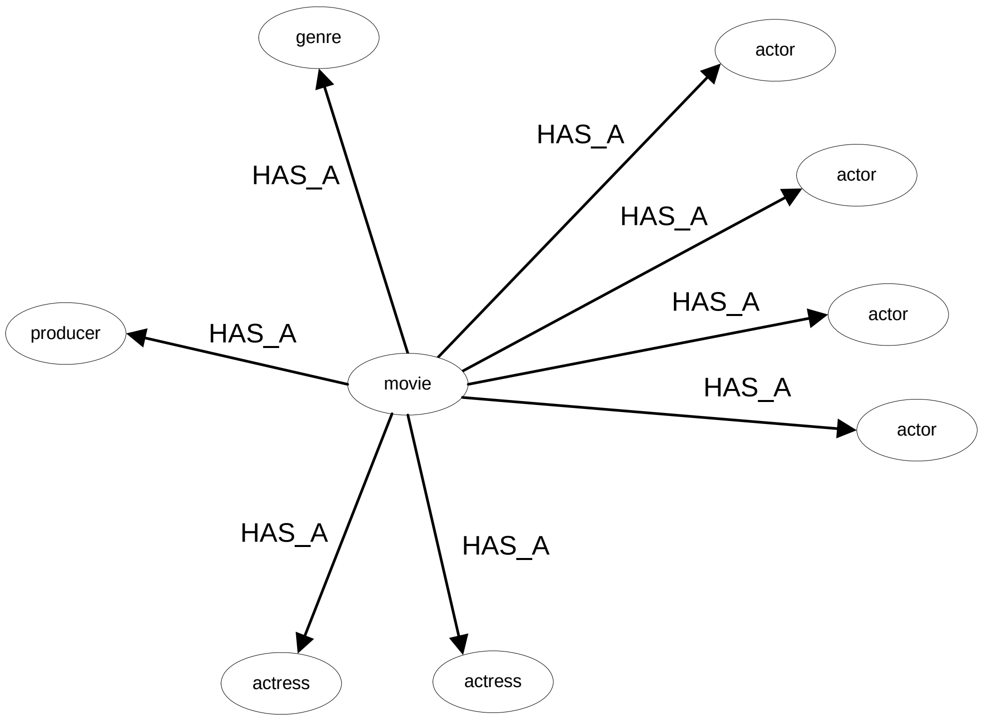

Figure 6.9 Normative pattern from graph representation of movie data.

This pattern is not surprising, as one would expect that a movie would consist of multiple actors and actresses, and considering the size of the database, it is possible that runs made with much larger limits might produce a larger normative pattern and include some of the other movie elements like director and writer. The lack of these elements in the normative pattern can be due to a couple of reasons. One is that genres like reality shows do not require directors and writers, and many documentaries do not have writers credited in the IMDB. Another is the possible resource constraints that have been discussed previously. If we were to allow GBAD to run with a larger beam and higher limits, a larger normative pattern might be discovered that would include the missing directors and writers.

Running the GBAD-MDL and GBAD-MPS algorithms on the IMDB data produced a variety of anomalies that all scored equally. The GBAD-MDL algorithm reported a single anomalous instance where an actor label was replaced by an anomalous actress label. The GBAD-MPS algorithm reported multiple anomalous instances consisting of a writer replacing an actor, a writer replacing the producer, another genre replacing the producer, a director replacing the producer, another actor replacing the producer and another actress replacing the producer. For the GBAD-P algorithm, the anomalous extension consisted of the title of the movie. Considering every movie has a different title (in most cases), this was an expected anomalous extension.

The issue with analyzing the move data set with these algorithms is that this approach is set up to find the small deviations from the norm that are indicative of fraud. While there are probably anomalies in the IMDB, the anomalies are more than likely a result of information being input incorrectly into the database, or just structural changes due to the construction of different movies. Again, without expert knowledge of the data and specifically any insights into actual anomalies, it is difficult to make any conjectures about what we are observing.

6.4 Summary

We have observed that even with non-synthetic data, as long as there is a large enough normative pattern, the data is fairly regular (such that the normative pattern is prevalent in most of the data), and the anomaly consists of a small deviation from the norm, these three algorithms are able to target specific anomaly types. Validating the results shown with the synthetic data, the algorithms obtain a near 100% discovery rate in the cargo shipment domain, with minimal or no false positives. We also validated our algorithms by not only demonstrating an improvement over other graph-based anomaly detection approaches, but also verifying these algorithms as an effective means of discovering anomalies in the very important real-world domain of network intrusions. In addition, we were able to analyze various other diverse data sets where unusual patterns were discovered that consisted of small deviations from a normative pattern.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1. Conclusions

In this work we presented our definition of a graph-based anomaly and how that is manifested in data that is represented as a graph. The purpose of this work was to present an approach for discovering the three possible graph anomalies: modifications, insertions and deletions. Using a practical definition of fraud, we designed algorithms to specifically handle the scenario where the anomalies are small deviations to a normative pattern.

In Chapter 4 we described three novel algorithms, each with the goal of uncovering one of the specified anomalous types. With the aid of several simple examples, we were able to describe the approaches and the simplicity of their implementation. In Chapter 5 we validated all three approaches using synthetic data. The tests verified each of the algorithms on graphs and anomalies of varying sizes, with the results showing very high detection rates with minimal false positives. Chapter 6 further validated the algorithms using real-world cargo data and actual fraud scenarios injected into the data set. Despite a less regular set of data, normative patterns did exist, and changes to those prevalent substructures were detected with 100% accuracy and no false positives. We also compared our approach on this dataset against other non-graph-based approaches where we were equally effective in all cases and better in others. We then compared our algorithms against a graph-based approach using intrusion detection data, again with better discovery rates and lower false positives. We also looked at other real-world datasets where we were able to show unusual patterns in two diverse domains. In short, the GBAD algorithms presented in this work are able to consistently discover all graph-based anomalies that are comprised of the smallest deviation of the normative pattern, with minimal false positives.

7.2. Future Work

There have been many approaches to anomaly detection over the years, most of which have been based on statistical methods for determining outliers. As was shown in this paper, recent research in graph-based approaches to data mining have resulted in new methods of anomaly detection. Our work shows promising approaches to this problem, particularly as it is applied to fraud detection. However, there are still many avenues to be explored.

7.2.1 Improved Picture of Anomaly

Currently, there is no connection between compressed substructures. In other words, once instances of a particular substructure have been compressed to a single vertex (i.e., when running multiple iterations of the GBAD-P algorithm), and a link (and vertex) extends from that compressed substructure, there is no information telling us what actual vertex is connected to that extension. If we can save that information for future iterations, that could prove to be useful for two reasons. One, we could possibly generate a better probabilistic model for determining which extensions are actually more anomalous. Second, it would allow us to create a better picture of the anomaly, as an analyst would be able to view the entire chain of connections in totality associated with the anomaly.

7.2.2 User Input to Guide Detection

A common data mining or fraud detection approach is to allow the user to provide feedback. Used as either a learning process, or merely as a means of adjusting parametric thresholds, the user is able to guide their application towards an expected result, or away from an uninteresting result. One possible modification to our approaches could be the input of a known normative pattern. For instance, a domain expert could know what the normative pattern is for a particular data set. By supplying that pattern, only deviations from that pattern would be explored. This could also be particularly useful if the non-user-specified normative pattern is very small, making it difficult to discover anomalous deviations. The disadvantage of this approach is that it would require a subgraph isomorphism, which is computationally intractable.

7.2.3 Other Probabilistic Ratios

The probabilistic algorithm presented in this work uses a simple probabilistic approach. More advanced statistical approaches should be investigated to determine possibly better choices for the anomalous substructure. For instance, currently, if there is only one type of extension (i.e., no other extensions available), that extension will get an anomalous score of 1.0 (i.e., maximum possible score), where the closer the score is to 0.0 the better. While the user is currently able to threshold the anomalous score (i.e., ignore reported anomalies above a certain score), perhaps other statistical comparisons could be made that would further delineate the targeted extension. Also, currently the extensions are evaluated against just other extensions of the normative pattern. Perhaps other substructures that were not the best substructure could be used in the evaluation of anomalous extensions.

7.2.4 Other Evaluation Metrics

The minimum description length principle was used as the metric for determining the normative pattern, as well as by the GBAD-MDL algorithm for discovering substructures with minor modifications. This metric is a key component of the algorithms presented in this work. However, other graph-based approaches have used various other metrics, such as size or coverage, for determining the normative pattern in a graph. Future work should include an analysis of these other metrics in lieu of the MDL approach.

7.2.5 Other Domains

Since 9/11, one of the more common domains used in data mining consists of terrorist activity and relationships. Organizations such as the Department of Homeland Security use various techniques to discover the inherent patterns in the network representation of known terrorists and their relations [Kamarck 2002]. Much research has been applied to not only understanding terrorist networks [Sageman 2004], but also discovering the patterns that discriminate the terrorists from the non-terrorists [Taipale 2003]. Much of this area of research has also been applied to what is known as social network analysis, which is a more general term for the measuring and mapping of relationships between people, places and organizations [Mukherjee and Holder 2004].

Through the Evidence Assessment, Grouping, Linking, and Evaluation (EAGLE) program, under the auspices of the Air Force Research Lab (AFRL), we have been able to gather counter-terrorism data. While the data is simulated, it does represent scenarios based on input from various intelligence analysts [Coble 2006]. The data represents different terrorist organization activities as they relate to the exploitation of vulnerable targets. Our goal is to use this data as another example of real-world data to further validate the effectiveness of this approach. If a terrorist can be distinguishable from a non-terrorist by a small deviation in a normative pattern, we should be able to discover actual terrorist instances within a network of people and relationships.

Another domain worth investigating would be data from the Financial Crimes Enforcement Network (FinCEN). The purpose of FinCEN is to analyze financial transactions for possible financial crimes including terrorist financing and money laundering. Again, if illegal transactions consist of small deviations from normal transactions, we should be able to uncover genuine fraudulent activity within a network of people and their related monetary dealings.

Similar techniques could be applied to a myriad of domains, including telecommunications call records and credit card transactions. In short, any data source where transactions and relationships can be represented structurally as a graph, and possible anomalous behavior consists of minor deviations from normal patterns, these approaches to graph-based anomaly detection could prove to be a viable alternative to traditional data mining endeavors. In addition, by analyzing the effectiveness of our algorithms against real-world, labeled data sets, we can establish a baseline of comparison that can be used in subsequent anomaly detection endeavors.

REFERENCES

Aha, D., Kibler, D. and Albert, M. *Instance-based Learning Algorithms. Machine Learning*, Vol. 6, Issue 1, 37-66, 1991.

Barthélemy, M., Chow, E. and Eliassi-Rad, T, *Knowledge Representation Issues in Semantic Graphs for Relationship Detection*. AI Technologies for Homeland Security: Papers from the 2005 AAAI Spring Symposium, AAAI Press, 2005, pp. 91-98.

Boykin, P. and Roychowdhury, V. *Leveraging Social Networks to Fight Spam*. IEEE Computer, April 2005, 38(4), 61-67, 2005.

Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A. and Wiener, J. *Graph Structure in the Web*. Computer Networks, Vol. 33, 309-320, 2000.

Caruso, C. and Malerba, D. *Clustering as an add-on for firewalls*. Data Mining, WIT Press, 2004.

Chakrabarti, D. *AutoPart: Parameter-Free Graph Partitioning and Outlier Detection*. Knowledge Discovery in Databases: PKDD 2004, 8th European Conference on Principles and Practice of Knowledge Discovery in Databases, 112-124, 2004.

Chung, F., Lu, L., Vu, V. *Eigenvalues of Random Power Law Graphs*. Annals of Combinatorics, 7, 21-33, 2003.

Coble, J. *Relational Discovery in Sequentially-connected Data Streams: Efficient Algorithms for Lossless Pattern Discovery and Change Detection*. PhD Thesis, University of Texas at Arlington, 2006.

Coble, J., Cook, D., Rathi, R and Holder, L. *Iterative Structure Discovery in Graph-Based Data.* International Journal of Artificial Intelligence Techniques, 14(1-2), 2005.

Cook, D. and Holder, L. *Graph-based data mining*. IEEE Intelligent Systems 15(2), 32-41, 2000.

Cortes, C., Pregibon, D. and Volinsky, C. *Computational Methods for Dynamic Graphs*. Journal of Computational and Graphical Statistics, Vol. 12, 950-970, 2003.

Customs and Border Protection Today: *Illegal textile entries: a way to save a few bucks?* March 2003. (http://www.cbp.gov/xp/CustomsToday/2003/March/illegal.xml)

Dehaspe, L. and Toivonen, H. Discovery of frequent datalog patterns. Data Mining and Knowledge Discovery. 3(1): 7-36, 1999.

Denning, D. *An Intrusion-Detection Model*. IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, 1987, pp 222-232, 1987.

De Raedt, L. and Kramer, S. *The levelwise version space application and its application to molecular fragment finding*. IJCAI '01, Seventeenth International Joint Conference on Artificial Intelligence, volume 2, pages 853-859, 2001.

Eberle, W. and Holder, L. *Detecting Anomalies in Cargo Shipments Using Graph Properties*. Proceedings of the IEEE Intelligence and Security Informatics Conference, 2006.

Faloutsos, M., Faloutsos P., Faloutsos, C. *On Power-law Relationships of the Internet Topology*. Proceedings of the conference on applications, technologies, architectures, and protocols for computer communications, SIGCOMM, pp. 251-262, 1999.

Gross, J, and Yellen, J. *Graph Theory and Its Applications*. CRC Press. 1999.

Gudes, E. and Shimony, S. *Discovering Frequent Graph Patterns Using Disjoint Paths*. IEEE Transactions of Knowledge and Data Engineering, 18(11), November 2006.

Hampton, M. and Levi, M. *Fast spinning into oblivion? Recent developments in money-laundering policies and offshore finance centres.* Third World Quartely, Volume 20, Number 3, June 1999, pp. 645-656, 1999.

Holder, L., Cook, D. and Djoko, S. *Substructure Discovery in the SUBDUE System*. Proceedings of the AAAI Workshop on Knowledge Discover in Databases, pp. 169-180, 1994.

Hu, W., Liao, Y. and Vemuri, V. *Robust Support Vector Machines for Anomaly Detection in Computer Security*. International Conference on Machine Learning and Applications, Los Angeles, California, June 2003.

Huan, J., Wang, W. and Prins, J. *SPIN: Mining Maximal Frequent Subgraphs from Graph Databases*. Knowledge Discovery and Data Mining, KDD '04, 2004.

Huang, Z. and Zeng, D. *A Link Prediction Approach to Anomalous Email Detection*. 2006 IEEE International Conference on Systems, Man, and Cybernetics, Taipei, Taiwan, October 8 -11, 2006.

Ide, T. and Kashima, H. *Eigenspace-based Anomaly Detection in Computer Systems*. Proceedings of the Tenth ACM SIGDD International Conference on Knowledge Discovery and Data Mining, 440-449, 2005.

Kamarck, E. *Applying 21st Century Government to the Challenge of Homeland Security*. Harvard University, PriceWaterhouseCoopers, 2002.

Kanungo, T, Mount, D., Netanyahu, N., Piatko, C., Silverman, R. and Wu, A. *The Analysis of a Simple k-Means Clustering Algorithm*. Proceedings on the 16th Annual Symposium on Computational Geometry, 100-109, 2000.

KDD Cup 1999. Knowledge Discovery and Data Mining Tools Competition. *http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html*. 1999.

Kruegel, C., Mutz, D., Robertson, W. and Valeur, F. *Bayesian Event Classification for Intrusion Detection*. Proceedings of the 19th Annual Computer Security Applications Conference, 2003.

Kumar, S. *Classification and Detection of Computer Intrusions*. PhD thesis, Purdue, IN, 1995.

Kuramochi, M. and Karypis, G. *An Efficient Algorithm for Discovering Frequent Subgraphs*. IEEE Transactions on Knowledge and Data Engineering, pp. 1038-1051, 2004.

Kuramochi, M. and Karypis, G. *Grew – A Scalable Frequent Subgraph Discovery Algorithm.* IEEE International Conference on Data Mining (ICDM '04), 2004.

Lane, T. *Machine Learning Techniques for the Domain of Anomaly Detection for Computer Security*. PhD thesis, Purdue, IN. 2000.

Lee, W. and Xiang, D. *Information-theoretic measures for anomaly detection*. IEEE Symposium on Security and Privacy, 2001.

Lee, W., Stolfo, S. and Mok, K. *A Data Mining Framework for Building Intrusion Detection Models*. IEEE Symposium on Security and Privacy, 1999.

Lee, W. and Stolfo, S. *Data Mining Approaches for Intrusion Detection*. Proceedings of the 7th USENIX Security Symposium, 1998.

Lin S. and Chalupsky, H. *Unsupervised Link Discovery in Multi-relational Data via Rarity Analysis*. Proceedings of the Third IEEE ICDM International Conference on Data Mining, 171-178, 2003.

Mae Dey Newsletter: *Customs Seizes Weapons*. Vol. 23, Issue 4, August/September (2003).

Maxion, R. and Tan, K. *Benchmarking Anomaly-Based Detection Systems*. In International Conference on Dependable Systems and Networks, IEEE Computer Society Press, 623-630, 2000.

Mitchell, T. *Machine Learning*. McGraw Hill. 1997

Mukherjee, M. and Holder, L. *Graph-based Data Mining on Social Networks*. Workshop on Link Analysis and Group Detection, KDD, 2004.

Müller, K., Mika, S., Rätsch, G., Tsuda, K. and Scholkopf, B. *An Introduction to Kernel-Based Learning Algorithms*, IEEE Transactions on Neural Networks, Vol. 12, No. 12, March 2001.

Nijssen, S. and Kok, J. *Faster association rules for multiple relations*. IJCAI ’01, Seventeenth International Joint Conference on Artificial Intelligence, volume 2, pages 891-896, 2001.

Noble, C. and Cook, D. *Graph-Based Anomaly Detection*. Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 631-636, 2003.

Portnoy, L., Eskin, E. and Stolfo, S. *Intrusion detection with unlabeled data using clustering*. Proceedings of ACM CSS Workshop on Data Mining Applied to Security, 2001.

Priebe, C., Conroy, J., Marchette, D. and Park, Y. *Scan Statistics on Enron Graphs*. Computational and Mathematics Organization Theory, Volume 11, Number 3, p229 - 247, October 2005

Rattigan, M. and Jensen, D. *The case for anomalous link discovery*. ACM SIGKDD Explor. Newsl., 7(2):41-47, 2005.

Rissanen, J. *Stochastic Complexity in Statistical Inquiry*. World Scientific Publishing Company, 1989.

Sageman, M. *Understanding Terror Networks*. University of Pennsylvania Press, 2004.

Scott, J. *Social Network Analysis: A Handbook*. SAGE Publications, Second Edition, 72-78, 2000.

Shetty, J. and Adibi, J. *Discovering Important Nodes through Graph Entropy: The Case of Enron Email Database*. KDD, Proceedings of the 3rd international workshop on Link discovery, 74-81, 2005.

Staniford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J. Levitt, K., Wee, C., Yip, R. and Zerkle, D. *GrIDS – A Graph Based Intrusion Detection System for Large Networks*. Proceedings of the 19th National Information Systems Security Conference, 1996.

Stein, G., Chen, B., Wu, A. and Hua, K. *Decision Tree Classifier for Network Intrusion Detection with GA-based Feature Selection*. ACM Southeast Regional Conference, 136-141, 2005.

Sun, J, Qu, H., Chakrabarti, D. and Faloutsos, C. *Relevance search and anomaly detection in bipartite graphs*. SIGKDD Explorations 7(2), 48-55, 2005.

Taipale, K. *Data Mining and Domestic Security: Connecting the Dots to Make Sense of Data*. Columbia Science and Technology Law Review, 2003.

Thomas, L., Valluri, S. and Karlapalem, K. *MARGIN: Maximal Frequent Subgraph Mining*. Sixth International Conference on Data Mining (ICMD '06), 109-1101, 2006.

U.S. Customs Service: *1,754 Pounds of Marijuana Seized in Cargo Container at Port Everglades*. November 6, 2000. (http://www.cbp.gov/hot-new/pressrel/2000/1106-01.htm)

Washio, T. and Motoda, H. *State of the art of graph-based data mining*. ACM SIGKDD Explorations Newsletter, 5(1):59-68, July 2003.

Webster's New Universal Unabridged Dictionary, Barnes & Noble Books, 1989.

Webster's New World Thesaurus, New Revised Edition, Prentice Hall, 1971.

WEKA, *http://www.cs.waikato.ac.nz/~ml/index.html*.

West, D. *Introduction to Graph Theory*. Prentice-Hall International. Second Edition. 2001.

Witten, I. and Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufman, Second Edition, 2005.

Yan, X. and Han, J. *gSpan: Graph-Based Substructure Pattern Mining*. Proceedings of International Conference on Data Mining, ICDM, pp. 51-58, 2002.

Zeng, Z., Wang, J., Zhou, L. and Karypis, G. *Coherent closed quasi-clique discovery from large dense graph databases*. Conference on Knowledge Discovery in Data, SIGKDD, 797-802, 2006.

BIOGRAPHICAL INFORMATION

William Fred Eberle received his Bachelor of Arts in Computer Science in 1986 from the University of Texas at Austin. In 1991, while working for General Dynamics in their flight simulation laboratory, he received his Master of Science in Computer Science from the University of Texas at Arlington, with an emphasis in artificial intelligence and natural language processing. After 11 years in the telecommunications industry, he went back to the University of Texas at Arlington, where, in 2007, while researching new techniques in fraud detection, he received his Ph.D. in Computer Science and Engineering.