Detecting Anomalies in Graphs with Numeric Labels

Michael Davis
mdavis05@qub.ac.uk

Weiru Liu
w.liu@qub.ac.uk

Paul Miller
p.miller@ecit.qub.ac.uk

School of Electrical and Electronic Engineering and Computer Science, Queen's University, Belfast, BT7 1NN, UK

George Redpath
gredpath@tycoint.com

CEM Systems, Airport Road West, Belfast, BT3 9ED, UK

ABSTRACT

This paper presents YAGADA, an algorithm to search labelled graphs for anomalies using both structural data and numeric attributes. YAGADA is explained using several security-related examples and validated with experiments on a physical Access Control database. Quantitative analysis shows that in the upper range of anomaly thresholds, YAGADA detects twice as many anomalies as the best-performing numeric discretization algorithm. Qualitative evaluation shows that the detected anomalies are meaningful, representing a combination of structural irregularities and numerical outliers.

Keywords

graph mining, transaction graphs, anomaly detection

1. INTRODUCTION

Graph representations are useful in many domains where data has a natural structure which cannot be easily represented as a set of attributes. Graphs are commonly used to represent complex structures such as social networks, biological or ecological processes and computer networks. Although graphs are often used to visualise networks for analysis by humans [9], machine learning techniques can improve on this by exposing knowledge hidden within the network structure. This paper presents a novel algorithm, YAGADA[1], to detect anomalies in labelled graphs.

Our research field is the security applications of network analysis, which include fraud detection [5] and intrusion detection [9]. In this paper, we analyse data from a physical Access Control System (ACS), of the type used to secure doors in buildings such as airports, power stations, hospitals and offices. The current generation of ACSs can detect *suspicious events*—such as the use of an ID Card which has been reported stolen, or an attempt to physically force a door—but cannot detect *suspicious patterns*—transactions which are innocent in themselves but anomalous when considered in context: an airport technician who regularly hangs around in the baggage handling area, or a clerk who is spending an unusually long time on their own in the cash room.

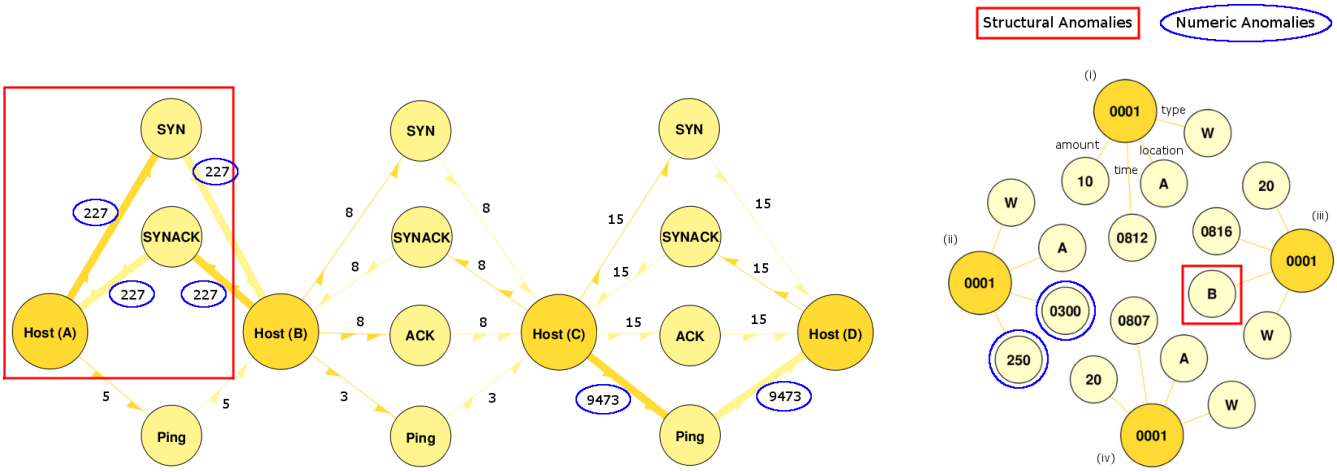
There is little published research on mining physical Access Control databases. [2] presents four algorithms to detect suspicious patterns: temporal patterns ("zone overstay"); repetitive access patterns; displacement patterns; and out-of-sequence patterns. The patterns are detected by comparing timing values to a threshold. This produces a binary result ("normal" or "suspicious") but does not indicate the degree of suspiciousness. If the "normal" duration of stay varies (*e.g.*, at different times of day), a single parameter cannot represent this. Furthermore, the algorithms are based on domain knowledge. In contrast, our approach represents the transactions as a graph, using a single algorithm to detect both *structural anomalies* (unusual paths through the building) and *numeric anomalies* (unusual timing data).

Existing graph-based anomaly detection algorithms [5, 14] define "normal" patterns in terms of frequent substructures. Subdue [4] uses a greedy beam search to discover frequent substructures and evaluates them using the Minimum Description Length (MDL) principle. After compressing the graph, substructures which are more frequent have a shorter Description Length (DL): by implication, substructures with a high DL are more anomalous. [10] uses this principle as the basis of a heuristic to detect anomalies. Each substructure is evaluated on the basis of *how much* and *how soon* it is compressed by Subdue. GBAD [5] is a suite of three anomaly detection algorithms, also based on Subdue: GBAD-MDL detects anomalous modifications; GBAD-P (Probability) detects anomalous insertions; GBAD-MPS (Maximum Partial Substructure) detects anomalous deletions. The above algorithms operate on unweighted graphs with discrete vertex and edge labels. If the graph contains continuous (numeric) attributes, these cannot be incorporated and are therefore ignored. We can improve the accuracy of anomaly detection by incorporating the numeric attribute values.

A few algorithms take numeric edge weights into consideration. OddBall [1] operates on *egonets* but ignores vertex and edge labels. OddBall can therefore detect anomalous vertices with respect to their immediate neighbourhood, but cannot detect anomalous substructures. [8] applies three edge-weighting approaches to gSpan for transaction graph mining. The search space for frequent subgraph discovery is reduced, by assuming that high-weighted edges are more significant than low-weighted edges.

[1]Yet Another Graph-based Anomaly Detection Algorithm

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.



(a) Single large graph representing TCP SYN and ICMP PING network traffic, with two Denial of Service (DoS) attacks taking place.

(b) Each subgraph *(i)–(iv)* represents a bank transaction at an automated teller machine.

Figure 1: Detecting (a) anomalous substructures and (b) anomalous subgraphs

[6] extends GBAD-P to detect anomalies in graphs with continuous labels, by including the numeric values into the probability that a particular edge should exist. It is assumed that the values are distributed normally. YAGADA can detect anomalies without making such assumptions about the prior distribution of numeric values.

Two recent papers incorporate numeric labels in a more general way: [11] includes a discretization step before attempting frequent subgraph matching, using one of several unsupervised discretization methods. Seven methods are proposed to estimate the number of bins k. [13] is a brief outline of an extension to Subdue, which incorporates numeric values using frequency histograms. We evaluate these approaches in Sect. 4 and demonstrate that YAGADA outperforms these algorithms for anomaly detection.

The rest of this paper is organised as follows: Sect. 2 provides an overview of our approach to detecting anomalies in structural and numeric data. Sect. 3 presents the details of YAGADA. In Sect. 4, we evaluate YAGADA on a real-world Access Control database and compare YAGADA's performance against the algorithms discussed above. Our conclusions are presented in Sect. 5.

2. DETECTING ANOMALIES IN GRAPHS WITH NUMERIC LABELS

The example graphs in Fig. 1 contain both structural and numeric anomalies. Fig. 1a is a single large graph with discrete labels and numeric edge weights. Fig. 1b is a database of graph transactions with discrete and numeric attributes.

Fig. 1a shows a fragment of traffic in a TCP/IP network. Two types of interchange between the hosts in the network are shown: **ping** is used to test that another host is alive, and **SYN** is used to synchronise the order of TCP packets. SYN is a three-part exchange: after the TCP client sends the SYN request, the server acknowledges (SYNACK) and waits for the client acknowledgement (ACK).

The anomalies in the graph represent two common Denial of Service (DoS) attacks. The first is a SYN flooding attack, where a client starts many SYN exchanges but fails to send the final ACK, forcing the server to hold all its ports open. The second is a ping flooding attack, where the client consumes the server's bandwidth.

To illustrate how Subdue searches for anomalous substructures, we first define some formal properties of graphs:

Definition 1. A graph $G = (V, E)$ consists of a set of vertices (or nodes) V and edges E. $E \subseteq \{(v, w) : v, w \in V \times V\}$. Two vertices v and w are *adjacent* iff $(v, w) \in E$. If the tuple (v, w) is ordered, the graph is said to be *directed*, otherwise it is *undirected*.

Definition 2. A substructure $S = (V^S, E^S)$ is said to be *isomorphic* to G iff $V^S \subseteq V, E^S \subseteq E$. If $\forall (v, w) \in E \setminus E^S : v \notin V^S, w \notin V^S$, the substructure is *unconnected*. An unconnected substructure is also known as a *subgraph*.

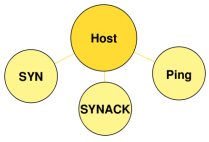
Definition 3. The weight of each edge $e \in E$ is denoted by $\mathcal{W}(e) \in \mathbb{R}$. If $\forall e : \mathcal{W}(e) = 1$, the graph is *unweighted*, otherwise it is *weighted*.

Definition 4. For a graph G containing substructure S, the most frequent substructure is the one that minimises:

$$M(S, G) = DL(G|S) + DL(S) \quad (1)$$

where $DL(G|S)$ is the Description Length of G after compressing it with S and $DL(S)$ is the Description Length of S [4]. This implies that substructures with a high value of $M(S, G)$ are infrequent and therefore anomalous.

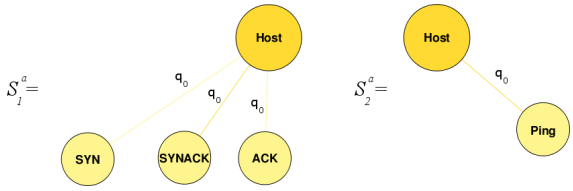
To illustrate structural anomaly detection, let G be the graph in Fig. 1a. Using Subdue's greedy beam search and MDL evaluation (Equation 1), we ascertain that the best frequent substructure S_1 is:



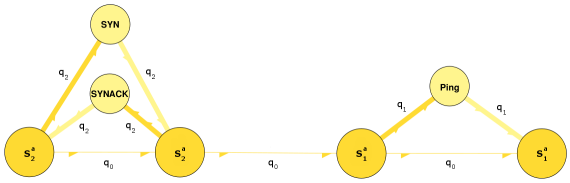
Note that Subdue operates on unweighted graphs, so $\mathcal{W}(e)$ was not taken into consideration. If we compress G with S_1, we obtain a new graph G_1:

In the next iteration, we discover the best frequent substructure $S_2 = \{\ldots, (\text{ACK}, S_1)\}$. We compress G_1 with S_2, and evaluate the DL of the remaining substructures. The substructure with the highest DL is highlighted with a box in Fig. 1a. Subdue has discovered the SYN flooding attack, but not the ping flooding attack. Incorporating the numeric edge weights $\mathcal{W}(e)$ into Subdue does not help, as 3, 5 and 9473 will be considered as distinct discrete values, resulting in substructures which are all equally anomalous.

Our approach solves this problem by making a distinction between numeric values which are "normal" and those which are anomalous, and then incorporating that information into the graph. Stated more formally, we would like to alter G such that $\mathcal{W}(e)$ evaluates to a constant value q_0 for all $\{e \in E : e \text{ is normal}\}$, but evaluates to a set of values $q_i \in \mathbb{R}$ for $\{e \in E : e \text{ is anomalous}\}$. We will defer our discussion of how q_i is to be calculated until Sect. 3. For the purposes of this discussion, let us assume that we can create $G^a = G$, where $\mathcal{W}(e) = 227$ is replaced with q_1, $\mathcal{W}(e) = 9473$ is replaced with q_2 and all other edge weights are replaced with q_0. Now we can apply Subdue to the weighted graph G^a, treating the values q_i as discrete. We discover that the two most frequent substructures S_1^a and S_2^a are:



After compressing G^a with S_1^a and S_2^a, we obtain G_1^a:



All of the normal substructures have been compressed away and only the anomalous substructures are left. As the DL of uncompressed substructures is always higher than the DL of compressed substructures, we can use $M(S_i^a, G^a)$ (Eq. 1) to measure the anomalousness of each substructure S_i^a.

We have demonstrated that replacing numeric edge weights with anomaly scores allows both structural and numeric anomalies to be detected in a weighted graph. We now extend this approach to the more general problem of labelled graphs. We formally define a labelled graph as follows:

Definition 5. Let L^V be the set of vertex labels and L^E be the set of edge labels. If $L^V \cup L^E = \emptyset$, the graph is *unlabelled*, otherwise it is *labelled*.

Definition 6. Let L^D be the set of discrete labels and L^N be the set of numeric labels, such that $L = L^V \cup L^E = L^D \cup L^N$. Let A^D be the set of discrete attribute values and $A^N \subset \mathbb{R}$ be the set of numeric attribute values, such that $A = A^D \cup A^N$. The label-to-value mapping function for vertices is denoted as:

$$\mathcal{L}_V : V \times (L^V \cap L^D) \to A^D,$$
$$\mathcal{L}_V : V \times (L^V \cap L^N) \to A^N$$

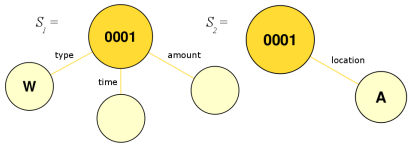
The label-to-value mapping function for edges $\mathcal{L}_E$ is defined in a similar manner. Edge weights can be considered as a special case of edge attributes: $\mathcal{W}(e) = \mathcal{L}_E(e, \text{"}weight\text{"})$.

Fig. 1b shows an example of a labelled graph which contains four graph transactions (subgraphs), $G_{(i)}$–$G_{(iv)}$. Each subgraph represents a cash withdrawal from a bank account. The "hub" vertex for each transaction is labelled with the account number (discrete). The hub is connected to four vertices labelled with the transaction type (discrete; **W** is a withdrawal), the physical location where the transaction took place (discrete; **A** or **B**), the time that the transaction took place (numeric) and the amount (numeric). This graph contains two anomalous subgraphs. Transaction *(ii)* took place at an unusual time of day and was for an unusual amount. Transaction *(iii)* took place in an unusual location.

Let G be the graph in Fig. 1b. We can use Subdue and calculate an anomaly score a_s for each subgraph G_s as:

$$a_s = 1 - \frac{1}{n}\sum_{i=1}^{n}(n-i+1)\left(\frac{DL_{i-1}(G_s) - DL_i(G_s)}{DL_0(G_s)}\right) \quad (2)$$

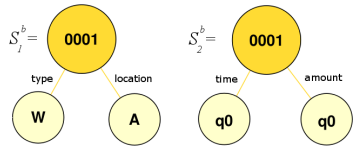
where n is the number of iterations and $DL_i(G_s)$ is the description length of G_s after i iterations [10]. Ignoring numeric labels A^N, we find the best frequent substructures S_1 and S_2:



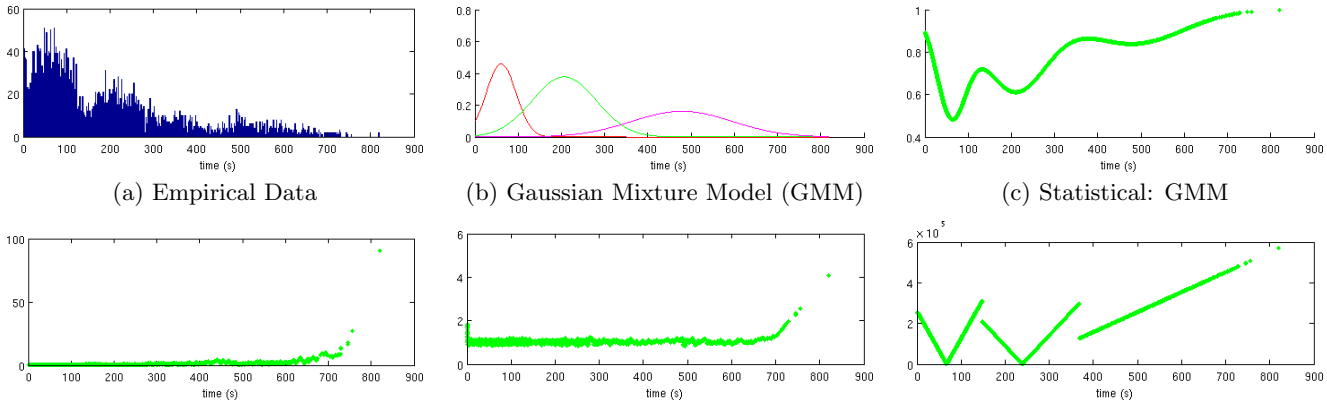
We compress G with S_1 and S_2 and evaluate a_s for all four substructures. As $G_{(i)}$, $G_{(ii)}$ and $G_{(iv)}$ are completely compressed, they have a lower anomaly score. $a_{(iii)}$ is the highest score, so we identify subgraph $G_{(iii)}$ as the most anomalous (highlighted with a box in Fig. 1b).

Once again, Subdue has correctly identified the structural anomaly but does not discover the numeric anomalies. Applying our approach, we map all numeric labels $l \in L^N$ to a constant value q_0 if they are "normal" and with a number q_i indicating the degree of anomalousness otherwise.

Thus we create a new graph G^b where $\mathcal{L}_V(G^b_{(ii)}, \text{"}amount\text{"}) = q_1$, $\mathcal{L}_V(G^b_{(ii)}, \text{"}time\text{"}) = q_2$ and all other numeric attributes take value q_0. Applying Subdue to G^b, we discover the best frequent substructures S_1^b and S_2^b are:



We compress G^b with S_1^b and S_2^b and evaluate a_s as before. Now, only $G^b_{(i)}$ and $G^b_{(iv)}$ are completely compressed and have the lowest anomaly scores $a_{(i)}$ and $a_{(iv)}$. The values for $a_{(ii)}$ and $a_{(iii)}$ are higher, correctly identifying both anomalous transactions.



(a) Empirical Data (b) Gaussian Mixture Model (GMM) (c) Statistical: GMM

(d) Nearest Neighbour-based: k-dist (e) Density-based: LOF (f) Clustering-based: CBLOF

Figure 2: Comparison of methods for detecting numeric outliers

3. YAGADA: AN ALGORITHM TO DETECT STRUCTURAL AND NUMERIC ANOMALIES

In Sect. 2, we described the spirit of our approach and showed how anomaly scores for numeric attributes can be integrated into the graph structure. Now we consider how to calculate the anomaly scores q_i and present our algorithm for graph-based anomaly detection, YAGADA.

There are a number of well-known approaches for detecting outliers in numeric data sets. The simplest is to have a threshold t: if a numeric value $x > t$, it is considered anomalous [2]. This approach has several weaknesses: it requires some method of determining t; it cannot detect anomalies in the middle of the data range, only at the extremes; and the result is a binary—"normal" or "anomalous"—with no measure of the degree of anomalousness.

Fig. 2a shows a dataset $T = \{t_0, t_1, \ldots, t_n\}$, where each $t_i, 0 \leq i \leq n$ represents the elapsed time for a staff member to walk between a specific pair of physical Access Control sensors in an office building (see Sect. 4 for details of the dataset). It is obvious from visual inspection that there is more than one pattern of behaviour. These distinct patterns can be modelled as a mixture of K Gaussian distributions. The probability of observing t_i is:

$$P(t_i) = \sum_{j=1}^{K} \omega_{j,i} \cdot \eta(t_i, \mu_{j,i}, \sigma_{j,i})$$

where K is the number of distributions; $\omega_{j,i}$, $\mu_{j,i}$ and $\sigma_{j,i}$ are the weight, mean and standard deviation of the j^{th} Gaussian in the mixture at time t_i; and η is a Gaussian pdf. The resulting Gaussian Mixture Model (GMM) is shown in Fig. 2b (with $K = 3$). In devising YAGADA, we compared four approaches to computing the anomaly score q_i:

Statistical: If we model the data using a GMM as shown above, $P(t_i \text{ is an outlier}) = 1 - P(t_i)$. The distribution of outlier scores for T is shown in Fig. 2c. Using this approach, we can set an anomaly threshold $0 \leq q_a \leq 1$ and define

$$q_i = \begin{cases} q_0 & \text{if } 1 - P(t_i) < q_a \\ 1 - P(t_i) & \text{otherwise} \end{cases}$$

GMMs can be rapidly computed using Expectation Maximisation. The main difficulty is selecting appropriate parameters: K and q_a must be calculated separately for each numeric label on every vertex and edge.

Nearest Neighbour-based: The above problem can be avoided by using a non-parametric approach, such as k-distances [12]. The distance is calculated from each t_i to its k^{th} nearest neighbour. k is a global smoothing parameter and the selection of k is independent of the statistical distribution of T. Reasonable results are obtained with $k = 1$. Higher values of k reduce the sensitivity of detection. Fig. 2d shows the k-distances for T with $k = 10$. Samples in dense parts of the dataset have very small k-distances (close to 0). Outliers have k-distances which are orders of magnitude larger. Therefore we can define

$$q_i = \begin{cases} q_0 & \text{if k-distance}(t_i) \approx 0 \\ \text{k-distance}(t_i) & \text{otherwise} \end{cases}$$

The general k-distances algorithm has $\mathcal{O}(N^2)$ complexity, but various heuristics have been proposed to reduce this [12]. If T is one-dimensional and has been sorted (as in Fig. 2d), the complexity can be reduced to $\mathcal{O}(N)$.

Density-based: Using GMMs, we detect outliers with respect to the global distribution of T. With k-distances, we detect outliers with respect to the global density of T. If the density of the clusters is not uniform, both approaches will detect sparse clusters as anomalies. Local Outlier Factors (LOF) [3] calculates an anomaly score based on both the distance from and the relative density of the local neighbourhood. Samples belonging to a dense cluster or deep within a sparse cluster have $\text{LOF}(t_i) \approx 1$. Outliers have LOF values several times larger. Using LOF, we define

$$q_i = \begin{cases} q_0 & \text{if LOF}(t_i) \approx 1 \\ \text{LOF}(t_i) & \text{otherwise} \end{cases}$$

LOF(T) is shown in Fig. 2e. Note that LOF has detected anomalies at the low end of the data range, which were not detected using k-distances. While LOF can detect anomalies very accurately, it suffers from $\mathcal{O}(N^2)$ complexity.

Cluster-based: CBLOF (Cluster-Based Local Outlier Factors) [7] is an efficient algorithm which detects outlierness relative to local clusters. Each cluster in the dataset is classified as "large" or "small" using two parameters, α and β. The outlier score $\text{CBLOF}(t_i)$ is the distance to the closest large cluster centroid. So samples in small or sparse clusters are more anomalous than those in big or dense clusters. To use CBLOF, we set an anomaly threshold q_a and define

$$q_i = \begin{cases} q_0 & \text{if CBLOF}(t_i) < q_a \\ \text{CBLOF}(t_i) & \text{otherwise} \end{cases}$$

CBLOF(T) is shown in Fig. 2f, using k-means clustering with $k = 3, \alpha = 0.95, \beta = 2$. The distribution of anomaly

values is similar to Fig. 2c: in fact, CBLOF has similar advantages and disadvantages to GMM. The complexity is generally low (depending on the chosen clustering algorithm), but the optimal values of k, α and β must be determined separately for each numeric label on every vertex and edge.

YAGADA must assign a constant value q_0 to normal values of t_i and an anomaly score q_i to numeric outliers. From our evaluation above, we see that a non-parametric approach is more suitable for unsupervised learning. Although LOF returned the most accurate results, its $\mathcal{O}(N^2)$ complexity made it unsuitable for use with large datasets. We therefore settled on K-Nearest Neighbours (k-distances) as the best trade-off between accuracy and efficiency. k-NN also has the property that it scales easily to multi-dimensional data. The choice of outlier detection algorithm is considered further in Sect. 5. Having chosen the outlier detection method, we can now present the complete algorithm for YAGADA:

Algorithm 1 YAGADA: detect anomalies in labelled graphs

Require: $G = (V, E, L^V, L^E, \mathcal{L}_V, \mathcal{L}_E)$
Return: Set of subgraphs $S = \{s_0, \ldots, s_n\}$; corresponding anomaly scores $A = \{a_0, \ldots, a_n\}$
1: Partition V into subsets $V_i : V = V_0 \cup V_1 \cup \ldots \cup V_{N-1}$, $\forall v, w \in V_i : v$ is the same type as w
2: **for all** $i : 0 \leq i < N$ **do**
3: **for all** $v \in V_i$ **do**
4: **let** $\mathcal{L}_V(v, \text{“anomaly score”}) = \text{Discretize}(v, V_i, \mathcal{L}_V)$
5: **end for**
6: **end for**
7: **let** $L^V = L^V \setminus L^N$
8: Partition E into subsets $E_j : E = E_0 \cup E_1 \cup \ldots \cup E_{M-1}$, $\forall (v, w), (x, y) \in E_j : (v, w)$ is the same type as (x, y)
9: **for all** $j : 0 \leq j < M$ **do**
10: **for all** $e \in E_j$ **do**
11: **let** $\mathcal{L}_E(e, \text{“anomaly score”}) = \text{Discretize}(e, E_i, \mathcal{L}_E)$
12: **end for**
13: **end for**
14: **let** $L^E = L^E \setminus L^N$
15: Execute Subdue(G) for *Limit* iterations
16: **for all** subgraphs $s \subset G$ **do**
17: Calculate anomaly score a_s by Eq. 2
18: **end for**

Algorithm 2 DISCRETIZE: calculate an anomaly score on the numeric attributes of a vertex or edge using k-distances

Require: $v, V, \mathcal{L}, k$
Return: Outlier score q_i
1: **let** $q_0 = 0$
2: **let** $FV = \mathcal{L}(V, L^N)$
3: **let** $fv_v = \mathcal{L}(v, L^N)$
4: **let** $fv_w = \text{k}^{\text{th}}$ nearest neighbour(fv, FV)
5: **let** $q = \text{distance}(fv_v, fv_w)$
6: **if** $q \approx 0$ **then**
7: **return** q_0
8: **else**
9: **return** q
10: **end if**

Algorithm 1 labels vertices with their anomaly score (lines 1–7). Vertices are partitioned so the attributes of vertices of the same type are considered together. The vertex type may be defined by its label (Fig. 1a) or by the type of edge it is connected to (Fig. 1b). The procedure is repeated for edges (lines 8–14). We then use Subdue to detect frequent substructures (line 15) and calculate the anomaly scores for the compressed graph (line 17). Algorithm 2 details the step of calculating the anomaly score from the numeric attributes. FV is a set of feature vectors created from the attributes of the vertices (or edges) under consideration (line 2). The dimensionality of FV is $|L^V \cap L^N|$ for vertices and $|L^E \cap L^N|$ for edges. fv_v, the feature vector for the current vertex, is compared to FV to find the k-distance to its k^{th} nearest neighbour (lines 4–5). If the k-distance is normal, we return the constant q_0; otherwise, we use the k-distance as the measure of outlierness (lines 6–10).

Although a formal analysis of complexity is beyond the scope of this paper, we will comment briefly: we optimised the algorithm by pre-sorting T (average $\mathcal{O}(N \log N)$; worst case $\mathcal{O}(N^2)$), which allowed us to calculate k-distances in linear time. We also maintained a cache of k-distances for all attribute-value pairs.

4. EXPERIMENTS

In this section, we compare the performance of YAGADA to standard Subdue (ignoring the numeric attributes) and to three of the discretization approaches proposed for frequent subgraph discovery [11, 13]. Subdue discovers structural anomalies only and gives a baseline for comparison. The discretization approaches consider numeric attributes by sorting into ten bins of (a) Equal Frequency and (b) Equal Width, and (c) k-means clustering with $k = 10$. We compared to YAGADA using k-distances with $k = 10$.

The dataset for our experiments is from an Access Control system in a typical office building. There are 256 staff members and 155,100 Access Card transactions, representing all movements within the building over the course of 15 months. We reorganised the data into approximately 40,000 graph transactions (subgraphs), where each transaction represents the path taken by a specific user on a specific day. Each vertex represents a door sensor, and directed edges represent movements. We added a numeric attribute to each edge (v, w) for the elapsed time in seconds between presenting an Access Card at v and presenting it at w.

We applied each of the above approaches and used Eq. 2 to calculate an anomaly score $a = [0, 1]$ for every subgraph. Fig. 3 shows the results as a cumulative probability distribution. Most of the data—the “normal” part—has $a \lesssim 0.33$ (not shown): this represents the boundary between anomalous and normal transactions. Fig. 3 shows the relative performance of the five anomaly detection approaches: Subdue detects very few anomalies; KMeans performs somewhat better; EqualFreq and EqualWidth perform similarly, and YAGADA detects the most anomalies. At the upper part of the range ($a > 0.8$), YAGADA detects almost four times as many anomalies as the next best algorithm. In the mid-part of the range ($a > 0.65$), YAGADA detects approximately twice as many anomalies. This difference decreases with decreasing values of a, until at the boundary between normal and anomalous values ($a \approx 0.33$), the algorithms agree. This behaviour shows that while all algorithms may be discovering the same anomalous patterns, YAGADA detects them with much higher confidence, and provides much better discrimination between slightly anomalous and very anomalous patterns. Table 1 bears this out: of the top 10 anomalies discovered by any of the other four approaches, YAGADA detects all of them and in most cases assigns an anomaly score

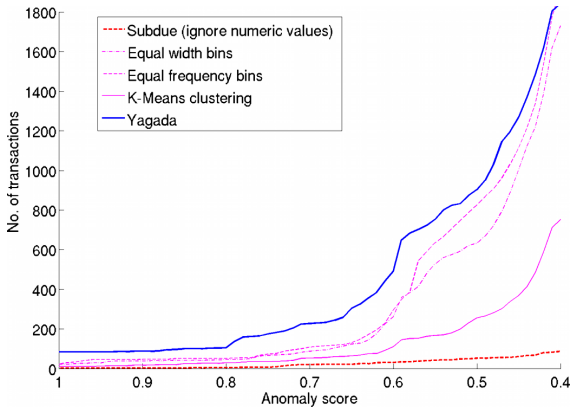


Figure 3: Cumulative Frequency Distribution showing transactions discovered as anomalous ($a \geq 0.4$)

Yagada	EqualFreq	EqualWidth	KMeans	Subdue
1.00	0.42	0.93	0.72	0.34
1.00	0.78	0.77	0.49	0.39
1.00	0.78	0.68	0.49	0.39
1.00	0.78	0.68	0.49	0.39
1.00	0.51	0.93	0.49	0.35
1.00	0.62	0.66	0.62	0.27
1.00	0.62	0.66	0.62	0.23
0.80	0.75	**0.81**	0.68	0.28
0.79	0.58	0.77	0.64	0.36
0.79	**0.96**	0.68	0.49	0.39

Table 1: Comparison of Top 10 Anomalies discovered by EqualFreq, EqualWidth and KMeans

which is greater than or similar to that produced by the best-performing of the alternative algorithms.

We also performed a qualitative analysis to confirm that the anomalies discovered by YAGADA are meaningful and useful. We examined a random sampling of anomalous patterns and checked that they were meaningful, such as failing to swipe at one sensor along a path. The patterns that YAGADA discovered with highest confidence were those which combined unusual paths with unusual timings.

We conclude that YAGADA is able to detect all of the anomalies which can be discovered using any of the alternative approaches, usually with greater confidence. YAGADA outperforms the other approaches as it uncovers a larger number of meaningful anomalies at the higher values of a.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we presented YAGADA, an algorithm for detecting anomalies in graphs that consist of both structural data and numeric attributes. The novelty of our method is to replace numeric values in the graph with a constant q_0 if the value is normal, and an anomaly score q_i otherwise. When we subsequently search the graph for frequent substructures, q_0 will be incorporated into frequent patterns. The values q_i are infrequent and therefore substructures which contain $q_i \neq q_0$ are more anomalous. Our experiments demonstrate that calculating an outlier value for numeric attributes outperforms other numeric discretization methods for anomalous substructure discovery.

We evaluated several approaches to numeric anomaly detection and concluded that LOF and k-Nearest Neighbour (k-distances) were most suitable. These algorithms return an outlierness score close to some constant (0 or 1) for normal values, and much larger scores for anomalous values. k-distances was chosen over LOF due to its lower computational complexity. If we can make some assumptions about the data, it may be possible to devise a hybrid version of LOF. This would be an application-specific version of YAGADA, which trades off generality for improved performance and accuracy. As discretization is performed by a separate function, YAGADA lends itself to this kind of customization.

YAGADA combines the values for all numeric labels on a vertex or edge into a single feature vector. In our experiments, we had only one feature, elapsed time. YAGADA allows that we extend this into a 2-dimensional vector (*e.g.*, elapsed time and time of day) or multi-dimensional. As the number of dimensions is increased, so is the complexity of computing the k-distance between samples. The most efficient variant of k-distances is $\mathcal{O}(c^{\delta} + N)$, *i.e.* linear complexity with regard to the data size but exponential with regard to the number of dimensions δ. In our future work, we will evaluate whether k-distances is the most appropriate outlierness measure for multi-dimensional numeric data.

The algorithm demonstrated here is for use on static graphs, so it could be used for forensic analysis of graph transaction databases. We plan to extend the approach for on-line detection of anomalies using dynamic graphs.

6. REFERENCES

[1] L. Akoglu, M. McGlohon, and C. Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *Advances in Knowledge Discovery and Data Mining*. Springer, 2010.

[2] R. P. Biuk-Aghai, Y.-W. Si, S. Fong, and P.-F. Yan. Security in physical environments: Algorithms and system for automated detection of suspicious activity. In *Workshop on Behavior Informatics 2010*. Springer, 2010.

[3] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, 2000.

[4] D. J. Cook and L. B. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15:32–41, March 2000.

[5] W. Eberle and L. Holder. Mining for insider threats in business transactions and processes. In *Computational Int. and Data Mining, 2009*, pages 163–170, March 2009.

[6] W. Eberle and L. B. Holder. Discovering anomalies to multiple normative patterns in structural and numeric data. In *22nd International FLAIRS Conference*, 2009.

[7] Z. He, X. Xu, and S. Deng. Discovering cluster-based local outliers. *Pattern Recog. Letters*, 24(9–10):1641–1650, 2003.

[8] C. Jiang, F. Coenen, and M. Zito. Frequent sub-graph mining on edge weighted graphs. In *12th Int. Conf. on Data Warehousing and Knowledge Discovery*. Springer, 2010.

[9] F. Mansman, L. Meier, and D. A. Keim. Visualization of host behavior for network security. In *VIZSEC 2007*, Mathematics and Visualization. Springer, 2008.

[10] C. C. Noble and D. J. Cook. Graph-based anomaly detection. In *9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2003.

[11] G. Perez, I. Olmos, and J. A. Gonzalez. Subgraph isomorphism detection with support for continuous labels. In *23rd International FLAIRS Conference*, 2010.

[12] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. *SIGMOD Rec.*, 29:427–438, May 2000.

[13] O. E. Romero, J. A. Gonzalez, and L. B. Holder. Handling of numeric ranges for graph-based knowledge discovery. In *23rd International FLAIRS Conference*, 2010.

[14] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining*, ICDM '02, pages 721–724. IEEE, 2002.