

Graph-Based Anomaly Detection

Caleb C. Noble

Department of Computer Science Engineering
250 Nedderman Hall
University of Texas at Arlington
Arlington, TX 76019
(817)272-5459
noble@cse.uta.edu

Diane J. Cook

Department of Computer Science Engineering
303 Nedderman Hall
University of Texas at Arlington
Arlington, TX 76019
(817) 272-3606
cook@cse.uta.edu

ABSTRACT

Anomaly detection is an area that has received much attention in recent years. It has a wide variety of applications, including fraud detection and network intrusion detection. A good deal of research has been performed in this area, often using strings or attribute-value data as the medium from which anomalies are to be extracted. Little work, however, has focused on anomaly detection in graph-based data. In this paper, we introduce two techniques for graph-based anomaly detection. In addition, we introduce a new method for calculating the regularity of a graph, with applications to anomaly detection. We hypothesize that these methods will prove useful both for finding anomalies, and for determining the likelihood of successful anomaly detection within graph-based data. We provide experimental results using both real-world network intrusion data and artificially-created data.

Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications - *Data Mining*.

Keywords

Data mining, anomaly detection, graph regularity.

1. INTRODUCTION

In the field of data mining, there is a growing need for robust, reliable anomaly detection systems. Although research has been done in this area, little of it has focused on graph-based data. In this paper, we introduce two methods for graph-based anomaly detection that have been implemented using the Subdue system. The first, anomalous substructure detection, looks for specific, unusual substructures within a graph. In the second method, anomalous subgraph detection, the graph is partitioned into distinct sets of vertices (subgraphs), each of which is tested against the others for unusual patterns. In addition, we introduce a measure of graph regularity called conditional substructure entropy, which defines the number of bits needed to describe an

arbitrary substructure's surroundings. We report experimental results obtained using the 1999 KDD Cup network intrusion dataset, as well as artificially-produced data.

2. BACKGROUND ON SUBDUE

The methods for graph-based anomaly detection presented in this paper are part of ongoing research involving the Subdue system [1]. At its core, Subdue is an algorithm for detecting repetitive patterns (substructures) within graphs. For the purposes of this paper, a *graph* consists of a set of vertices and a set of edges, which may be directed or undirected. Furthermore, each vertex and edge contains a *label* to identify its type, which need not be unique. A *substructure* is a connected subgraph of the overall graph. Subdue keeps an ordered list of discovered substructures called the parent list; at the beginning, this list simply holds 1-vertex substructures for each unique vertex label. Subdue repeatedly removes all the substructures from the parent list, generates and evaluates their extensions, and inserts the extensions onto the list. An *extension* of a substructure is generated by adding either a new vertex (and its corresponding edge), or just a single edge within the substructure. As new substructures are being generated, a second list is maintained holding the best substructures discovered so far. When this process is finished, the substructure with the top value is reported, and possibly used to compress the graph before the next iteration begins. *Compressing* the graph refers to replacing each instance of the substructure with a new vertex representing that substructure. Each substructure is evaluated using the Minimum Description Length heuristic [5]. The minimum description length (or simply "description length") is the lowest number of bits needed to encode a piece of data; Subdue contains an algorithm that will approximate this value for any given graph. Using this heuristic, we consider the best substructure to be the one that minimizes the following value:

$$F1(S, G) = DL(G | S) + DL(S)$$

where G is the entire graph, S is the substructure, $DL(G|S)$ is the description length of G after compressing it using S , and $DL(S)$ is the description length of the substructure.

Here is a simple example of how Subdue works. Suppose that we begin with the graph shown in Figure 1.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGKDD '03, August 24-27, 2003, Washington, DC, USA.

Copyright 2003 ACM 1-58113-737-0/03/0008...\$5.00.

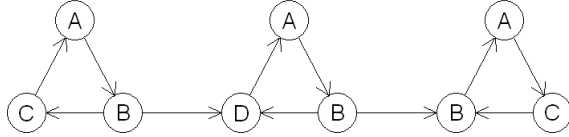


Figure 1. Example graph.

Notice that the substructure $A \rightarrow B$ appears twice. Subdue will generate and evaluate all substructures, and this substructure will be ranked as the best. If another iteration will be run, then Subdue will first compress the graph by replacing the instances of this substructure with a new vertex. As with many algorithms involving graphs, the time complexity of Subdue is exponential in the worst case, but can be reduced to polynomial in practice [1].

3. ANOMALY DETECTION TECHNIQUES

Although much research has been done in the area of anomaly detection, it remains difficult to give a general, formal definition of what an anomaly is. For the purposes of this paper, we will be using the intuitive notion of an anomaly as *a surprising or unusual occurrence*. With this in mind, we introduce two techniques for graph-based anomaly detection using Subdue.

3.1 Anomalous Substructure Detection

This first approach is the simpler of the two, and it is also more general. The objective of anomalous substructure detection is to examine an entire graph, and to report unusual substructures contained within it. This sounds simple, but there are some subtleties involved. For example, it is not enough to simply look for substructures occurring infrequently, since very large substructures are expected to occur infrequently. (For example, if we consider the entire graph as a substructure, it cannot occur more than once.) Below, we introduce a method that circumvents this and other problems, using a variant of the MDL principle.

As discussed earlier, the Subdue system is essentially a mechanism for discovering patterns within graphs. The "patterns" in this case are substructures that produce low values of the quantity $F1(S, G)$. The key to our approach is that an anomaly can be thought of as the "opposite" of a pattern -- just as patterns occur frequently in a graph, anomalies occur infrequently. So one possible method for detecting anomalous substructures is to simply invert the measure, and flag substructures producing *high* values of the quantity $F1(S, G)$.

Cases in which very small (single vertex) substructures and very large ($S=G$) substructures are considered create problems for this approach. A heuristic can be used to remove these problems. We define this heuristic as follows:

$$F2(S, G) = \text{Size}(S) * \text{Instances}(S, G)$$

where $\text{Size}(S)$ is the number of vertices in S , and $\text{Instances}(S, G)$ is the number of times that S appears in the graph G . This function serves as an approximate inverse of $F1$; in other words, as $F1(S, G)$ increases, $F2(S, G)$ tends to decrease. This means that we can use $F2$ instead of $F1$ for anomaly detection -- we consider substructures to be anomalous if they produce *low* values of $F2(S, G)$. This removes the problems associated with

very large and very small substructures. Large substructures (e.g., the entire graph) will not be flagged as anomalous since $\text{Size}(S)$ will be very high, and single-vertex substructures will only be considered anomalous if they do not appear very often.

Suppose, as before, that we begin with the graph in Figure 1. The most anomalous substructure in this graph is D . Its $F2$ value is $\text{Size}(S) * \text{Instances}(S, G) = 1 * 1 = 1$, which is the smallest possible. Several 2-vertex substructures have an $F2$ value of 2 (2 vertices * 1 instance); among them are: $A \rightarrow C$ and $D \rightarrow A$. The least anomalous substructure is the entire graph; it has 9 vertices and 1 instance, so its $F2$ value is $9 * 1 = 9$.

It is important to realize that this measure is biased toward discovering very small substructures. This is because larger substructures are *expected* to occur only a few times; the smaller the substructure, the less likely it is to be rare.

3.2 Anomalous Subgraph Detection

The first method is well-suited for detecting specific, unusual substructures anywhere within a graph. In some situations, though, it could prove useful to partition the graph into distinct, separate structures (subgraphs), and determine how anomalous each subgraph is compared to the others.

One immediate application of this method is the analysis of data represented using a collection of (attribute, value) pairs. In a graphical representation, each record has a "hub" vertex representing the record itself, along with a vertex for each attribute, each of which is connected to the hub. We can then represent the entire table by combining all the records into a single graph. To find anomalous database records, we would be interested in determining how unusual each of the separate star configurations was.

Application of anomalous subgraph detection to attribute-value databases is straightforward, but many other classes of databases can benefit from this technique; it can be applied to any graph in which the vertices can be grouped in a meaningful way. An example is web clickstream data. This type of data has a natural graph-based representation: vertices correspond to web pages, and directed edges correspond to navigational links selected by the user. Furthermore, the vertices can be grouped into subgraphs in a meaningful way -- organized by user. Subgraph A would contain all the clickstream data from user A, subgraph B would contain data from user B, and so forth. By performing anomalous subgraph detection on the overall graph, we would then be testing each user for unusual web-navigation patterns.

Next we describe a method for anomalous subgraph detection using Subdue. First, some background is necessary. Subdue can be set to run multiple iterations on a single graph. After each iteration, the graph is compressed using the discovered substructure; in other words, every instance of the substructure is replaced by a single vertex. The next iteration of Subdue will then operate on the newly compressed graph. This multiple-iteration capability is used in our approach. It is important to realize that the "best" substructures will be discovered in the first several iterations, while later substructures will become less and less valuable (i.e., less common). For our purposes, we are

assuming that Subdue halts once the graph contains no substructure with more than one instance.


The rationale for our method lies in the idea that *subgraphs containing many common substructures are generally less anomalous than subgraphs with few common substructures*. This is related to the underlying idea behind anomalous substructure detection – that common substructures are, in a loose sense, the “opposite” of anomalous substructures. On each iteration, Subdue discovers the “best” substructure (in the MDL sense), and then compresses the graph with it. It stands to reason, then, that anomalous subgraphs tend to experience less compression than other subgraphs, since they contain few common patterns.

We define our subgraph anomaly measure as follows. To each subgraph, we assign a value A ; the higher A is, the more anomalous the subgraph. A is given by the

formula $A = 1 - \frac{1}{n} \sum_{i=1}^n (n - i + 1) * c_i$, where n is the number of iterations and c_i is the percentage of the subgraph that is compressed away on the i^{th} iteration. c_i is more rigorously defined as $\frac{DL_{i-1}(G) - DL_i(G)}{DL_0(G)}$, where $DL_j(G)$ is the description length of the subgraph after j iterations.

Some explanation of the above formula for A is in order. The idea is that all subgraphs begin with an A -value of 1 (i.e., completely anomalous), and the values drop off as portions of the subgraphs are compressed away during the iterations. The c_i term will vary from 0 to 1; a value of 0 means that the subgraph was not changed on the i^{th} iteration, while a value of 1 means that the entire subgraph was compressed away. The $(n - i + 1)$ term will vary from n to 1 as i increases; this causes A to drop off more sharply for compressions that occur early on. The $(1/n)$ term guarantees that the final value will be between 0 and 1, since the maximum possible value for the summation is n . (This would occur if the entire subgraph was compressed away on the first iteration.)

Again, consider the example graph in Figure 1. Suppose that the three triangles represent three separate subgraphs. (This is acceptable, even though edges connect the subgraphs; the edges running between are not considered to be part of any subgraph.)

Recall that after one iteration, the substructure  is ranked highest, and will be used to compress the entire graph. If this is the only iteration under consideration, then we would consider the third subgraph to be the most anomalous; it will not be compressed at all, whereas two of the three vertices will be compressed away in each of the first two subgraphs.

4. CONDITIONAL SUBSTRUCTURE ENTROPY

As noted in [3], an important consideration in any anomaly-detection system is the regularity of the data. “Regularity” can be thought of as a synonym for “predictability”; generally, the more predictable the data, the easier it is to detect anomalies. Furthermore, anomaly-detection systems that are configured using training data may perform poorly on data with a different amount of regularity. A good deal of research has been done in

the area of regularity measures (e.g., [2]), but little work has been focused on graph-based data. Here, we define a measure for the regularity of graphs, which has been implemented using Subdue. (Note: In the following discussion, we use the term *entropy* instead of *regularity*. The two terms are opposites – the higher the entropy, the lower the regularity.)

The concept of entropy is well-known. Broadly speaking, entropy measures the number of bits needed to describe a particular occurrence or event. Similarly, *conditional* entropy measures the amount of information needed to describe an event, *given* that some other event is known to have occurred. Here we let X represent the set of possible events and Y represent the set of *prior* events (one of which is known to have occurred). We then say that the conditional entropy of X given Y is

$$H(X | Y) = - \sum_{y \in Y} \sum_{x \in X} P(y) * P(x | y) * \log(P(x | y))$$

where $P(y)$ is the probability that event y occurred, $P(x|y)$ is the probability of event x given event y , and $\log(0)$ is understood to be 0.

This value measures how well an event from Y can predict an event from X . Suppose that for any given $y \in Y$, we can always predict with certainty which event from X will occur. Then $P(x|y)$ will be either 0 or 1 for all $x \in X$, $y \in Y$, and $H(X|Y)$ will be zero. On the other hand, suppose that knowing the event from Y tells us nothing about the event from X – i.e., X and Y are not correlated. Then $P(x|y) = P(x)$ for all $x \in X$, $y \in Y$, and $H(X|Y)$ will degenerate to $H(X)$, the (unconditional) entropy for the set X .

Consider the application of conditional entropy to the domain of strings. A natural use of this concept is to determine the conditional entropy of a character, given a certain number (n) of previous characters. In other words, the conditional entropy answers the question: “Given a certain number of characters in a sequence, how many bits are needed to describe the next character?” Let us consider the example string “abcaabcc”, and suppose that $n = 3$. Then $X = \{‘a’, ‘b’, ‘c’\}$, and $Y = \{‘abc’, ‘bca’, ‘caa’, ‘aab’, ‘bcc’\}$. $P(‘abc’)$ equals $2/6$, since 2 of the 6 3-character substrings are “abc”. $P(‘a’ | ‘abc’)$ = $1/2$, since ‘a’ appears after “abc” 1 out of 2 times.

Our definition of conditional substructure entropy follows directly from the above approach for strings. We are now answering the question: “Given an arbitrary n -vertex substructure, how many bits are needed to describe its *surroundings*?” By “surroundings,” we are referring to the edges and vertices adjacent to the substructure. The surroundings can be thought of as a set of *extensions* to the substructure; we define an extension of a substructure to be the addition of either a single vertex (along with the edge connecting it to the substructure), or a single edge within the substructure. Y is defined to contain all n -vertex substructures within the graph, and X contains all extensions of the substructures in Y . For a given substructure $y \in Y$, $P(y)$ is defined as the number of instances of y in G , divided by the total number of instances of all n -vertex substructures. For particular substructures $x \in X$, $y \in Y$, we define $P(x|y)$ to represent the percentage of instances of y that extend to an instance of x .

There is one complication: we cannot use the above formula $H(X|Y)$ exactly as given above. In that definition, it is assumed

that *exactly one* event in the set X has occurred. In our case, however, multiple “events” may occur (since a substructure can have more than one extension), and we want to measure the bits needed to describe *which events occurred and which ones did not occur*. To account for this, $H(X|Y)$ must be changed to

$$H(X|Y) = \sum_{y \in Y} \sum_{x \in X} P(y) * (P(x|y) * \log(P(x|y)) + ((1 - P(x|y)) * \log(1 - P(x|y))))$$

With this revision, the definition of conditional substructure entropy is complete.

As an example, consider the graph in Figure 2.

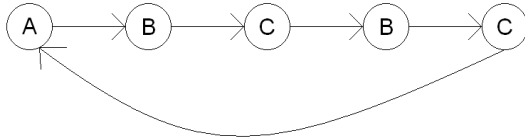
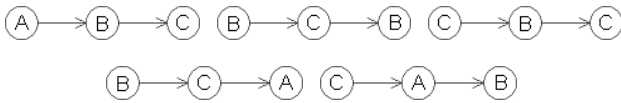


Figure 2. Example graph.

For the set Y , let us choose a substructure size of $n = 2$. Then Y will contain these four substructures:



X will contain all extensions of the substructures in Y :



Suppose that y is $(B) \rightarrow (C)$ and that x is $(B) \rightarrow (C) \rightarrow (B)$. Then $P(x|y) = 1/2$, since one of the two instances of y can be extended to produce x .

5. EXPERIMENTAL RESULTS

5.1 Anomaly Detection

We tested our anomaly-detection methods using the 1999 KDD Cup network intrusion dataset [6]. The data consists of connection records, each of which is labeled as “normal” or as one of 37 different attack types. Each record contains 41 features describing the connection (duration, protocol type, number of data bytes, etc.); some of these features are continuous, others discrete. In the original competition, the dataset was split into two sections: the training data and the test data. Participants were able to train their detectors with the training data, and were then judged based on their performance on the test data.

Since our approaches involve unsupervised learning instead of supervised learning (i.e., no training is involved), we focused solely on the test data. In each test, we sampled a certain number of records from the dataset, and attempted to find the attacks located within the sample. Each individual test involved only one particular attack type; the sampling was essentially random, but controlled so that most selected records (96-98%) were labeled “normal,” while the rest were of the one attack type. Purely random sampling would have worked very poorly, since attacks are quite common in the test data; one of the assumptions of unsupervised anomaly detection is that the anomalous events are

generally rare. In the case of network intrusion data, this is a reasonable assumption; in most situations, attacks would be quite uncommon compared to normal connections. Each sample was performed with replacement, so overlap between samples was possible. We were interested in determining how anomalous the actual attacks were reported to be.

We ran three groups of tests, varying the percentage of attacks and the overall number of records. Each sample was converted into a graph using the star-configuration method described in section 3.2. In the first group, each sampled dataset contained 50 records, 1 of which was an attack. In the second group, samples contained 50 records and 2 attacks; in the third, they contained 100 records and 2 attacks. We only include the first group of results here. Briefly, the results of the second group were significantly worse than those of the first group. This is to be expected; since the attacks in the second group comprised 4% of the records instead of 2%, they would not be considered as anomalous. The results of the third group were slightly better than those of the first group. This suggests that if anomalous events occur at a fixed rate, increasing the amount of available data improves the chances of successful anomaly detection.

For the first method (anomalous substructure detection), we ran one test for each attack type. In each test, we used Subdue to discover the most anomalous substructures within the graph. For the sake of time, we only discovered substructures consisting of 2 or 3 vertices. Also, since we were only interested in the most anomalous substructures, we ignored substructures with a value of $F2 \leq 6$. (The number 6 is somewhat arbitrary, but provided a convenient value for these tests.) We then looked at the fraction of substructures that appeared in an attack record, compared to the total number. We used a *weighted* fraction, to give the most anomalous substructures a higher contribution. This was accomplished by giving each substructure a contribution of $1/F2$. For example, suppose that there are three substructures discovered, with $F2$ values of 2, 3, and 4 respectively, and that the second substructure occurs in an attack record. Then we would say that the attack accounts for $(1/3) / (1/2 + 1/3 + 1/4) = 4/13$ of the discovered anomalies. Clearly, it is desirable for attacks to account for a high percentage of the anomalies.

The results from the first group of tests (50 connection records, 1 attack) are displayed in Figure 3. The numbers displayed are the inverses of the weighted fractions; e.g., if the attack accounts for $1/15$ of the anomalies, then a 15 is displayed. The lower the number, the more anomalous the attack was considered to be.

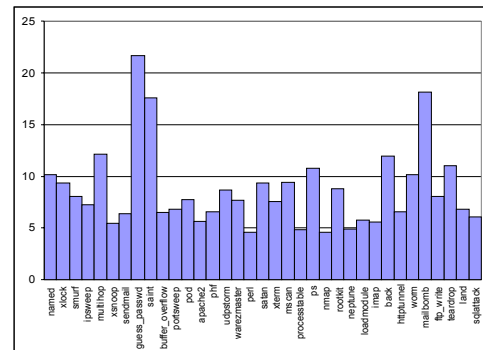


Figure 3. Anomalous substructures; 50 records, 1 attack.

Overall, the results were good. For most types, the attack accounted for at least 1/10 of the discovered anomalies. Since there were 50 records, an average record would only account for 1/50 of the anomalies. It should be noted, however, that two of the attack types (*snmpgetattack* and *snmpguess*) performed so poorly that they could not be shown on the graph; their values were about 2211 and 126, respectively. Interestingly, these two types caused very poor results in the anomalous subgraph detection tests as well. Excluding these two types, the average value was about 8.64.

For the second approach (anomalous subgraph detection), we ran 10 separate tests for each attack type. As described earlier, each test sample consisted of 50 (or 100) records, with one or two of the records being attacks. For each sample, we used anomalous subgraph detection to rank the connection records from 1 to 50 (or 100), with 1 being the most anomalous. Again, only the first group of test results is shown here.

Figure 4 shows results from the first group of tests (50 records, 1 attack). Each number represents an average of the 10 tests on a particular attack type, with the numbers representing the ranking of the single attack record. The lower the number, the more anomalous that attack was considered to be.

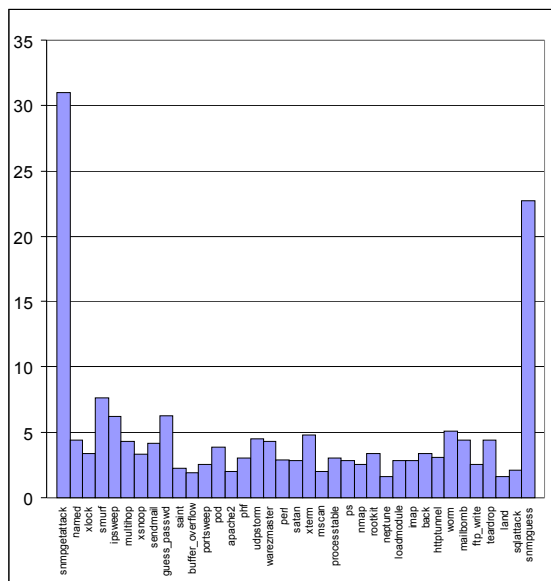


Figure 4. Anomalous subgraphs; 50 records, 1 attack.

As can be seen, the results are reasonably good; most attack types had an average ranking below 5. Only two attack types (*snmpgetattack* and *snmpguess*) had an average ranking of more than 10. The overall average was about 4.75.

5.2 Conditional Substructure Entropy

To test our definition of conditional substructure entropy, we artificially generated a number of graphs and used Subdue to produce conditional entropy values for the graphs. Each graph contained 96 vertices and 96 edges. In each graph, patterns containing two vertices connected with an edge, three vertices in a triangle, or four vertices in a square were inserted. The rest of the vertices and edges were then added with randomly-selected

labels (out of several possibilities) and randomly-selected edge endpoints. The following features were varied: 1. the number of vertices in the inserted pattern; 2. the number of labels in the graph; and 3. the number of inserted patterns. (The second factor specifies how many possibilities exist for vertex and edge labels.) Also, the value of n (size of substructures used to calculate the conditional entropy) was varied, producing several different measures for any given graph. For each combination of factor 1, factor 2, and value of n , we plotted the conditional entropy values vs. the number of patterns inserted in the graph. Ideally, the values would fall off smoothly as more patterns were inserted into a graph. In each chart, the number of inserted patterns increases from left to right; the leftmost data point represents a graph with no instances of the pattern, and the rightmost point indicates a graph consisting entirely of the pattern.

The results varied widely depending on the combination of factors. For example, Figure 5 shows the results for graphs with 3-vertex patterns, 6 labels, and a value of $n = 2$.

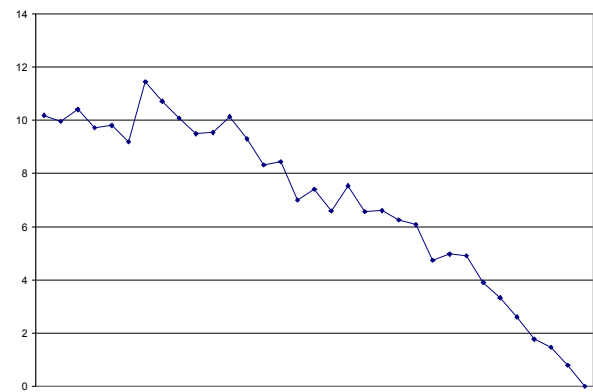


Figure 5. Conditional substructure entropy; 3-vertex patterns, 6 labels, $n = 2$.

These charts display the desired appearance; the values fall off reasonably smoothly as patterns are added.

The results of other feature combinations were not as good. As an example, Figure 6 shows the conditional entropy values for graphs containing 2-vertex patterns and 10 labels, with $n = 2$.

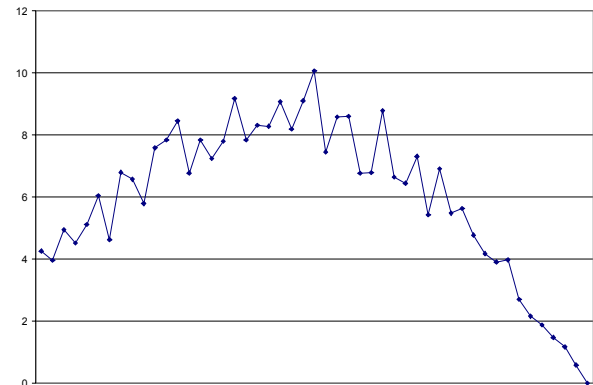


Figure 6. Conditional substructure entropy; 2-vertex patterns, 10 labels, $n = 2$.

This hill-like shape appears in several of the conditional entropy charts. This is due to the fact that for graphs on the left side (those with few inserted patterns), many 2-vertex substructures appear only a few times within the graph. When the average number of instances is very low, the conditional entropy will always be underestimated. This effect is well-known when estimating the (unconditional) entropy of a data set; see, for example, [4]. As our results illustrate, this potential underestimation is important to consider when estimating conditional entropy as well.

5.3 Applications of Graph Regularity to Anomaly Detection

Finally, we tested our hypothesis that the level of regularity in a graph affects the ability to perform anomaly detection on the data. We again used the 1999 KDD Cup dataset. For each attack type, we tested 11 samples, varying the amount of data regularity. Each sample contained 51 “normal” records and one attack record, similarly to how the data was sampled for the anomaly detection tests. The difference was that in each of the 11 samples, a certain number of the “normal” records were identical, i.e., were actually the same record. In each case, the first normal record selected was repeated anywhere from 0 to 50 times, progressing by 5 from test to test. We then used anomalous subgraph detection to rank the single attack record from 1 (most anomalous) to 52 (least anomalous), as described in section 5.1. These rankings were then averaged across all 37 attack types. Figure 7 displays the average rankings vs. the number of repeated “normal” records; the regularity of the data increases from left to right.

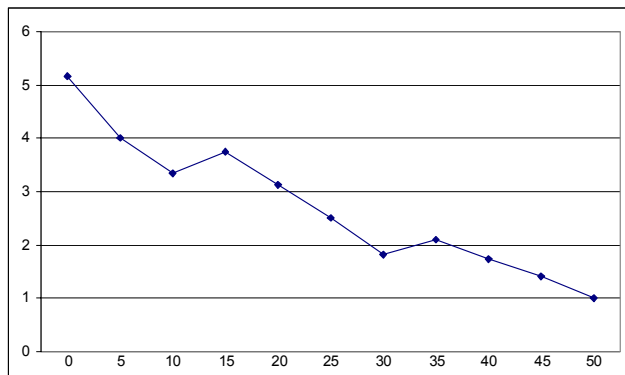


Figure 7. Ranking of attacks as regularity increases.

As can be seen, the average rankings of the anomalous records drop fairly smoothly as the regularity of the normal records is increased. This supports our hypothesis that the regularity of a graph affects the ability to detect anomalies within it.

6. CONCLUSIONS

Graph-based anomaly detection is a promising area that has received little attention. In this paper, we have defined two methods for detecting unusual patterns within graph-based data. We have also introduced a measure for calculating the regularity of a graph, using the concept of conditional entropy. These approaches have been implemented and tested using the Subdue system, with encouraging results. Future work will include theoretical analysis of the abilities and limitations of these methods. We will also be investigating strategies for automatically selecting an optimal substructure size when computing the conditional entropy of a graph, as well as the possibility of considering multiple sizes at once. A detailed examination of the relationship between graph regularity and anomaly detection is also needed.

7. REFERENCES

- [1] Cook, D.J. and Holder, L.B. Graph-Based Data Mining. *IEEE Intelligent Systems*, 15(2), pages 32-41, 2000.
- [2] Lee, W. and Xiang, D. Information-Theoretic Measures for Anomaly Detection. *Proceedings of The 2001 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001.
- [3] Maxion, R.A. and Tan, K.M.C. Benchmarking Anomaly-Based Detection Systems. *International Conference on Dependable Systems and Networks*, pages 623-630, New York, New York; 25-28 June 2000.
- [4] Miller, G.A. *Note on the Bias of Information Estimates. Information Theory in Psychology: Problems and Methods*, Free Press, 1955.
- [5] Rissanen, J. *Stochastic Complexity in Statistical Inquiry*. World Scientific Publishing Company, 1989.
- [6] <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>