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 methods 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.

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 describe two measures of graph regularity, using concepts from information theory. The first measure, substructure entropy, describes the number of bits needed to describe an arbitrary substructure of fixed size. The second measure is conditional substructure entropy; for an arbitrary substructure of fixed size, it describes the number of bits needed to describe the substructure's immediate surroundings. We report experimental results obtained using the 1999 KDD Cup network intrusion dataset, as well as artificially-produced data.

2. BACKGROUND ON SUBDUE

Many types of data contain temporal or spatial relationships between elements that would best be represented in graphical form. For example, using a graph representation of credit card transactions, we could create relationships (edges) between transactions occurring within a mile or within a second of each other. These kinds of relationships could prove useful in certain applications, and would be difficult to represent without a graph-based format. 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.

The methods for graph-based anomaly detection presented in this paper are part of ongoing research involving the Subdue system [1]. This is a graph-based data mining project that has been developed at the University of Texas at Arlington. At its core, Subdue is an algorithm for detecting repetitive patterns (substructures) within graphs. 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 [8]. 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:

$$F1(S,G) = DL(G \mid S) + DL(S)$$

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.

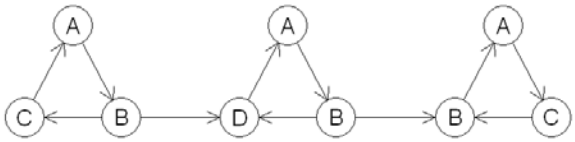
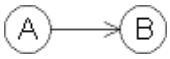


Figure 1. Example graph.

Notice that the substructure



appears twice. Subdue will generate and evaluate all substructures, and this substructure will be ranked as the best. If another iteration is going to be run, then Subdue will replace the instances of this substructure with a new vertex. If we designate the new vertex with the label "S", then the newly compressed graph will be as shown in Figure 2.

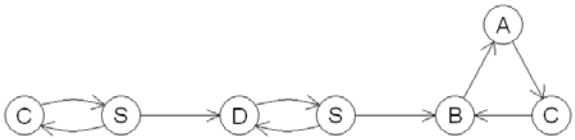


Figure 2. Example graph after compression.

Subdue can then go on to search for another substructure, although in this example, there are no multiple-vertex substructures remaining with more than one instance.

As with many algorithms involving graphs, the time complexity of Subdue is exponential in the worst case. There are a number of parameters that can be set, however, to reduce it to polynomial time. These parameters include the beam width of the ordered list, a limit on the number of substructures expanded, an option to prune substructures with less value than their parent, and others. For details on these parameters, as well as a closer examination of the algorithm itself, see [1]. The basic algorithm has been extended for use in several different ways, including concept learning [3] and clustering [4].

3. TECHNIQUES FOR ANOMALY DETECTION

Although a great deal of 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). There are a couple of problems with this, however. One problem is that if S = G (i.e., we are considering the entire graph as our substructure), then this quantity will be very high -- indeed, higher than the description length of the graph, since

$$F1(S,G) = DL(G \mid S) + DL(S) = DL(G \mid G) + DL(G) \geq DL(G)$$

This is a higher value than would be returned for almost any other substructure in the graph, so the substructure consisting of the entire graph would always be flagged as very anomalous. This would, of course, be useless. A similar problem occurs at the other end of the spectrum, for substructures consisting of a single vertex. In this case, attempting to compress the graph using S would result in no compression at all. DL(G|S) would then equal DL(G), and we would again have

$$F1(S,G) = DL(G \mid S) + DL(S) = DL(G) + DL(S) \geq DL(G)$$

Again, this is a very high value, so single vertices would always be flagged as anomalous, regardless of how many times they appeared in the graph.

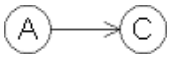
Clearly, using this method directly would be problematic. The basic idea is sound, however, and it turns out that a heuristic can be used to remove the problems mentioned above. We define this heuristic as follows:

$$F2(S,G) = Size(S) * Instances(S,G)$$

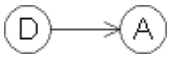
where Size(S) is the number of vertices in S, and Instances(S, G) is the number of times that S appears in the graph G. This function serves as an approximate inverse of F1; in other words, as F1(S, G) increases, F2(S, G) tends to decrease. (We do not provide a formal justification of this heuristic, but it should appear reasonable. F1(S, G) essentially measures how well a substructure compresses a graph, and the amount of compression is closely tied to the substructure's size and its number of instances.) 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 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 this:

Its F2 value is Size(S) * 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:



and



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. Suppose that we have a table of credit card transactions, and we wish to look for unusual or suspicious transactions. In this example, our table contains 5 fields:

amount – amount of the transaction
num_trans – number of transactions on this account within the last 24 hours before the purchase
trans_category – category of business at which the transaction was made (online, department store, etc.)
credit_limit – credit limit on the card
months_active – number of months the card has been active before the transaction

We can represent each transaction record in graphical form using a star configuration, as shown in Figures 3 and 4.

amount	num_trans	trans_category	credit_limit	months_active
350.88	3	online	10,000	23

Figure 3. Example database record.

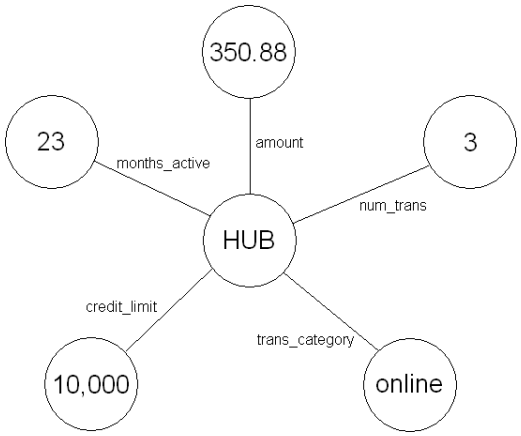


Figure 4. Graphical representation of record.

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. (Note that both vertices and edges are labeled.) 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 links selected by the user to navigate from page to page. 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 sub*structure* 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.

It is not sufficient, however, to simply wait until Subdue is finished, and then check how much each subgraph was compressed. In the later iterations, Subdue will begin finding substructures that only occur a few times, since all the more common patterns have already been discovered. The following example should show why this could cause problems. Suppose that a graph contains two subgraphs that are completely identical, but are very unusual in the overall graph. For most of the iterations, these subgraphs will remain uncompressed, since they do not contain any of the common patterns that Subdue is discovering. Eventually, however, since there is nothing unique about either subgraph, they will both be compressed away completely. Hence, neither subgraph will appear anomalous, even though they are both anomalous in the context of the entire graph. From this example, we see that another factor is needed – *how soon* compression takes place in a subgraph. For example, suppose that after all iterations have completed, subgraphs A and B have both been compressed to half of their original size. If A was compressed on the 1st iteration, and B was compressed on the 50th, then B would be considered more anomalous than A.

To put these concepts together, a measure is needed that considers both *how much* and *how soon* a subgraph is compressed. We define this measure as follows. To each subgraph, we assign a value A; the higher A is, the more anomalous the subgraph. A is given by this 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 graph appears as shown in figure 2. If this is the only iteration under consideration, then we would consider the third subgraph to be the most anomalous; it was not compressed at all, whereas two of the three vertices have been compressed away in the first two subgraphs.

4. MEASURES OF GRAPH REGULARITY

As noted in [6], 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., [5]), but little work has been focused on graph-based data. Here, we define two different measures for the regularity of graphs, both of which have 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.)

4.1 Substructure Entropy

The concept of entropy is well-known; it is covered in many textbooks (e.g., [2]). Broadly speaking, entropy measures the number of bits needed to describe a particular occurrence or event. For a given set of possible events X, the entropy H is given by

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

where P(x) is the probability that x occurred and log(P(x)) is the base 2 logarithm of P(x). For H(X) to be well-defined, log(0) is understood to be 0.

Clearly, the concept of entropy is useful for determining the regularity or predictability of a given data set. All that is needed is a good definition of the set of possible events, X, and the corresponding probabilities P(X). For example, consider the domain of strings. A simple measure of string entropy would be the predictability of an arbitrary character within that string. In this approach, X would be the set of all characters contained in the string. Furthermore, for x ε X, we would define P(x) as the number of occurrences of x in the string, divided by the length of the string. A more sophisticated approach would be to let X be all substrings of a certain length (n). P(x) would then be the number of occurrences of substring x, divided by the total number of substrings of length n. As an example, let us choose a substring length of n = 3. Suppose further that the given string is "abcaabcc". X would then be {"abc", "bca", "caa", aab", "bcc"}, and P("abc") would be 2/6. The denominator is 6 because there are 6 (overlapping) substrings of length 3: "abc", "bca", "caa", aab", "abc", and "bcc".

A similar idea holds for graphs. Our definition of substructure entropy follows directly from the above approach for strings, except that we replace *substring length* with *substructure size* (as measured by the number of vertices). For a given size n and graph G, we define X to be the set of all n-vertex substructures

within G. For a given substructure x ε X, P(x) is defined as the number of instances of x in G, divided by the total number of instances of all n-vertex substructures. With this definition, the substructure entropy of a graph measures the number of bits needed to describe an arbitrary substructure of size n.

A couple of comments about this definition are in order. First, it should be clear that in general, the more regularity or "pattern" a graph contains, the lower its substructure entropy measure will be. As certain substructures occur more and more frequently, the probabilities of these substructures will increase, while the probabilities for other substructures will decrease (assuming a fixed graph size). As is well known, entropy is highest when the probabilities are uniform, and it decreases as the probability distribution becomes less uniform. Second, there is no single entropy measure for a given graph; the value is dependent on the selected substructure size, n. For any given graph, how to choose the "best" value of n is an open question. (In fact, it is not obvious how "best" would be defined in this situation.) Clearly, n = 1 is not the best choice; this would only measure the entropy of vertex labels, without considering relationships between vertices. Very large values of n would, of course, be worthless. Another possibility would be to consider multiple sizes at once (perhaps even all sizes). More research is needed in this area before definite conclusions can be reached.

As an example, consider the directed graph shown in Figure 5.

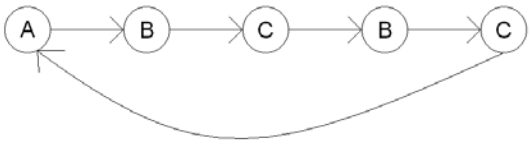
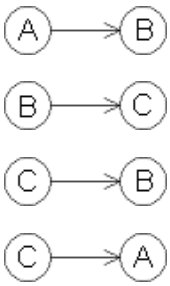


Figure 5. Example graph.

If we choose a substructure size of n = 2, then X will contain the following substructures:



The values of P(x) for these substructures are 1/5, 2/5, 1/5, and 1/5 respectively.

4.2 Conditional Substructure Entropy

A second, related measure can be defined using the concept of *conditional* entropy. Conditional entropy measures the amount of information needed to describe an event, *given* that some other event is known to have occurred. We now must work with two sets instead of one: X, which is again the set of possible events, and Y, which is 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

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

Again, 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 ε 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 ε X, y ε 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 ε X, y ε Y, and H(X|Y) will degenerate to H(X).

In the domain of strings, a natural use for these concepts 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 again 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.

As before, our definition of conditional substructure entropy follows the above ideas closely. It is slightly more complicated, however. 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.

The set Y and its associated probabilities P(Y) are defined just as X and P(X) were defined for substructure entropy; Y contains all n-vertex substructures within the graph. However, some care must be taken in defining X and the associated conditional probabilities, P(X|Y). According to the above discussion, X should contain all possible extensions of all n-vertex substructures. Since an extension (as defined above) may add a new vertex to the substructure or merely a single edge, this means that X should contain all substructures containing n or (n + 1) vertices. For particular substructures x ε X, y ε Y, we define P(x|y) to represent the percentage of instances of y that extend to an instance of x.

There is one more 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

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

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

As an example, let us again use the graph in Figure 6.

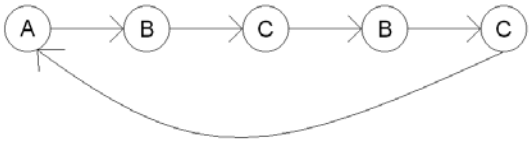
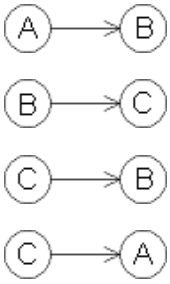
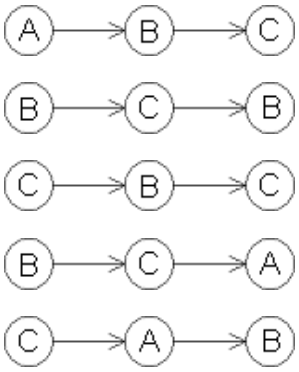


Figure 6. Example graph.

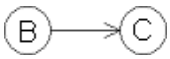
For the set Y, let us choose a substructure size of n = 2. Then Y will contain these substructures:



Earlier, we defined X to contain all substructures containing n or (n + 1) vertices. In this simple example, however, no edge-only extensions exist; in other words, there is no way to extend a given substructure from Y without adding a new vertex. So the only substructures in X that are worth considering are those containing n + 1 = 3 vertices:



Suppose that y is



and that x is

Then P(x|y) = 1/2, since one of the two instances of y will extend to x.

5. EXPERIMENTAL RESULTS

5.1 Anomaly Detection

We tested our anomaly-detection methods using the 1999 KDD Cup network intrusion dataset [9]. 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 approach involves 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. 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. Each sample was converted into a graph using the star-configuration method described in section 3.2.

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 $\frac{1}{F2}$ instead of just 1. 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 $\frac{\frac{1}{3}}{\frac{1}{2}+\frac{1}{3}+\frac{1}{4}} = \frac{4}{13}$ 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 7. 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.

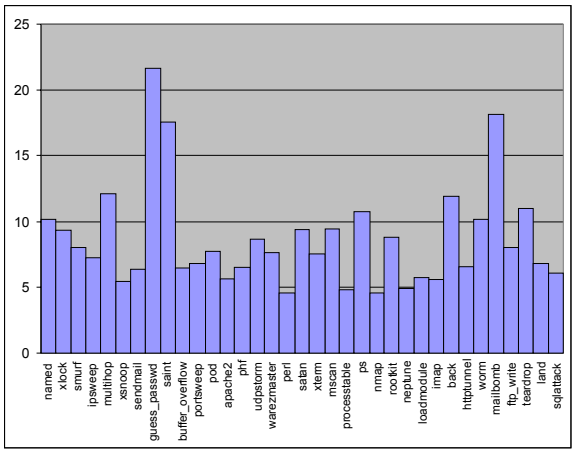


Figure 7. Anomalous substructure detection; 50 records, 1 attack.

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; in other words, the attack accounted for between 1/8 and 1/9 of the discovered anomalies, on average.

The results from the second group of tests (50 records, 2 attacks) are given in Figure 8. The separate fractions of the two attacks have been added, and the inverse of this sum is shown for each attack type.

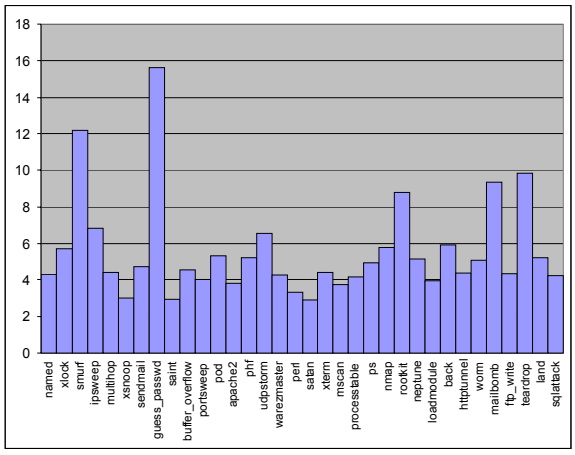


Figure 8. Anomalous substructure detection; 50 records, 2 attacks.

The results were poorer for the second group, which is to be expected; the attacks comprised 4% of the records instead of 2%, so they would not be considered as anomalous. Other things being equal, we would expect the numbers to be cut in half, since we are now considering the contribution of two records instead of one. (If a record accounts for 1/8 of the anomalies, then two of them account for 1/4 – i.e., the denominator is cut in half.) Excluding *snmpgetattack* and *snmpguess* (which had respective values of 456 and 3184), the overall average was about 5.56, significantly higher than half of the first group's average. This demonstrates the principle that anomaly detection systems cannot be fully trusted in a situation where unwanted behavior occurs too frequently to be considered anomalous.

The results of the third group (100 records, 2 attacks) are shown in Figure 9. Again, the contributions of the two attacks were summed, and the inverses of these sums are displayed.

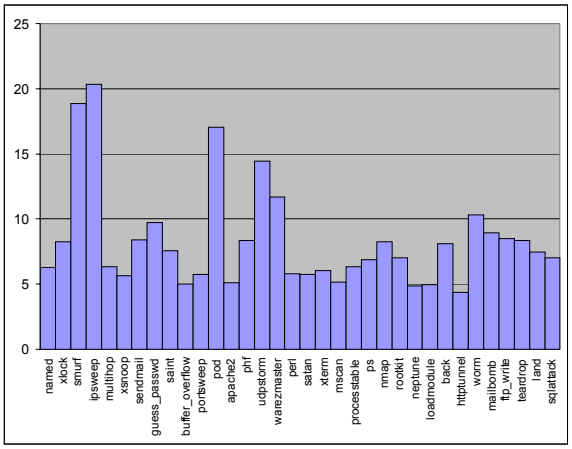


Figure 9. Anomalous substructure detection; 100 records, 2 attacks.

The results for this group were the best of the three, although only slightly better than those of the first group. Other things being equal, the numbers would be expected to be similar to the first group's, since considering 2 records out of 100 is just like considering 1 out of 50. Again excluding *snmpgetattack* and *snmpguess* (which had values of 901 and 710), the average was about 8.33, a slight improvement over the first group's average of 8.64. 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 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.

Figure 10 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.

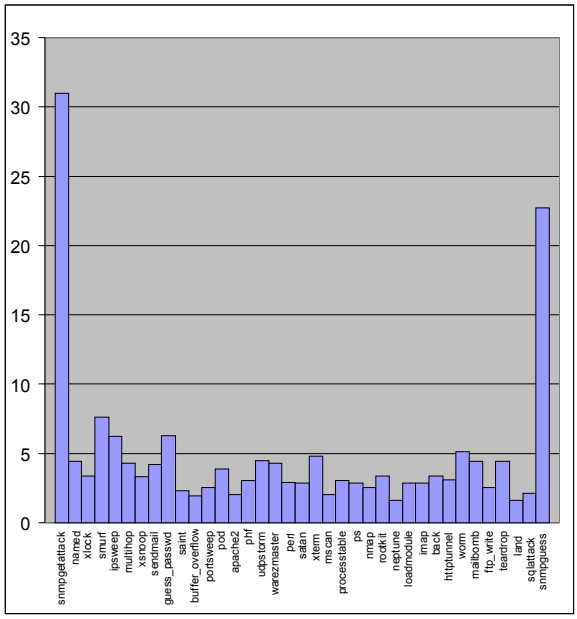


Figure 10. Anomalous subgraph detection; 50 records, 1 attack.

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

The results for the second group (50 records, 2 attacks) are displayed in Figure 11. In this case, each number is an average of 20 rankings, since each of the 10 samples contained 2 attacks instead of 1. The *imap* attack type was not included in this group, since only a single *imap* record exists in the test data.

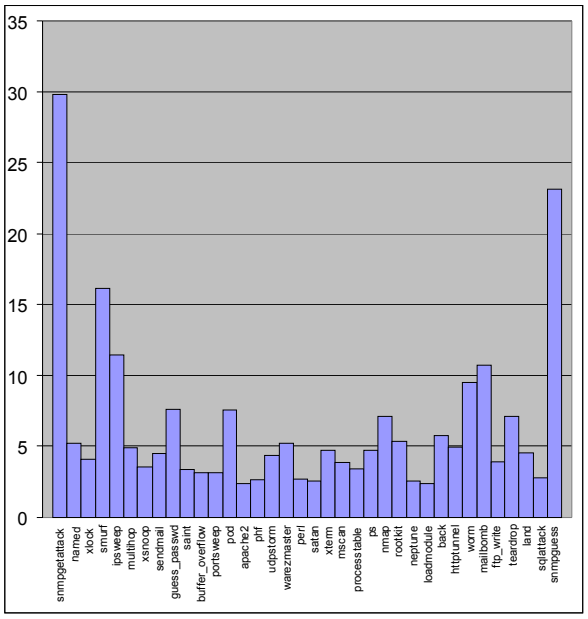


Figure 11. Anomalous subgraph detection; 50 records, 2 attacks.

The results were not as good for this group; 5 of the 36 attack types had an average ranking above 10, with several others above 5. The overall average was about 6.41, up from the first group's average of 4.75. This group performed more poorly than the first in the anomalous substructure detection tests as well.

Figure 12 shows the results for the third group (100 records, 2 attacks).

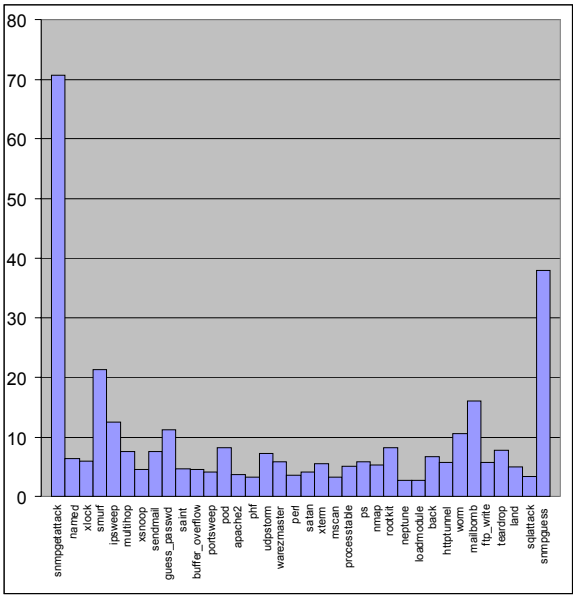


Figure 12. Anomalous subgraph detection; 100 records, 2 attacks.

As with the anomalous substructure detection tests, the third group's results were the best of the three. With 100 records instead of 50, the rankings would be expected to double, all other things being equal. However, many of the attack types still had an average ranking of 5 or less, with all but 7 types having an average of 10 or less. The overall average was around 9.28, slightly less than twice the first group's average.

5.2 Substructure Entropy and Conditional Substructure Entropy

To test our definitions of substructure entropy and conditional substructure entropy, we artificially generated a number of graphs and used Subdue to produce both entropy and conditional entropy values for the graphs. Each graph contained 96 vertices and 96 edges. In each graph, a particular pattern was inserted a certain number of times. This pattern consisted of two vertices connected with an edge, three vertices in a triangle, or four vertices in a square. 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 graph factors 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 entropy or 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 calculated entropy and conditional entropy values vs. the number of patterns inserted in the graph. Ideally, the reported values would fall off smoothly as more and 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, Figures 13 and 14 show the results for graphs with 3-vertex patterns, 6 labels, and a value of n = 2.

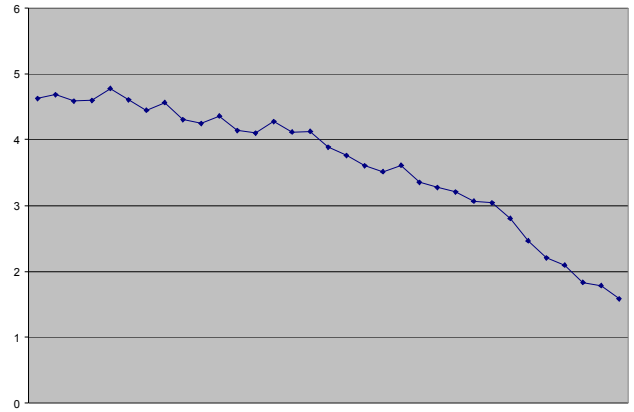


Figure 13. Substructure entropy; 3-vertex patterns, 6 labels, n = 2.

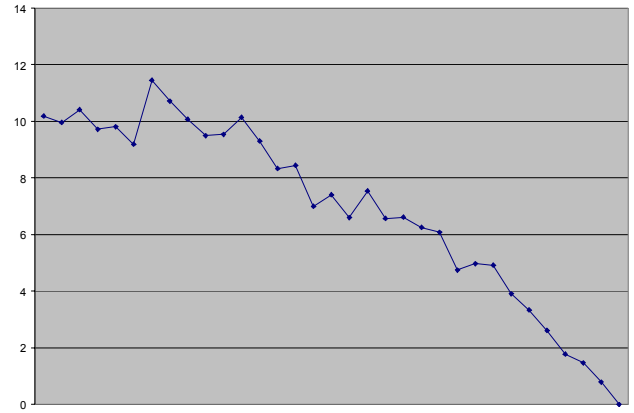


Figure 14. Conditional substructure entropy; 3-vertex patterns, 6 labels, n = 2.

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

The results of other combinations of factors were not as good. As an example, Figure 15 shows the conditional entropy values for graphs containing 2-vertex patterns and 10 labels, along with n = 2.

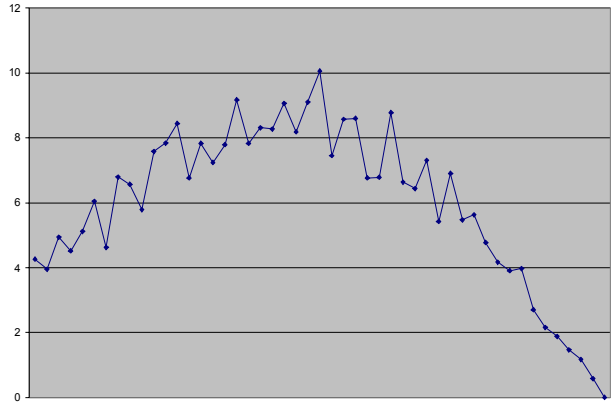


Figure 15. Conditional substructure entropy; 2-vertex patterns, 10 labels, n = 2.

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 of the 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. For a given graph, we consider the "true" conditional substructure entropy to be that of an imaginary graph that is constructed using the same rules but is infinite in size. As is well known, underestimation is also likely when calculating the (unconditional) entropy of a finite data set (see, for example, [7]). In our tests, however, the effect was more noticeable for conditional entropy.

One observed pattern in the (unconditional) entropy charts was that if the value of n was larger than the size of the inserted pattern, the values dropped off very slowly. For example, Figure 16 contains the results for graphs containing a 3-vertex pattern and 6 labels, using a value of n = 4.

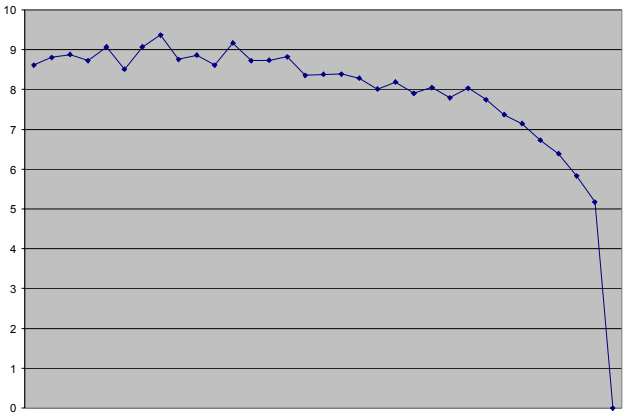


Figure 16. Substructure entropy; 3-vertex patterns, 6 labels, n = 4.

This can be explained as follows: if n (the size of substructures

used to calculate the entropy) is greater than the pattern size, then the patterns are "beneath notice." As the number of (3-vertex) patterns increases, 3-vertex substructures are becoming more and more predictable, but the effect is much less pronounced if we are considering the predictability of *4*-vertex substructures. Many of the patterns will be ignored, unless a randomly-placed edge happens to join them to some other vertex. This shows that the choice of n is important when measuring the substructure entropy of a graph; a poorly chosen value might cause some regular features of the graph to be ignored.

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 17 displays the average rankings vs. the number of repeated "normal" records; the regularity of the data increases from left to right.

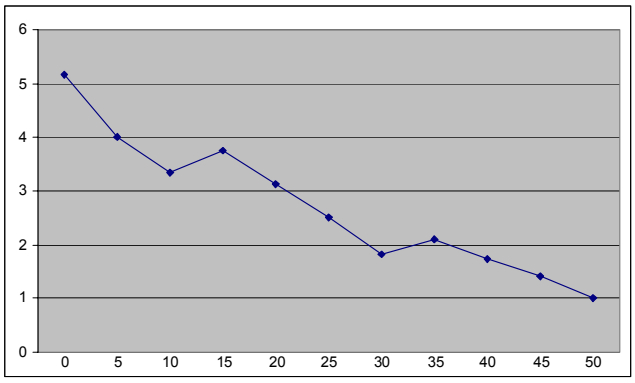


Figure 17. Ranking of attacks as regularity increases.

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 graph-based data 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 defined two measures for calculating the regularity of a graph, using the concepts of entropy and 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 entropy or 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] Cover, T.M. and Thomas, J.A. Elements of Information Theory. Wiley, 1991.

[3] Gonzalez, J., Holder, L.B., and Cook, D.J. Graph-Based Concept Learning. Proceedings of the Seventeenth National Conference on Artificial Intelligence, 2000.

[4] Jonyer, I., Holder, L.B., and Cook, D.J. Discovery and Evaluation of Graph-Based Hierarchical Conceptual Clusters. Journal of Machine Learning Research, 2, pages 19-43, 2001.

[5] 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.

[6] 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.

[7] Miller, G.A. Note on the Bias of Information Estimates. Information Theory in Psychology: Problems and Methods, Free Press, 1955.

[8] Rissanen, J. Stochastic Complexity in Statistical Inquiry. World Scientific Publishing Company, 1989.

[9] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html