GPU Accelerated Graph-Based Anomaly Detection

Ian James Thomas Dr. Enyue (Annie) Lu*

August 13, 2020

Abstract

Detecting abnormal events in massively interconnected networks have become challenging due to the immense amount of data and run time limitations of currently existing anomaly detection systems. This research experiment explores different approaches to detecting anomalous network traffic in large data sets which are capable of taking advantage of high performing GPUs (Graphics Processing Units). As anomaly detection typically depends on some form of machine learning or intensive data analysis, parallel computing methods are used to accelerate anomaly detection systems. The massive parallelism offered by modern GPUs has yet to be widely explored, and could allow for anomaly detection in real-time. This paper will detail the implementation and comparison of two approaches to graph-based anomaly detection; graph compression (using the SUBDUE algorithm) and graph clustering (using the barycentric clustering approach).

*Department of Mathematics and Computer Science, Salisbury University

1 Introduction

Detecting abnormal events or patterns in a network has become challenging due to massive increases in the amount of data transmitted between connected systems. As a result, better performing anomaly detection systems are becoming necessary for high traffic networks. In this project, two common approaches to graph-based anomaly detection are explored, and are implemented fully in CUDA, a GPGPU platform which takes advantage of high performing NVIDIA GPUs to accelerate program run-time.

Data can often be better represented as a graph (a set of vertices and the edges connecting them) if there is an interconnected nature of the data being analyzed. The 1999 KDD-Cup Dataset is a labeled dataset of TCP records commonly used for anomaly detection experimentation. Records are labeled as either normal, or as pertaining to a type of intrusion/DDoS attack [1]. The two approaches being explored will be used to represent the KDD-Cup dataset as a graph, then label each record as anomalous or normal based on similarity measures to other data in the dataset.

Both approaches are unsupervised learning methods (which is standard for anomaly detection systems to discourage bias towards specific 'normal' data being experimented on).

2 Brief Technological Overview

The NVIDIA CUDA programming platform will be used for the experiments. CUDA architecture is structured such that streaming multiprocessors can execute portions of a program that can take advantage of parallel computing. Streaming multiprocessors consist of thread-blocks, which are collection of threads which share a common memory bank[2]. Multiple thread-blocks are capable of running concurrently, and as a modern GPU consists of several streaming multiprocessors, each of which capable of executing multiple thread-blocks, portions of a program that can be parallelized can be greatly accelerated when running on a GPU platform. GPU computing was chosen for this experiment as the size of the dataset does not justify the overhead of distributed computing, and most of the algorithms rely on arithmetically intense operations which could easily be performed on a GPU.

3 Graph Compression Approach

The SUBDUE graph compression algorithm is used for exploring the graphical compression approach to anomaly detection. Typical graph compression algorithms operate by replacing a specific pattern of vertices in a graph with a single vertex. The SUBDUE algorithm operates in this manner, where the given graph is iteratively traversed, and all repetitive subgraphs (e.g. substructures within the graph) are replaced with a single vertex until there are no more subgraphs consisting of two or more vertices. Each subgraph which is compressed is stored in a list (referred to as a *Parent List*) in order to keep track of which vertex represents which subgraph. The SUBDUE algorithm is commonly used for locating frequently occurring patterns in large scale graphs, however in this context, some modifications to the original algorithm were made to use the SUBDUE algorithm for detecting records with very few frequently occurring patterns in contrast to the rest of the dataset.

There is an underlying assumption in this approach, which is that records containing few frequently occurring patterns are considered "more anomalous" than records containing many occurring patterns. Individual records will be represented in a star-graph configuration, whereas the record itself is considered the internal vertex while all its attributes are its leaf vertices. Frequently occurring leaf vertex orientations (patterns where two or more leaf vertices are configured identically in multiple records) are replaced with a single leaf vertex until there are no more leaf vertex patterns consisting of two or more vertices. After all patterns have been compressed (replaced by a single vertex) each record is given a score based on the percentage of the record that was compressed on each iteration. The output will be referred to as an "anomaly score" A, where $0 \leq A \leq 1$. The following equation is applied to each individual record in order to generate its anomaly score:

$$A = 1 - \frac{1}{n}\sum_{i=1}^{n}(n - i + 1) * C_i$$

Whereas n is the number of iterations, i is the current iteration, and C_i is the percentage of the record which has been compressed on the i^{th} iteration. C_i can be defined as $\frac{1}{(|R|)}$ for a given record, where R is the record (and $|R|$ is the number of attributes in the record, in the case of KDD-Cup, there are 41 attributes in a given record, however the number of attributes decreases as they are compressed).

The equation is meant to allow compressions of which occurred earlier in the algorithm to have a greater influence on the overall score. This is because patterns which are found earlier on appear more frequent, and thus assumed to be less anomalous than patterns found later in the algorithm which appear in a smaller portion of records. The $(n - i + 1)$ portion will vary from n to 1 as i increases, causing A to drop off sharply if more compressions occur earlier. The $1/n$ term guarantees a final result between 0 and 1, whereas a score closer to 1 represents a high likelyhood of the record being anomalous.

4 Graph Compression. Implementation details

In the implementation, the star graph configuration of each record is composed of each unique two element pairing of all its attributes. For example, a record composing of three attributes (tcp, 0, 12) will require the pairings (tcp, 0), (tcp, 12), (0, 12) to be observed. For the KDD-Cup dataset, each record consists of 41 attributes, meaning 820 pairings will be observed for each record. In practice, these pairings are not generated, but rather directly compared when observing multiple records in to avoid the memory overhead of generating $\binom{n}{k}$ pairings, where n is the number of attributes in a record and k is the length of a pairing (memory overhead of which can easily be greater than the memory available on a GPU).

All pairings are observed, and the most frequently occurring pairing will have both attributes replaced with a single element. In the implementation, each element that is compressed is replaced by the patterns negative index in the parent list, as negative values do not appear in any KDD-Cup records. The original SUBDUE algorithm requires that the algorithm terminate

only after there are no more subgraphs consisting of two or more vertices. Past experiments show that around 30 iterations appear sufficient [3]. Most subgraphs located after 30 iterations only appear in a small portion of the records, which by definition is anomalous.

The algorithm consists of three primary parts, locating the most frequently occurring subgraph iteratively, compressing said subgraph iteratively, and scoring all records based on how compressed they are by the termination of the algorithm. Compressing and scoring are both done in approximately R time. However, locating the most frequently occurring subgraph can take $\binom{n}{k}R^2$ time as each unique pairing must be compared to all other unique pairings which appear in the rest of the dataset. To overcome this time constraint, each CUDA thread-block is responsible for a unique pairing, whereas the threads within a block are assigned to unique records. In practice, $\binom{n}{k}$ thread-blocks are required for observing all two element pairings (820 for KDD-Cup), whereas t threads in each block are strided amongst all records to observe the specific pairing. The most common pairing is then selected by the host, and that element is compressed in all records it appears in (on the host).

Accuracy and sensitivity measurements are taken using each records anomaly score A and some threshold for determining whether a score was anomalous or not (usually between [.5, .6]) whereas any record with a score above that threshold is assumed to be anomalous. Using these scores and the actual label for each record, the true positive (TP), true negative (TN), false positive (FP), and false negative (FN) rates could be determined. Accuracy and sensitivity were determined through the same equations as [2]

$$Sensitivity = \frac{TN}{TN + FP}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Testing was performed with smaller samples. Out of 499 normal records and one anomalous record (an IP sweep attack) the anomalous record was correctly identified with an A score of .509 (slightly above the threshold of .5 chosen for the full dataset). This was the case for most attack types, scoring around .501 - .522. The ftp write, back, and neptune attack types scored around .33, well below the threshold. A solution for false negatives could be a dynamic threshold which would change based on the distribution of A scores.

When many records and attacks were used, there was a much wider distribution of A scores, which is the desired result when using a static threshold. Testing has found that the graph compression approach has high accuracy, but low sensitivity. A subset of the entire dataset was used in testing consisting of 5500 records, with varying number of attacks. Accuracy remains rather consistent with a slight decrease when the number of attacks varies, though sensitivity gradually increases when more attack types (especially of the same specificity) increases. Figure 1.1 shows how accuracy and sensitivity vary when there is a greater percentage of attack records in the dataset.

It is worth noting that the type of attacks that were used were proportional to how they appear in the entire dataset. Most attacks in the

Attacks	Accuracy	Sensitivity
33 (.06% of dataset)	99.1%	30.3%
82 (.15% of dataset)	99.28%	87.8%
204 (.38% of dataset)	99.32%	94.12%
381 (.79% of dataset)	98.72%	87.4%

Figure 1.1 Results of test using 500 KDD-Cup 99 dataset records on graph compression method

dataset are of the Neptune type, whereas other attack types are in small proportions to the rest of the attacks. If there were greater variation in attack types, sensitivity would likely decrease (as there would be less similarity in attacks), but accuracy would likely increase slightly (as attack records would be even more anomalous since there is less similarity).

5 Graph Clustering Approach

When implementing a graph clustering approach to anomaly detection, one must consider whether a clustering approach makes any assumptions of the size of the clusters that will be generated. For example, the commonly used spectral clustering method assumes all clusters will be about evenly sized. In anomaly detection this is not the case as the size of the cluster consisting of normal records will be far greater than that of any anomalous cluster. The barycentric graph clustering method makes no assumptions regarding the size of the clusters. Barycentric clustering uses a physics based model, where edges are imagined as springs connecting vertices. The weight of an edge is representative of the amount of force on a given vertex, where a high weight would represent a high amount of force. Vertices are moved to completely random locations, where they are then 'released' to see where their new position will be based upon the forces enacting on them. After t trials, edges with abnormally high weights are cut, and leaving all closely related vertices in clusters.

A complete graph is formed from the dataset, where vertices are represented of individual datapoints, and edges between all vertices are made from some similarity measure between the vertices. All vertices are then moved to a randomized location in physical space (one-dimensional space is sufficient for representing physical space [4]) and released allowing the weight of the edges to exert a force, influencing the new physical location of the vertex.

The edge connecting vertices i, j exerts a force $-k_{ij}(p_i - p_j)$ where k is the weight of the edge connecting i and j, and p_i and p_j are the physical locations of the vertices i, j. Each vertex is influenced by the force exerted by all of its neighbors, such that a single iteration adjusts each vertices position by

$$\Delta p_i = -\mu \sum_{j=1}^{n} k_{ij}(p_i - p_j)$$

Where Δp_i is the locational change of vertex i, n is the number of neighboring vertices, and j is the index of the neighbor vertex.

Some gain μ is also chosen in order to bound the force exerted by all neighbors as some vertices are drastically pulled apart when one neighbor exerts a stronger force than another. The gain μ can be chosen as $\mu_i = \dfrac{1}{d_i + 1}$ where $d_i = \sum_{1 \le j \le n} w_{ij}$ where w is the weight of the edge connecting vertex i and j.

This process of iteratively moving vertices

based on the force Δp_i is repeated s times. There are t total iterations where vertices are moved to random positions in physical space and iteratively moved based on neighboring forces.

After t iterations, edge lengths (if working in one-dimensional space, measured by $|p_i - p_j|$) are averaged and abnormally long edges are simply cut. This will leave all closely related vertices in clusters with few outgoing edges to vertices that belong to other clusters.

6 Graph Clustering, Implementation details

To first represent the dataset as a graph, a radial basis function was used a similarity measure to determine how similar two network records are, and to use as the edge weight between vertex i and j

$$W_{ij} = \frac{1}{exp(||\vec{v_i} - \vec{v_j}||)^2}$$

Where $\vec{v_x}$ is the vector associated with the x^{th} record, in this context representative of the x^{th} graph vertex. All continuous record attributes (e.g. source/destination bytes) are scaled between [0, 1]. Symbolic attributes (e.g. protocol type) are assigned either 0 if identical, or 1 if they are different. The purpose of this function is to allow a higher distance between two vertices if they are not similar (thus decreasing the weight of the edge connecting them), and oppositely a lower distance and higher weight to two similar vertices in the graph.

In the original approach, a complete graph is generated (whereas every pair of vertices is connected by a unique edge). In practice, this is unfeasible as the full dataset contains over 5 million network records, meaning there would be slightly less than 12.5 trillion edges to represent the full dataset as a complete graph. To remedy this, a K-nearest neighbors graph is introduced. A Knn graph serves as an approximation of complete graphs for the purpose of clustering. Edges only exist between a vertex and their k nearest neighbors (decided by edge weight). Previous experiments suggest a K value of 50 is adequate [5].

After generation of the Knn graph, five iterations of distributing each vertex to a random location, calculating the force upon the vertices based on their neighbors, and averaging the distance between each vertex and their neighbor are calculated. Five trials were sufficient in [4]. After the end of all trials, edges of above average length were cut.

The barycentric clustering algorithm only takes R time if the k value is chosen to be infinitesimal compared to R, making the runtime reasonable. Though the generation of the Knn graph can take as long as R^2 whereas the weight of all unique edges must be generated in order to determine if they are within the k highest weighted edges of a given vertex. Generation of the Knn graph is implemented on the GPU simply by distributing each vertex to a unique thread.

With graph compression, the final output was simply a percent assigned to each record. With clustering, the output is another graph with less edges than it had originally. To measure accuracy for this approach, the outgoing edges in each vertex are recorded, and an average for each vertex is taken based on how many outgoing edges are the same type as that vertex. For example, if

a vertex representing a normal record has three outgoing edges to other normal records, and one to an attack record, the accuracy for that specific vertex would be 75%. Sensitivity is measured similarly, whereas the specificity of a given vertices attack type of an outgoing record is compared to that of the origin vertex.

Accuracy for a dataset containing few anomalies will naturally have a slightly lower accuracy and sensitivity as there will be less vertices to form clusters composed of anomalous records. The same set of 499 records and one anomalous record as tested in the previous approach resulted in a total accuracy and sensitivity of 99.1%. Though it would be difficult to determine if the record was anomalous as it would still belong to the main cluster of normal vertices as there simply are no other vertices to form an anomalous cluster.

7 Comparison of Approaches

It was observed that the graph clustering approach had significantly better performance than the graph compression approach. This is likely due to the clustering approach only having one significant operation which takes R^2 time, whereas the compression approach requires a similarly expensive $\binom{n}{k}R^2$ operation, although upon each iteration.

Both approaches had roughly the same accuracy. The graph clustering approach typically had a better sensitivity when very similar attack types were used. For example, a subset of 30,000 records, where all attacks were of the same three types (smurf, teardrop, ipsweep) resulted in a very high accuracy and sensitivity (both above 99.7%) for the clustering approach.

Figure 1.2 is the result of five tests performed on various record counts on the barycentric clustering method. Sensitivity is not listed as it closely followed the accuracy (at worst, being .01% less than accuracy). Results taken from the graph compression method are also listed in Figure 1.3. Note that large scale testing could not be performed in this experiment due to technological constraints.

Records	Runtime	Accuracy
50	0.28s	98%
500	0.33s	99.1%
5500	0.898s	99.7%
35000	19.37s	99.9%
50000	33.94s	99.7%

Figure 1.2 Results of test using various amounts of records, using graph clustering method

Records	Runtime	Accuracy
50	0.27s	99.9%
500	0.33s	99.1%
5500	11.21s	98.7%
10000	40.2s	98.3%

Figure 1.3 Results of test using various amounts of records, using graph compression method

It can be inferred from the results that the graph clustering approach is much more suitable for most applications which deal with larger datasets, especially those with very homogeneous anomalous records. The graph compression has shown to have a poor runtime, though past experiments using Apache Hadoop (a distributed computing platform) have resulted in some adequate runtimes for much larger datasets using

the same SUBDUE algorithm [3]. The GPU accelerated implementation has decent performance and accuracy for smaller datasets, and in some cases may be more suitable than clustering if there are very few anomalous records.

References

[1] S. D. Bay, D. F. Kibler, M. J. Pazzani, and P. Smyth *The UCI KDD Archive of Large Data Sets for Data Mining Research and Experimentation.* 2000.

[2] Jayshree Ghorpade, Jitendra Parande, Madhura Kulkarni, Amit Bawaskar *GPGPU Processing In CUDA Architecture.* Jan. 2012.

[3] Joyce, B. *Graph Based Anomaly Detection using MapReduce on Network Records.* University of North Carolina, August 2016

[4] Jonathan Cohen *Barycentric Graph Clustering.* 2008.

[5] Corbin McNeill, Enyue Lu, Matthias Gobbert *Distributed Graph-Based Clustering for Network Intrusion Detection.* Wheaton College, IL, 2015