Graph Filtering to Remove the "Middle Ground" for Anomaly Detection

William Eberle
Department of Computer Science
Tennessee Tech University
Cookeville, TN, USA
weberle@tntech.edu

Lawrence Holder
School of Electrical Engineering
& Computer Science
Washington State University
Pullman, WA, USA
holder@wsu.edu

***Abstract*—Discovering patterns and anomalies in a variety of voluminous data represented as a graph is challenging. Current research has demonstrated success discovering graph patterns using a sampling of the data, but there has been little work when it comes to discovering anomalies that are based upon understanding what is normative. In this work we present two approaches to reducing graph data: subgraph filtering and graph filtering. The idea behind the proposed algorithms is the removal of a "murky middle", where data that may not be normative or anomalous, is removed from the discovery process. We empirically validate the proposed approach on real-world, pseudo-real-world, and synthetic data, as well as compare against a similar approach.**

Keywords—graph-based anomaly detection, graph filtering, knowledge discovery

I. INTRODUCTION

The ever-increasing volume, velocity and variety of data continues to challenge our ability to extract knowledge from data. In addition, interconnected relationships across data sources introduce further representational and computational challenges. Examples abound such as social networks, biological networks, communication networks, and even brain networks. The graph has emerged as an appropriate representation for such data, and numerous methods have been developed for extracting knowledge from networks. However, since the type of data we seek to analyze is continually growing, such as in telecommunications networks, we must be able to handle the data in a computationally effective way in order to extract relevant knowledge.

Current work in the area of pattern discovery and anomaly detection in networks, or more specifically in graphs, has had to deal with computational difficulties. Most have addressed this issue by either only handling a sample of the graph [6][7][8][9], transforming the graph into a smaller representation of its structural properties [12][10], or reducing a visualization of the graph [14][11]. In any case, they are not dealing with the task of anomaly detection in the graphs while still reducing the computational complexities.

In this work, we develop new methods for filtering graphs so as to reduce the computational complexities without losing our ability to discover relevant normative patterns and interesting anomalous subgraphs. We present two approaches to address the computational challenges: subgraph filtering, for reducing the number of subgraphs at each iteration of edge extensions when growing the list of candidate patterns; and, graph filtering, for reducing the size of the input graph before searching for patterns and anomalies. What makes this novel is that we are intelligently filtering the graph rather than sampling. By excluding the "middle" of the graph, or the "middle" candidate subgraphs, we are able to retain the best normative patterns and the best potential anomalous subgraphs – thereby, reducing the number of graph matches by ignoring those subgraphs that should never be normative or never be anomalous. We evaluate these methods using actual, near real-time network data provided by one of the leading managed security services providers at a major telecommunications company, with input from their domain experts.

II. RELATED WORK

Much work has been done on sampling and filtering graphs in order to improve the efficiency of graph mining methods that operate on high-volume data. Kashtan et al. [6] propose a sampling method that randomly samples *n* nodes in order to extract connected subgraph samples that are of order n. Their approach consists of repeatedly sampling subgraphs of order n until the desired sample size is reached. The concentration of different subgraphs is then estimated and used to find motifs in the network. Wernicke [7] modified the sampling scheme of Kashtan et al. in order to correct the sampling bias. The TIES algorithm [8] also uses a sampling scheme; however, the objective of TIES is to sample a subgraph that is representative of the entire graph. Ahmed et al. [9] propose the PIES algorithm for sampling a representative subgraph from a large streaming graph represented by a sequence of edges. PIES samples edges randomly and then stores the nodes from the sampled edges. It also performs stream sampling by maintaining a reservoir of nodes. When the reservoir is full, it probabilistically decides to drop old nodes and their incident edges from the sample and include new nodes and edges.

However, in all of the above cases, the approaches are not dealing with anomaly detection. Sampling is complicated in our setting, because in the case of finding normative patterns, we want to keep common structures and discard unique structures; whereas, for anomaly detection, we want the opposite. This suggests more of a filtering strategy that identifies "middle

978-1-7281-6251-5/20/$31.00 ©2020 IEEE

ground" structures that are not common enough to be normative, yet not unique enough to be anomalous; therefore, candidates for exclusion from the data.

There has been some work in graph filtering. Irniger [12] proposes a graph database retrieval approach for dealing with the computational complexities associated with graph matching. Irniger's graph database filtering approach involves generating features that can be quickly computed, and then using them in discriminating the graphs. Lugowski et al. [14] propose a toolkit that allows users to perform complex analysis of huge datasets, including in-place graph filtering. Filtering is applied on the fly, reducing the graph visualization of what the domain expert would be viewing. Egilmez and Ortega [10] propose a graph-based filtering technique for detecting anomalies in wireless sensor networks. Their filtering approach captures various structural information that can then be used in principal component analysis (PCA) – something that could not be done without filtering the graph into numeric values. Eems et al. [11] propose to filter a graph visualization based upon various graph metrics (e.g., degree). Their approach then subsequently uses the visualization to determine outliers. Pawar and Zaveri [13] propose a filtering approach prior to matching for graphical symbol recognition. Their approach uses graph databases, and to reduce the computational complexities of graph matching they reduce the number of symbols in the database prior to matching.

However, in all of these filtering approaches, they are either not dealing with anomaly detection, and filtering is only applied to retrieve a sample for normative pattern discovery; or they are not using filtering to reduce the size of the graph, or subgraphs, in order to decrease the computational complexity of pattern discovery and anomaly detection. One approach that comes the closest to our objective of discovering both normative and anomalous subgraphs while detecting anomalies can be found in the preliminary work of Ioannidis et al. called GraphSAC [15]. They propose an approach that samples graphs, performs predictions on the samples, and those predictions that result in incorrect classifications are filtered out. The subsequent graphs are then used to train the model, and report anomalies from the trained model. This approach does lead to some reduction in running times, but the focus is on homophilic anomalies, which are nodes whose attributes are dissimilar to their neighbors, and the purpose of filtering is to improve the accuracy of the model training. Another approach that performs graph-based anomaly detection with some level of graph reductions can be found in the work of Parkinson et al. called GraphBAD [18]. They propose an approach that detects relational anomalies, reducing the input graph by merging vertices that are not directly connected but share a similar set of edges. While they show some success on the KDD Cup '99 data set, their technique is specifically targeting the detection of anomalies in security system configurations.

Our approach, described in Section IV, uses filtering to improve efficiency, but in a way that supports both normative pattern discovery and anomaly detection. We first provide some background in the next section.

III. Graph-Based Anomaly Detection

A. Definitions

In this, work we define a graph as follows:

Definition: A **graph** $G = (V, E, A)$ consists of a set of vertices or nodes V, a set of edges $E \subseteq V \times V$, and a mapping A on vertices and edges to their attributes. Edges can be directed or undirected; self-loops and multi-edges are allowed. Attributes are represented as a set of key/value pairs. Each edge has a designated "edge type" attribute.

Definition: A **pattern** P in graph G is a connected graph that is isomorphic to a subgraph of G, and the attributes on the vertices and edges of P are a subset of the attributes on the corresponding vertices and edges in G. This is sometimes referred to as a **substructure**.

Definition: An **instance** I of a pattern P in a graph G is any subgraph of G that is isomorphic to P.

We next define an *anomaly*. Note that this definition of an anomaly is unique in that it is defined in the context of a normative pattern, i.e., a pattern that occurs often in a graph. In our work, an anomaly is a closely, but not exactly, matching instance of a normative pattern that has an unexpected deviation. This is to distinguish anomalies from *noise*, which are expected deviations, and *outliers*, which are not related to a normative pattern. The importance of this definition lies in its relationship to deceptive practices that are intended to illegally obtain or hide information. For example, the United Nations Office on Drugs and Crime states the first fundamental law of money laundering as "The more successful money-laundering apparatus is in imitating the patterns and behavior of legitimate transactions, the less the likelihood of it being exposed" [1].

Definition: An **anomaly** I_A of a normative pattern P in graph G is a subgraph of G such that distance $0 < d(I_A, P) < T_1$, and probability $P(I_A \mid P, G) < T_2$, where d is a distance metric between two graphs, and T_1 and T_2 are user-defined thresholds.

Our main objective in this work is to efficiently and effectively mine patterns and anomalies from graphs

B. GBAD

The approach in this work is based on our previous work on graph-based anomaly detection (GBAD) [3]. Here we briefly review the GBAD approach.

There are three general categories of anomalies in a graph: insertions, modifications and deletions. Insertions consist of unexpected additional structure in an instance of the pattern. Modifications consist of unexpected structure in place of a part of an instance of the pattern. Deletions consist of the unexpected absence of structure from an instance of the pattern. GBAD discovers each of these types of anomalies. Using a greedy beam search and a minimum description length (MDL) heuristic, GBAD first discovers the best subgraph, or normative

pattern, in an input graph. The minimum description length (MDL) approach is used to determine the best subgraph(s) (i.e., normative pattern) as the one that maximizes the following:

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

where C is the amount of compression; G is the entire graph; S is the subgraph pattern; $DL(G)$ is the description length of G; $DL(G|S)$ is the description length of G after compressing it using S, and $DL(S)$ is the description length of S. Using a beam search (a limited length queue of the best few patterns that have been found so far), the algorithm grows patterns one edge at a time, continually discovering which subgraphs best compress the input graph. The strategy iteratively extends each subgraph by one edge (and its attributes) and evaluates each extended subgraph based upon its compression value. A list is maintained of the best subgraphs, and this process is continually repeated until either there are no more subgraphs to consider or a user-specified limit is reached.

The GBAD approach is based on the exploitation of structure in data represented as a graph. We have found that a structural representation of such data can improve our ability to detect anomalies in the behaviors of entities being tracked [2]. GBAD discovers anomalous instances of structural patterns in data that represent entities, relationships and actions. GBAD uncovers the relational nature of the problem, rather than solely the traditional statistical deviation of individual data attributes. Attribute deviations are evaluated in the context of the relationships between structurally similar entities. In addition, most anomaly detection methods use a supervised approach, requiring labeled data in advance (e.g., illicit versus legitimate) in order to train their system. GBAD is an unsupervised approach, which does not require any baseline information about relevant or known anomalies. In summary, GBAD looks for those activities that appear to match normal/legitimate/expected transactions, but in fact are structurally different. For more information regarding the GBAD algorithms, the reader should refer to [3].

IV. Proposed Filtering Approaches

Our proposed strategy for scaling graph pattern learning and anomaly detection methods is to filter candidate patterns and anomalies based upon what we know about the graph at each stage of the pattern search (i.e., subgraphs discovered after each edge extension.) We propose two approaches: subgraph filtering and graph filtering. Since our definition of anomaly is based upon deviations to the normative pattern, it is important that we not only discover instances of the normative pattern, but also those (supposedly few) anomalous instances of the normative pattern. Therefore, those instances that are not normative and are not anomalous (i.e., the "middle ground") can be filtered.

A. Subgraph Filtering

In the proposed *subgraph filtering* approach, a percentage of "middle" valued subgraphs, or patterns, are removed from consideration after each edge extension, thus reducing the list of candidate subgraphs for the next round of edge extensions. The idea is that we will keep the potentially best (normative) and worst (anomalous) *subgraphs*, and avoid processing candidate subgraphs that will never be either. Algorithm 1 (*FilterSubs*) below implements this filtering strategy. F indicates the percentage of the "middle" substructures, where 100-F/2 of the top, and 100-F/2 of the bottom substructures, are retained. For example, if a value of 10 is chosen for F, the top 45% and bottom 45% (i.e., 100-F/2) would not be filtered out. The filtering percentage F can be chosen by the user or tuned based on the data.

Algorithm 1: **FilterSubs** (*SL*, *F*)

Input: *SL*: subgraph list
F: filtering percentage

1. Filtered subgraph list *FSL* = {}
2. *FSL* = *FSL* ∪ {*S* in *SL* | *C(S,G)* in top (100-*F*)/2 subgraphs in *SL*}
3. *FSL* = *FSL* ∪ {*S* in *SL* | *C(S,G)* in bottom (100-*F*)/2 subgraphs in *SL*}
4. return *FSL*

At each extension of candidate subgraphs (*PSL*), which is used in the subgraph discovery as shown in Algorithm 2 (*DiscoverSubs*), extended subgraphs are given a value (e.g., compression score) and stored in sorted order (highest to lowest – most normative to least normative) in the candidate subgraph list. The middle filtering percentage of the current list of subgraphs is removed from consideration after each edge extension, thus reducing the list of candidate subgraphs (*PSL*) for the next round of edge extensions.

Algorithm 2: **DiscoverSubs** (*G*, *F*)

Input: *G*: graph
F: filtering percentage

1. Discovered subgraphs list *DSL* = {}
2. *PSL* = {all unique one-vertex subgraphs in *G*, ordered by value}
3. while *PSL* not empty
   a. if *F* > 0, then *PSL* = **FilterSubs**(*PSL*, *F*)
   b. *DSL* = *DSL* ∪ *PSL*
   c. *PSL* = extend each subgraph in *PSL*, ordered by value
4. return *DSL*

B. Graph Filtering

In the proposed *graph filtering* approach, a *graph filtering percentage* is specified, whereby the middle filtering percentage of the current list of edges - based upon the frequency of edge types - are removed from the input graph, thus reducing the size of the graph. The idea is to keep the potentially normative and potentially anomalous *edges*, and filter edges that could never be either. Algorithm 3 (*FilterGraph*) implements the graph filtering approach.

Algorithm 3: **FilterGraph** (G, F)

Input: G: graph
F: filtering percentage

1. Maintain sorted counts by edge type of edges E in G
2. $E' = \{$edges with top $(100\text{-}F)/2$ most frequent edge types in $G\}$
3. $E' = E' \cup \{$edges with bottom $(100\text{-}F)/2$ most frequent edge types in $G\}$
4. Store and process only edges in E', discarding edges in $(E - E')$

While we will use GBAD to empirically evaluate both of these filtering approaches, the algorithms are generic and can be applied to other anomaly detection methods. For instance, the storage and filtering of edges could be done in a graph database. This would allow other graph-based anomaly detection approaches [4][5] to implement the *FilterGraph* algorithm within a database.

V. Experiments and Analysis

We tested the proposed filtering approaches on the following three different types of data:

1. Network traffic from a major telecommunications company, in order to test the efficacy of the approach in a real-world setting;
2. Pseudo-real-world data representing a social network, in order to evaluate with more complex relationships and actions; and
3. Synthetic data, in order to test varying normative patterns and increasing volumes.

Performance is evaluated on both run-time (CPU seconds) and accuracy, where ground truth is known. In addition, we conclude our experiments by comparing our proposed approach against the publicly available GraphBAD approach mentioned earlier [18].

A. Data and Graphs

The following is a description of each of the datasets used in these experiments, and the different characteristics present for evaluating a diverse set of data.

1) Telecommunications Records

The telecommunications data used in these experiments consists of text messages (SMS), call detail records (CDR), and private network data records (IPDR), captured from the network of a major telecommunications company. This data is used by analysts for the purposes of monitoring for security risks associated with vehicular communications traffic.

For the following reported results, the data was captured from a single day (September 11, 2018), and consists of a total of ~8M records, where ~3K records per minute are received during non-business hours, ~5K vertices/edges per minute during early morning and late in the afternoon, and ~10K vertices/edges per minute in the middle of the day. Each record consists of a mobile device number, date and time that the event started and ended, the duration of the event, the size of the packet, source, destination, geolocation, and various other telephony pieces of information (refer to documentation on CDR SMS, and IPDR records).

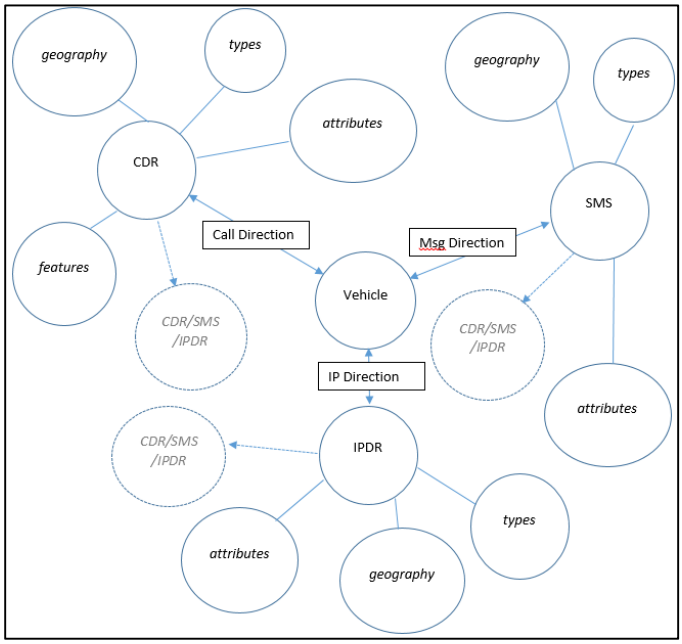

Figure 1. Graph topology of telecommunications records.

The data is then converted to a graph where nodes represent each record type (SMS/CDR/IPDR) and their individual attributes (e.g., call direction, termination code, service feature code, etc.). Each record type is linked if its mobile device number is the same, and directional edges represent the direction from source (origination address) to target (destination address), whether it is a text message, a call, or internet traffic.

Figure 1 is a high-level representation of the graph topology design used to create the graph input files for the following experiments. The usefulness of this data set is the real-world structure of a system that handles multiple telecommunication-types of records, and the access to ground truth anomalies.

It should be noted that the researchers of this work were not granted access to other information that would have identified any individual or individuals. In addition, the tools and algorithms identified in this work were only allowed to access the data on secure, internal company servers. However, the company's network security analysts were able to provide the ground truth anomalies for this data, allowing us to evaluate the accuracy of our approach. It should also be noted that this data is proprietary, and is unavailable for public distribution.

2) Social Network

The social network data was generated using the Linked Stream Benchmark [17]. LSBench generates data that represents users and their actions, including GPS locations, posts, and photo albums, as well as "like"s and "know"s (https://code.google.com/p/lsbench/). LSBench generates sets

of RDF triples of varying sizes and periods of time, that contain user information, and their associated locations, devices used, postings, photos, likes, and whom they know.

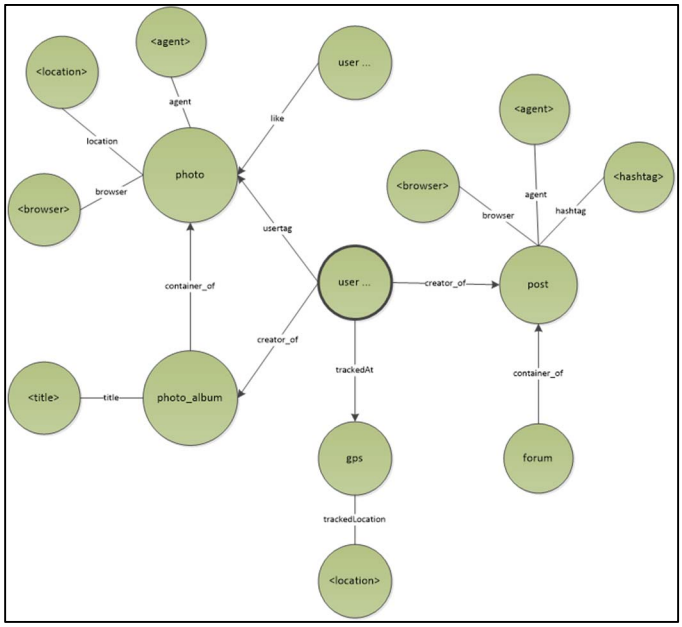

Figure 2. Graph topology of social network.

We represent the LSBench objects as vertices, and actions as edges. Figure 2 is a high-level representation of the graph topology design used to create the graph input files for the following experiments. However, unlike the telecommunications data, we do not have any intrinsic anomalies created. Thus, we will seed random anomalous vertices and edges, including unusual likes (e.g., a person liking a group that does not fit their usual profile), unusual follows (e.g., a person joining a forum that is different from the forums they typically follow), and unusual memberships (e.g., a person belonging to an organization that is out of the ordinary). Figure 3 is an example of an anomaly, where the pop star Cyndi Lauper is discovered among the fan group of opera star Andrew Bocelli.

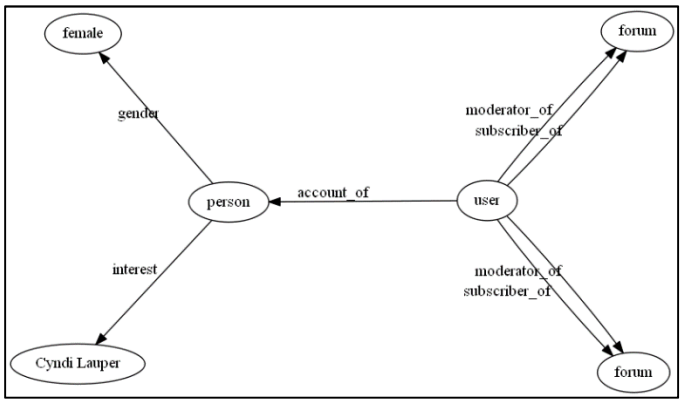

Figure 3. Social network example of an anomaly.

3) Synthetic

For the synthetic graphs, we used a tool called *subgen* [16] that generates random graphs based upon user-specified parameters, including:

- total number of vertices and edges
- possible vertex and edge attributes and probabilities
- subgraph pattern
- amount of connectivity

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

By using the synthetic graph generator, we can evaluate the effectiveness of our approach on subgraphs with varying structural properties. Since the telecommunications and social network data typically provide star graph structures, we will experiment with three types of graph structures: paths, trees, and cliques. Figure 4 is a high-level representation of the graph topology for each of the normative substructures. We randomly seed different types of structural anomalies into the data, and given the control we have over the generation of the graphs, we are able to generate many graph instances, each with random differences, so as to further validate the effectiveness of the approach.

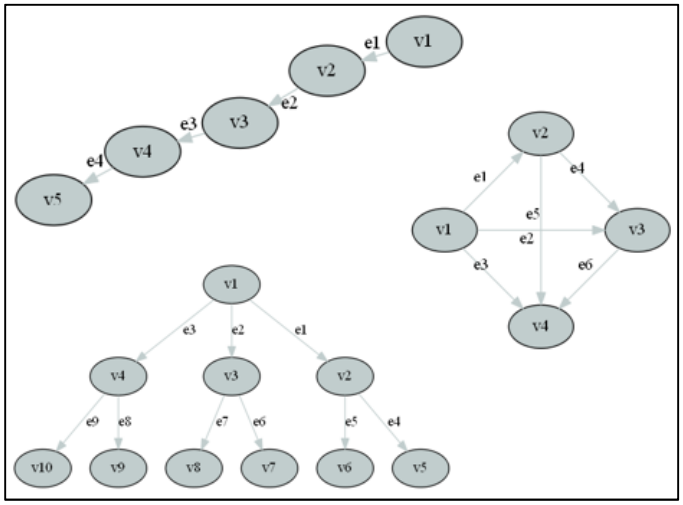

Figure 4. Graph topologies for path, tree, and clique.

Finally, in order to evaluate the approach on a variety of scales, for each of the different patterns (shown in Figure 4), we varied the total number of vertices/edges as follows: *small sparse*, or low connectivity, graphs (~100K vertices/~100K edges) and *dense*, or high connectivity, graphs (~100K vertices/~500K edges); medium-sized graphs (~1M vertices/edges); and large graphs (~10M vertices/edges). We then performed 30 experiments per filtering level, with each graph instance consisting of different random insertion/deletion/modification anomalies, for a total of 300

runs *per* graph topology (i.e., path, tree, and clique) and size type (small sparse, small dense, medium, and large).

B. Subgraph Filtering

The following are results of our experiments with varied subgraph filtering percentages.

1) Telecommunications Data

Performing normative pattern discovery and anomaly detection on *15 minutes of data at a time*, with varying subgraph-filtering percentages, we observe the results shown in Figure 5. We note that due to the wide range of running times given the diversity in scale for the various experiments, we are presenting the duration in seconds as a logarithmic value, which provides an easier visual understanding of the trend.

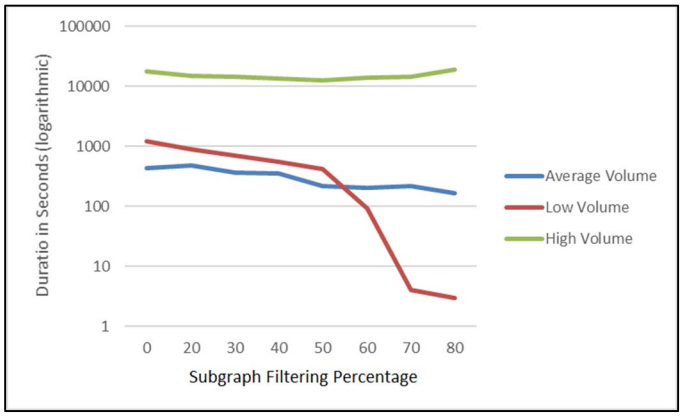

Figure 5. Runtime performance using subgraph filtering (telecom).

The graphs representing data from the non-business-hours traffic (low volume) consists of ~3K vertices/edges per minute. With these low volumes, we observe a 3 orders-of-magnitude speed-up in the running times using maximum filtering.

The graphs representing data from non-peak business-hours traffic (average volume) consists of ~5K vertices/edges per minute. We notice an increase in run-time with a substructure filter of 20%, with the running times decreasing substantially thereafter, flattening out at around 50% where time is cut by more than half, and then decreasing more slowly until 80%. An analysis of where the algorithm is spending its time in the code leads us to believe that the overhead associated with storing and processing the subgraph candidates, and in particular, figuring out which subgraphs are in the "middle" and thus are to be removed, is taking up a significant proportion of the total running time. (Particularly noticeable in this example early on at 20%.)

The graphs representing peak business-hours traffic (high volume) consists of ~10K vertices/edges per minute. We observe a 30% reduction in running times when the subgraph filter is 50%, but then the running times increase up until an 80% threshold. Again, it appears that the overhead of determining and processing the "middle ground" is offsetting the performance savings of filtering these subgraphs.

For the low-volume data, any subgraph filtering percentage up until 80% produces the same normative pattern and same anomalous substructures, except for when the filter is 70%. In this case, we get two different anomalies as the top anomalies (that are not as interesting), but the second-highest scoring anomalies are the original (targeted) anomalies.

For the average-volume data, any subgraph filtering percentage up until 70% produces the same normative pattern and same anomalous substructures. However, at 80% the normative pattern reported is smaller, and the targeted anomalies (as identified when working with the security analysts) are replaced with less interesting anomalies.

For the high-volume data, all substructure filter values produce the same normative pattern and same anomalous substructures. However, as noted in the previous section, the overhead associated with managing the memory requirements of large graphs currently limits the effectiveness of this approach.

As noted earlier, due to security concerns, we are unable to disclose the actual patterns and anomalies discovered. In addition, given the nature of this data, we cannot access typical metrics such as precision, recall, and accuracy. The security analysts can only comment on the interestingness of discovered anomalies. From this limited view of what is reported and the speed with which discoveries can be made, it appears that a subgraph filter of 50% would be a good choice. However, subsequent analysis of the other data sets – ones where we can quantitatively assess the usefulness of the discoveries - should provide more insights.

In summary, at low to moderate volumes, the subgraph filtering is able to significantly reduce the run times, without sacrificing anomaly detection performance. However, with more aggressive volumes, the run times are flat after 50% and in some instances actually worse. Subsequent analysis of the other data sets will provide more insights.

2) Social Network Data

Performing normative pattern discovery and anomaly detection on one days-worth of data, with varying subgraph filtering percentages, we observe the results shown in Figure 6.

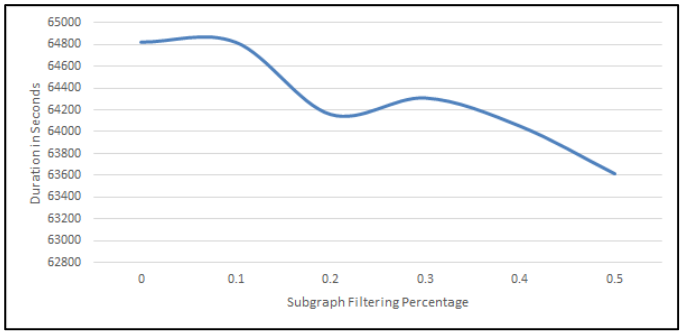

Figure 6. Runtime performance using subgraph filtering (social network).

For all levels of subgraph filtering (i.e., 10-50%), all targeted anomalies are discovered with no false positives. In other words, precision is 100%. In addition, there are no false negatives. Thus, recall is 100%.

In summary, even with 50% subgraph filtering, the reduction in running times is only 2%. Again, as noted in the

previous experiments on high volume data (and somewhat on average-sized data volumes), the overhead of managing the removal of "middle" subgraphs is influencing the overall performance of this approach.

3) Synthetic Data

Performing normative pattern discovery and anomaly detection on the synthetic graphs, with varying subgraph-filtering percentages, we observe the results shown in Figure 7.

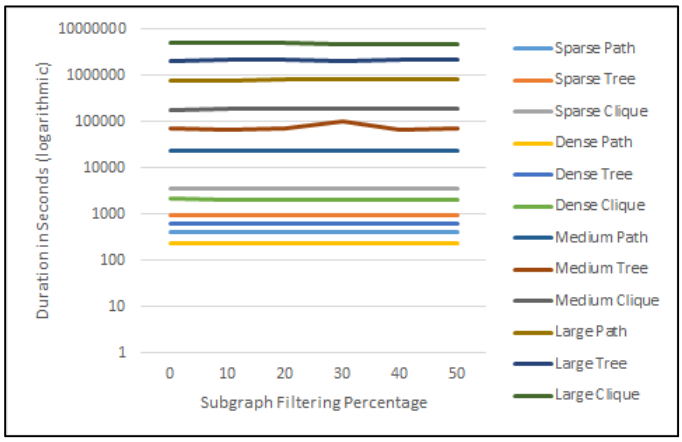

Figure 7. Runtime performance using subgraph filtering (synthetic).

Similar to what is observed on the social network data, for all levels of subgraph filtering (i.e., 10-50%), all targeted anomalies are discovered with no false positives, and no false negatives.

Again, as noted in the previous experiments with high volume data, the reduction in running times is negligible, as a fair amount of overhead minimalizes any potential gains. What is also interesting is that while the running times on the synthetic data are basically flat when it comes to varying levels of subgraph filtering, we do notice an improvement in the real-world (telecommunications) data at average (and lower) volumes, and on the simulated real-world (social network) data. The primary difference between the synthetic data set and the other two data sets is that the synthetic data is very repetitive – the same structure with small deviations is repeated throughout the total graph. However, in the real-world and semi-real-world data sets, data is not so consistent with its variations.

C. Graph Filtering

The following are results of our experiments with varied graph filtering percentages.

1) Telecommunications Data

Performing normative pattern discovery and anomaly detection on 15 minutes of data at a time, with varying graph-filtering percentages, we observe the results shown in Figure 8.

For the graphs representing low volume traffic, we again observe a 3 orders-of-magnitude speed-up in the running times. In fact, starting with a graph filter of 30%, run times are negligible.

For the graphs representing average-volume traffic, we observe similar behavior as was shown with the subgraph filtering, except without the increase (at 20%) associated with the overhead.

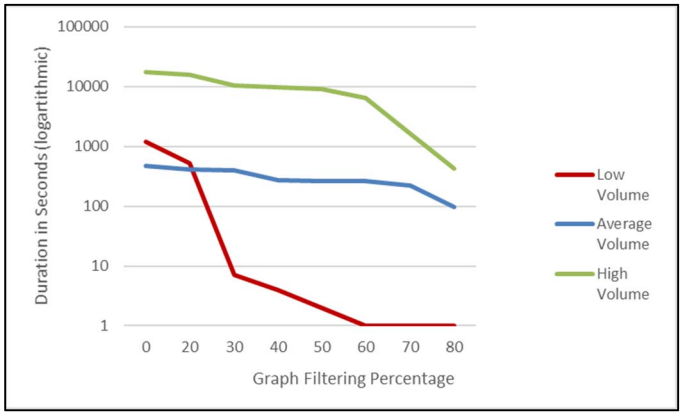

Figure 8. Runtime performance using graph filtering (telecommunications).

For the graphs representing high volume traffic, we observe a 2-orders of magnitude speedup, as again, the issue of memory management when tracking substructure candidates is not present for graph filtering. This is due to the algorithmic difference whereby all reduction is performed on the graph before pattern discovery and anomaly detection is performed.

As shown in Figure 8, for the low-volume data, any graph filtering up until 70% produces the same targeted anomalies, albeit the normative pattern is slightly smaller. At 80% the normative pattern is even smaller, and we lose the targeted anomalies. However, given the significant running times, we can choose an aggressive graph filter of 70% and still get the targeted anomalies in a fraction of the time. (Or even 60% and still get the same normative patterns.)

For the average-volume data, any graph filtering up until 20% produces the same normative pattern and same four anomalous substructures. However, from 30% to 70%, we get the same normative pattern but only three of the anomalous substructures. And at 80% the normative pattern reported is smaller, and the target anomalies (as identified when working with the security analysts) are replaced with less interesting anomalies.

For the high-volume data, graph filtering up to 40% produces the same normative pattern, and anywhere from 50-80% produces a slightly smaller normative pattern. The targeted anomalies are still discovered up to 40%, and then other uninteresting anomalies are reported after that. Fortunately, the best graph filter of 40% for discovering anomalies also results in an order of magnitude speed-up in the running times.

In short, on this domain, graph filtering demonstrates a significant speedup in running times with each increase in the amount of graph filtering. Again, we cannot quantitatively access the effectiveness of our discoveries in this domain (only anecdotally). However, results show that the filtering techniques are effective in a real-world domain.

2) Social Network Data

Applying graph filtering to the social network data results in the running times shown in Figure 9.

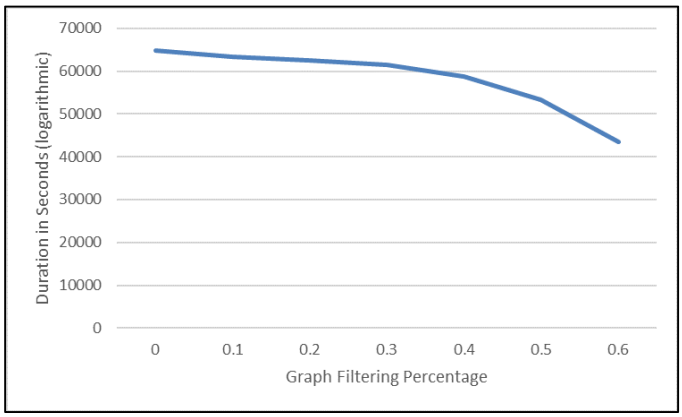

Figure 9. Runtime performance using graph filtering (social network).

As observed, the approach decreases in runtime with each increase in graph filtering, with ~32% reduction at 60% filtering level.

For all levels of graph filtering (i.e., 10-60%), all targeted anomalies are discovered with no false positives. In other words, precision is 100%. In addition, there are no false negatives. Thus, recall is 100%.

3) Synthetic Data

Applying graph filtering to the synthetic data results in the running time trend shown in Figure 10.

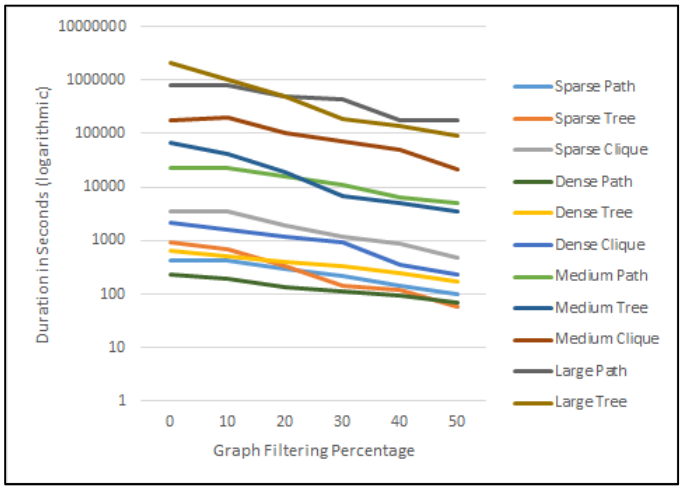

Figure 10. Runtime performance using graph filtering (synthetic).

Unlike with subgraph filtering, we notice a consistent order-of-magnitude (or more) speed-up in performance as the filtering percentage increases.

In terms of precision, at all levels of graph filtering (10-50%), with the exception of the sparse, medium, and large graphs containing cliques, all targeted anomalies are discovered with no false positives, as shown in Figure 11.

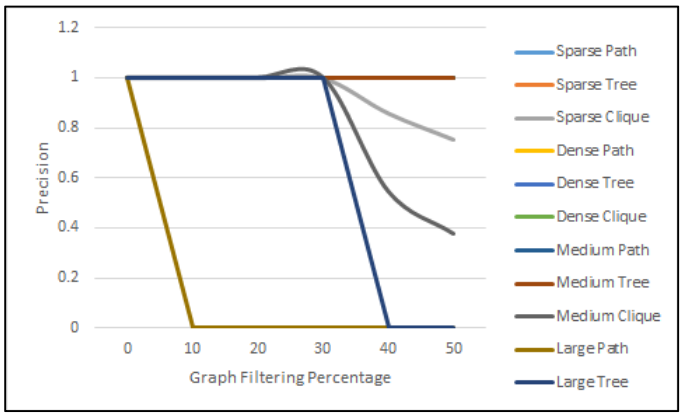

Figure 11. Precision using graph filtering (synthetic).

For the small, sparse cliques, the precision is at 100% up until 10% graph filtering. However, after 10% no anomalous modifications are discovered, and after 40% no anomalous insertions are discovered. (Anomalous deletions are discovered all the way up to 50%.) For the medium-sized graphs composed of cliques, after 40% no anomalous insertions are discovered, and after 10% no anomalous modifications are reported. (Again, all anomalous deletions are discovered at all graph filtering levels.) It should be noted that large cliques (over 10 million vertices and edges) had running times that were too prohibitive to report in this work. Based upon a few initial runs, and what was observed with medium-sized cliques, we expect to see an order of magnitude decrease in the running times, with similar anomaly detection percentages.

In terms of recall, only the dense graphs achieve a 100% recall for all graph filtering percentages, as shown in Figure 12. (Note that due to having similar recall, the sparse experiments are overlapping lines at the top of the chart.) After 10%, recall varies across the different types and sizes of graphs. The worst behavior is exhibited by the graphs containing tree structures, followed by cliques, and then paths. Thus, while we may not be detecting false positives (as noted earlier in the discussion regarding precision), we are losing some of the targeted anomalies above a 10% graph filtering threshold.

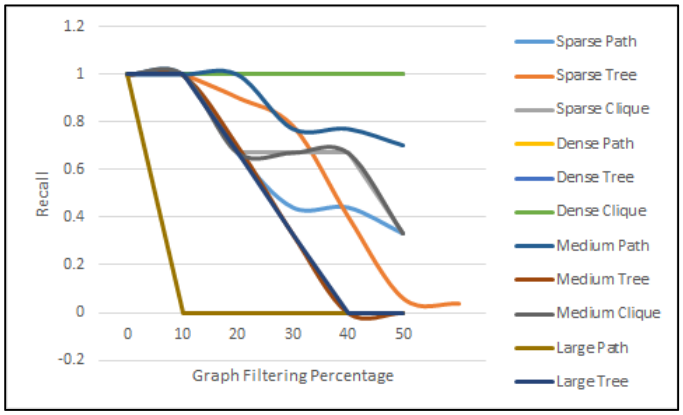

Figure 12. Recall using graph filtering (synthetic).

D. Comparison Approach

As mentioned in Section II, another approach that performs graph-based anomaly detection with some level of graph reductions can be found in the work of Parkinson et al. called GraphBAD [18]. In their approach, they detect relational anomalies, and reducing the input graph by merging vertices that are not directly connected but share a similar set of edges (i.e., perform a form of filtering). However, while the primary goal of the GraphBAD approach is to detect anomalies in security system configurations, they also validate their work on the well-known KDD 99 cup dataset [19].

In order to compare our intelligent filtering approach on graph-based anomaly detection, we will compare what is reported in [18] with runs on the KDD cup data on random records of similar size. (What is specified in [18] does not indicate which actual records were used for their reported experiments.) Using a multi-threaded implementation, they use 8 threads to both reduce the graph and anomaly detection. On a data set of 100,000 random entries/records from the KDD cup data, they are able to complete the task in 6834 seconds (~1.9 hours).

For our approach, we created graphs of each entry in the KDD cup data set, using a topology like the one shown in Figure 13. We then ran our proposed graph-based anomaly detection with graph filtering set to 10% on graphs representing 100,000 entries with an anomaly percentage of 2% (which was the distribution of anomalies in the random 100K of data that was sampled). Our implementation ran in 6.48 hours *using a single thread*, with a *fpr* of 0.085 (8.5%), no false negatives, a *tpr* of 1.0 (100%), and an overall *accuracy* of 0.92. So, while the amount of data was not what one would consider "big data", this was the size used by GraphBAD, allowing us to make a fair comparison of graph-based anomaly detection approaches with filtering.

Table 1 shows a comparison between our GBAD approach and GraphBAD where the anomalous percentage is 2%.

Table 1. Comparison between GBAD and GraphBAD.

Approach	tpr	fpr	tnr	fnr	Accuracy
GBAD	1.0	.085	.915	.00	0.915
GraphBAD	1.0	.17	.83	.00	0.835

The results show that our proposed approach achieves higher accuracy than GraphBAD. In terms of running time, GraphBAD is faster, but it is a multi-threaded approach, and the results were generated using 8 threads. Our approach is a single-threaded application, but can be modified to be multi-threaded and would then be more competitive in terms of run time.

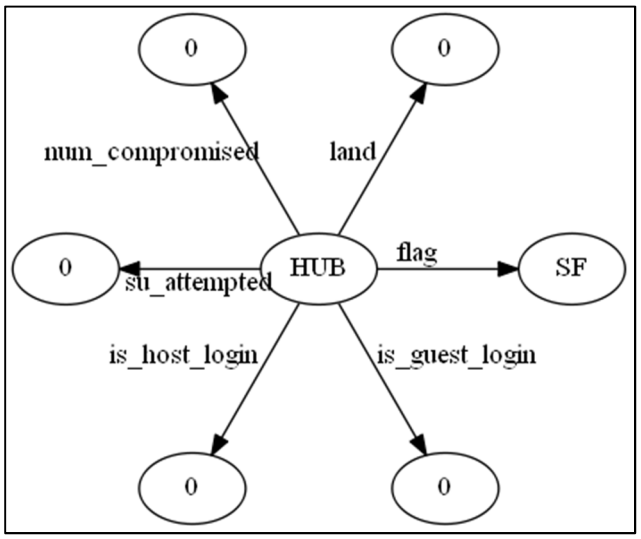

Figure 13. Graph topology of normative pattern in KDD cup data.

VI. Conclusions and Future Work

Given the typical high volume of real-time graph data, extracting knowledge from this data is an even greater challenge. In this work, we presented new methods for filtering graphs so as to address computational challenges with existing graph-based anomaly detection approaches. We then evaluated our approaches with real-world network datasets using the expertise of security analysts, as well as semi-synthetic and synthetic data. In addition, we evaluated our algorithm against another graph-based anomaly detection approach and demonstrated better accuracy. Overall, the results indicate that the moderate amounts of filtering achieves significant reduction in run time without significant loss of accuracy.

Currently, we are continuing to evaluate this approach against other real-world datasets (with known anomalies) in terms of their running times and their accuracy. The primary drawback of the subgraph filtering approach is in the management of the candidate substructure list, which currently resides in memory. In the future, we are planning on investigating other memory management strategies, as well as possibly using graph databases. In addition, we are currently researching other metrics for determining which sub-structures or edges can be considered to be in the "middle". In this work, determining the middle ground is solely based upon frequency. Other possible techniques for determining the best candidates could be (1) a probability of fitness, and (2) the entropy of a substructure or graph edge as it applies to its utility to the graph.

References

[1] M. P. Hampton and M. Levi, "Fast spinning into oblivion? Recent developments in money-laundering policies and offshore finance centres," *Third World Q.*, vol. 20, no. 3, pp. 645–656, Jun. 1999.

[2] W. Eberle and L. Holder, "Streaming Data Analytics for Anomalies in Graphs," *IEEE International Symposium on Technologies for Homeland Security*, April 2015.

[3] W. Eberle and L. Holder, "Anomaly Detection in Data Represented as Graphs," *Intelligent Data Analysis*, Vol. 11, No. 6, 2007.

[4] N. Manuel, "Fraud Detection using Graph Technology", https://neo4j.com/blog/fraud-detection-using-graph-technology/ [blog], November

16, 2018.
[5] T. Schindler, "Anomaly Detection in Log Data using Graph Databases and Machine Learning to Defend Advanced Persistent Threats." *GI-Jahrestagung*, 2017.
[6] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon, "Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs," *Bioinformatics*, vol. 20, no. 11, pp. 1746–1758, 2004.
[7] S. Wernicke, "A faster algorithm for detecting network motifs," in *Proceedings of WABI '05, number 3692 in LNBI*, pp. 165–177, 2005.
[8] N. Ahmed, J. Neville, and R. Kompella, "Network Sampling via Edge-based Node Selection with Graph Induction.", *Purdue University Pubs*, 2011.
[9] N.K. Ahmed, J. Neville, and R. Kompella, "Space-efficient Sampling from Social Activity Streams," in *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, pp. 53–60, 2012.
[10] H. E. Egilmez and A. Ortega, "Spectral anomaly detection using graph-based filtering for wireless sensor networks," 2014 *IEEE International Conference on Acoustics, Speech and Signal Processing* (ICASSP), pp. 1085-1089, 2014.
[11] T. van der Eems, E. Haasdijk, and S. Mongeau, "Outlier Detection using Graph Mining," *Vrije Universiteit Amsterdam*, November 2014.
[12] C.A. Irniger, "Graph Matching: Filtering Databases of Graphs Using Machine Learning Techniques," *Volume 293 of Dissertationen zur Künstlichen Intelligenz*, 2005.
[13] V. Pawar and M. Zaveri, "Graph Based Filtering and Matching for Symbol Recognition," *Journal of Signal and Information Processing*, 9, 167-191, 2018.
[14] A. Lugowski, A. Buluç, J. R. Gilbert and S. Reinhardt, "Scalable complex graph analysis with the knowledge discovery toolbox," *2012 IEEE International Conference on Acoustics, Speech and Signal Processing* (ICASSP), Kyoto, 2012, pp. 5345-5348, 2012.
[15] V. Ioannidis, D. Berberidis, and G. Giannakis, "GraphSAC: Detecting anomalies in large-scale graphs," Cornell University, October 2019.
[16] Holder, Lawrence. [n. d.]. *SUBDUE-Graph Based Knowledge Discovery*. http://ailab.wsu.edu/subdue/datasets/subgen.tar.gz, Last accessed on 03/30/2020.
[17] LSBench. A benchmark for Linked Stream Data processing engines. https://code.google.com/archive/p/lsbench/ .
[18] Parkinson, Simon, et al. "GraphBAD: A general technique for anomaly detection in security information and event management." *Concurrency and Computation: Practice and Experience*, 2018.
[19] KDD Cup 1999 Data. From KDD-99 conference data mining competition. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html