Identifying Anomalies in Graph Streams Using Change Detection

William Eberle
Tennessee Technological University
Department of Computer Science
Cookeville, TN USA
1-931-372-3278
weberle@tntech.edu

Lawrence Holder
Washington State University
School of Electrical Engineering & Computer Science
Pullman, WA USA
1-509-335-6138
holder@wsu.edu

ABSTRACT
Anomaly detection in graph streams requires both the discovery of normative subgraph patterns in the stream and then the identification of subgraphs that are unexpected deviations from the normative patterns. Both of these processes are computationally complex, and therefore we should only update them when necessary. We present an approach based on a change detection metric used for graph sampling that selectively updates the normative patterns only when significant change has occurred. Using a batch processing method on the graph stream, we show that the change detection approach significantly reduces running times while maintaining the accuracy of more exhaustive approaches.

CCS Concepts
• Information systems~Data stream mining
• Computing methodologies~Anomaly detection.

Keywords
Anomaly detection; graph stream; change detection.

1. INTRODUCTION
Many real-world data sets in need of analysis for anomalies (e.g., network traffic for potential intrusions; credit card transactions for fraud; sensor networks for insider threats; etc.), are handled as a stream – a continuous sequence of ordered data. Because this data is relational, the stream can be represented as a sequence of edges added, or modifications made, to a graph. Due to the high volume and velocity of these graph streams, current graph-based approaches for anomaly detection do not scale.

In regards to scaling the first step in anomaly detection, which is to find the normative patterns in the data, there has been some recent research using graph sampling by Ahmed et al. [3]. In their work, the concept of change detection is used to determine the appropriate sampling of a graph in order to discover patterns. We propose that this concept of using a change detection metric can also be used to reduce the run-time complexity associated with discovering anomalous subgraphs in graph streams, while still maintaining similar accuracies. In particular, if we use the definition of a graph-based anomaly as one for which anomalous subgraphs are small deviations from a normative pattern, much like one would find in real-world scenarios such as fraud or insider threats, then a sampling approach based upon detecting change would reduce the time spent discovering what is normative.

One of the key stumbling blocks to accepting graph-based anomaly detection approaches in the real-world is the requirement to discover anomalies in real-time (as in the case of fraud detection). What we are proposing is applicable to any graph-based anomaly detection approach that relies on first discovering normative patterns. In this work, we present a novel graph-based anomaly detection algorithm that handles graph streams presented as batches of new edges. For each batch we calculate a graph property, and if the value of this property has not significantly deviated from previous batches, we do not attempt to rediscover a normative pattern, but use the existing normative pattern as the basis for finding anomalous subgraphs. We evaluate our approach using three diverse data sets, and show that running time can be improved without sacrificing accuracy.

2. RELATED WORK
In order to process data as a stream, one must be able to process elements in a sequential order, restrict the processing space to something manageable, process each element in a timely fashion, and minimize the number of passes over the data [1]. The key issue is that a stream of data is an unbounded sequence of records which clearly cannot be captured and processed in total. While much of the work dealing with data streams has not involved graphs [18], some recent graph streaming research has been applied to a variety of problems and domains [11][22].

Change detection is the process of identifying differences in the state of an object by observing it at different times [28]. Variations of change detection methods have been investigated, primarily for discovering variances in the data in order to predict future data patterns. One approach is the Cumulative Sum (CUSUM) method, which attempts to identify abrupt changes in the statistics of the data [15]. In graphs, change detection involves the discovery of time points at which a graph changes significantly. Various metrics involving graph edit distance [14], graph similarity [21], scan statistics [23], locality-based scan statistics [30], and commute distance [17][4][26], can be used to determine if a graph has changed beyond some threshold.

While the work of Ahmed et al. [3] successfully dealt with the use of graph sampling, using change detection as a means of reducing the time complexity for discovering recurring subgraphs (i.e., sampling) is not a viable option for discovering anomalous subgraphs, as samples may miss potential anomalous subgraphs which are rare. While normative patterns should be "recurring" (and hence, normative), anomalous patterns are not, as one would expect them to occur infrequently. However, the discovery of the normative pattern is necessary in order to discover anomalous patterns. Therefore, we hypothesize that we can use a sampling-based change detection approach to discover normative patterns, which will then aide us in the detection of anomalies.

3. GRAPH-BASED ANOMALY DETECTION

In order to lay the foundation for this effort, we hypothesize that a real-world, meaningful definition of a graph-based anomaly is an *unexpected deviation to a normative pattern*. The importance of this definition (which we more formally define below) lies in its relationship to any deceptive practices that are intended to illegally obtain or hide information [20].

Definition 3.1. A labeled graph $G = (V,E,\lambda)$, where V is the set of vertices (or nodes), E is the set of edges (or links) between the vertices, and the function λ assigns a label to each of the elements in V and E.

Definition 3.2. A subgraph SA is anomalous in graph G if ($0 < d(SA,S) < TD$) and ($P(SA|S) < TP$), where $P(SA|S)$ is the probability of an anomalous subgraph SA given the normative pattern S in G. TD bounds the maximum distance (d) an anomaly SA can be from the normative pattern S, and TP bounds the maximum probability of SA.

Definition 3.3. The anomalous score of an anomalous subgraph SA based on the normative subgraph S in graph G is $d(SA,S) * P(SA|S)$, where the smaller the score, the more anomalous the subgraph.

The advantage of *graph-based anomaly detection* is that the *relationships* between entities can be analyzed for structural oddities in what could be a rich set of information, as opposed to just the entities' attributes. However, graph-based approaches have been prohibitive due to computational constraints, because graph-based approaches typically perform subgraph isomorphisms, a known NP-complete problem. Yet, in order to use graph-based anomaly detection techniques in a real-world environment, we need to take advantage of the structural/relational aspects found in dynamic, streaming data sets.

In order to test our approach, we will implement the publicly-available GBAD test suite[1], as defined by Eberle and Holder [2007], within our proposed approach. 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 MDL approach is used to determine the *best subgraph*(s) as the one that minimizes the following.

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

where G is the entire graph, S is the subgraph, DL(G|S) is the description length of G after compressing it using S, and DL(S) is the description length of the subgraph. The complexity of finding the normative subgraph is constrained to be polynomial by employing a bounded search when comparing two graphs. Previous results have shown that a quadratic bound is sufficient to accurately compare graphs in a variety of domains [19].

For more details regarding the GBAD algorithms, the reader can refer to [12]. In summary, the key to the GBAD approach is that *anomalies* are discovered based upon small deviations from the norm (e.g., insider threat, identity theft, etc.) – not *outliers*, which are based upon significant statistical deviations from the norm.

4. STREAMING APPROACH TO GRAPH-BASED ANOMALY DETECTION

While there has been research that attempts to handle large graphs by segmenting the graph into smaller, individual partitions [5][27], these approaches do not deal with anomaly detection, and the graphs are considered static. Recent research has attempted to use a batch-based approach on graph streams for the discovery of anomalies [2][13]. In order to evaluate our approach of using *change detection* as a means of *reducing the runtime* of batch-based approaches, we will use the general Pattern Learning and Anomaly Detection in Streams framework (PLADS) described in the work of Eberle and Holder [13], as Aggarwal et al.'s [2] work is primarily focused on specific types of undirected graphs. The idea behind the PLADS approach is to utilize information from previous batches while processing the current batch in order to significantly reduce the cost over processing the entire graph, but maintain similar accuracy. We include the pseudocode for PLADS as Algorithm 1.

PLADS accepts as input a stream of batches of edges, where the batches are generated either by partitioning a static graph (such as using a tool like METIS [16]), or fed in over time (where a batch represents data over a specified interval).

ALGORITHM 1. Pattern Learning and Anomaly Detection in Streams (PLADS)

```
Input: batches of new edges, number of batches in window ($N$),
number of normative patterns per batch ($M$)
Output: normative patterns and anomalous subgraphs in current window
1. Wait for first $N$ batches to arrive; process each in parallel
    a. Discover top $M$ normative patterns in each batch.
2. Determine best normative pattern $P$ among $NM$ possibilities.
3. Detect anomalous subgraphs in each of $N$ batches based upon $P$.
4. Evaluate anomalous subgraphs across $N$ batches in current
   window and report most anomalous subgraph(s).
5. Upon arrival of new batch, do
    a. Add new batch to window and remove oldest batch from window.
    b. Determine top $M$ normative patterns from new batch.
    c. Determine best normative pattern $P'$ among all batches in window.
    d. If ($P' \neq P$), each batch discovers new anomalous
       subgraphs based upon $P'$.
    e. Else, only new batch discovers anomalous subgraph(s).
    f. Evaluate anomalous subgraphs across batches and report
       most anomalous subgraph(s).
    g. Repeat.
```

Algorithm 1 is a generic algorithm for applying graph-based anomaly detection methods to streaming data *partitioned into graph batches*. The user can apply any normative pattern discovery techniques and any graph-based anomaly detection

[1] www.gbad.info

algorithms with this approach. To summarize, the PLADS approach processes the batches in parallel, each reporting what they discover as the best normative pattern. PLADS then compares all the choices, and the best normative pattern (i.e., the most frequent across all batches) is chosen. Then each batch is evaluated for anomalous subgraphs *based upon* the best normative pattern. Finally, a similar comparison to what was performed for determining the best normative pattern is executed for reporting the most anomalous subgraph. The process is then repeated on the next batch, and the oldest batch is removed from the set – thus, creating a "sliding window" over the graph batches.

The parameters to Algorithm 1 are defined as follows:

N – number of batches in the sliding window. This will be the initial number of graph batches processed in parallel, and the number of batches considered for determining the normative pattern *and* the anomalous subgraphs as each subsequent batch is processed. A *sliding window* consists of the most recent *N* batches; older batches are discarded.

M – number of normative patterns to retain. This will be the number of normative patterns saved from each batch to compare against other batches.

Z – size of each batch, calculated based upon either the number of nodes/edges generated, or the time window for which the data is collected. Different values of *Z* affect the performance of our algorithm. Values too small risk missing complex patterns. Values too large risk unreasonable run times. We vary the value of *Z* in the experiments depending upon the domain of the data. This value is important because our approach assumes that the batches will be large enough (and numerous enough) to discover the normative pattern.

For more details regarding the above approach, the reader can refer to Eberle and Holder's work [13]. It is relevant to point out that PLADS was implemented in the above work using GBAD for evaluating anomalous subgraphs, whereby PLADS provides the normative pattern to GBAD for its subsequent anomaly detection.

5. CHANGE DETECTION FOR ANOMALY DETECTION

Current approaches using graph properties to determine structural changes in a graph face two primary issues: (1) the scalability of calculating the corresponding metric on a large graph, and (2) the inability to isolate the actual structural change associated with the metric. However, for our approach, we use these graph property metrics on considerably smaller graphs, i.e., batches. Also, we do not need to know what the actual structural changes are, only that a graph has structurally changed. We then use this knowledge to determine if we need to re-calculate the normative pattern. Using this knowledge, the calculation of a normative pattern will be infrequent, and computation time will be greatly reduced.

5.1 Graph Properties

In order to test the effectiveness of this approach, we experiment with different types of graph metrics commonly used when studying graph properties. Each of these graph property metrics represent some measure of the overall structure of a graph, and as such, will enable us to determine whether or not normative structure has potentially changed. Other graph property metrics, and their combinations, will be explored in the future.

Connectedness. Broder et al. [7] define the "connectedness" *C* of a graph based on the set *P*, that contains all pairs (*u*,*v*) such that there is a path from *u* to *v* in *G*, where the cardinality of *P* is divided by the number of possible pairs: $C = |P| / |V|^2$.

Density. Scott [24] defines density *D* as the ratio of the number of actual edges *E* to the maximum possible number of edges: $D = |E| / |V|^2$.

Clustering Coefficient. Boykin and Roychowdhury [7] identify the graph clustering coefficient *CC* to be the average of the clustering coefficients of each vertex:

$$CC = \frac{1}{|V'|}\sum_{i=1}^{|V'|}\frac{2|E_i|}{k_i(k_i-1)}$$

where $|V'|$ is the total number of vertices of degree greater than 1, $|E_i|$ is the number of edges connected to *i*, and k_i is the degree of vertex *i*.

Eigenvalue. The maximum eigenvalue of the graph's adjacency matrix provides a useful graph property, as also noted by Chung et al. in their study of eigenvalues as it relates to graphs [9].

Community Modularity. In general, community modularity is the difference between the fraction of edges inside the community (in our case, a graph batch) and the fraction of edges expected by a random version of the network, preserving the degree distribution of the nodes [29]. Using the Clauset-Newman-Moore community modularity detection method for large graph networks [10], we iteratively compare two communities that contribute maximum positive value to global modularity, and merge them, until all communities are detected, returning the community modularity of the graph.

Number of Triangles. In social sciences and related fields, triads, or subgraphs of three nodes, have shown to aide in the understanding of social groups [8]. For this metric, we will simply count the number of triangles in the corresponding graph.

Entropy. We use the Sole and Valverde [25] definition of entropy whereby instead of just analyzing the degree distribution, P_k, in a graph, they use the remaining degree, which is the number of edges leaving a vertex and not arriving to a vertex. This new distribution, *q(k)*, is defined as:

$$q(k) = \frac{(k+1)P_{k+1}}{k}$$

They then use this distribution to calculate the following entropy measure:

$$H(q) = -\sum_{k=1}^{N} q(k)\log(q(k))$$

The entropy of the graph is a measure of uncertainty. A higher degree of entropy implies a wider range of unique patterns.

Each of the above metrics measures some aspect of the *structure* of a targeted graph. The graph property values calculated, which we call *GP*, indicate some measure of structure. Thus, the difference between a graph G_1's graph property GP_1, and a graph G_2's graph property GP_2, is an indicator that there is a structural difference between graphs G_1 and G_2. We will use the knowledge of that difference using a form of cumulative summation to determine when the normative pattern in a stream needs to be re-evaluated.

5.2 Change Detection Algorithm

In order to evaluate the effectiveness of each of the previously discussed graph properties, we modify Algorithm 1 to include the calculation and evaluation of a specified graph property metric. Algorithm 2 shows our proposed algorithm with **modifications and additions in bold**.

The use of μ and σ allows a cumulative summation to determine when a new normative pattern needs to be discovered due to a change beyond a standard deviation, without having to resort to a user-defined threshold. Even though we are currently only determining a single best normative pattern (Steps 2 and 5d), we can modify the algorithm to consider multiple normative patterns, and anomalies to them. However, we have only focused on the top normative pattern here, mainly to reduce the sources of variance in the comparison of the different metrics for change detection.

ALGORITHM 2. Pattern Learning and Anomaly Detection in Streams **using Change Detection** (PLADS-**CD**)

```
Input: batches of new edges, number of batches in window ($N$),
number of normative patterns per batch ($M$)
Output: normative patterns and anomalous subgraphs in current
window
1.  Wait for first $N$ batches to arrive; process each in parallel
    a.  Discover top $M$ normative patterns in each batch.
    b.  Calculate and store graph property GP metric for
        each batch.
2.  Determine best normative pattern $P$ among $NM$ possibilities.
3.  Detect anomalous subgraphs in each of $N$ batches based upon
    $P$.
4.  Evaluate anomalous subgraphs across $N$ batches in current
    window and report most anomalous subgraph(s).
5.  Upon arrival of new batch, do
    a.  Add new batch to window and remove oldest batch
        from window.
    b.  Calculate and store graph property metric GP' for
        new batch.
    c.  Calculate mean $\mu$ and standard deviation $\sigma$ based
        on graph property metrics in current window.
    d.  If $|GP' - \mu| > \sigma$
        i.   Determine top $M$ normative patterns from new
             batch.
        ii.  Determine best normative pattern $P'$ among all
             batches in window.
        iii. if ($P' \neq P$), each batch discovers new anomalous
             subgraphs based upon $P'$.
        iv.  Else, only new batch discovers anomalous
             subgraph(s).
    e.  Else, new batch discovers anomalous subgraph(s)
        using P from previous batches.
    f.  Evaluate anomalous subgraphs across batches and
        report most anomalous subgraph(s).
    g.  Repeat.
```

It is important to note that we never assume that the normative pattern is the same between each evaluation of patterns, i.e., the sliding window. The PLADS approach (Algorithm 1) recalculates the normative pattern each time a new batch of data is introduced into the window. With our proposed approach (Algorithm 2), we use graph property metrics to determine whether or not the graph structure has changed enough to warrant a recalculation of the normative pattern, and the amount of change is immaterial (i.e., it can be a completely new normative pattern).

6. EVALUATION DATA SETS

Three different data sources are used to evaluate our approach: e-commerce review data, social network data, and network traffic. Each source varies in graph structure, and will be used to generate graph stream batches of various sizes.

6.1 Berlin

The Berlin SPARQL Benchmark[2] is a synthetic generator built around an e-commerce use case in which a set of products is offered by different vendors and consumers post reviews about the products. The Berlin data model contains the following classes: *Product, ProductType, ProductFeature, Producer, Vendor, Offer, Review,* and *Person*. We chose to represent this data as a graph where classes are nodes, associations are edges, and attributes are linked to their corresponding class (node). Figure 1 shows our graph representation, where every class vertex has an associated month, day, and year (shown only for the Review class in the figure).

The Berlin data generator creates an XML file of products along with associated information, offers, and reviews. The user specifies the number of products generated, and all products, offers, and reviews are generated between the dates of March 15, 2008 and June 19, 2008. We create batches based upon offer and review times, *where each batch* consists of all offers and reviews for products in a specified "time window" (e.g., batch 1 contains all product offers and reviews made on March 15, batch 2 contains all product offers and reviews made on March 16, etc.). In order to evaluate our approach on a volume that is more representative of a graph stream, we specify increasing amounts of products and batches. Also, given that "anomalies" inherent in the Berlin data are uninteresting, and do not represent the types of anomalies that one might find in real data of this type (i.e., products being sold and associated reviews), we seeded the data with unusual years (i.e., publication dates that would appear suspicious), and unusual publishers (i.e., people that normally do not publish in the associated venue).

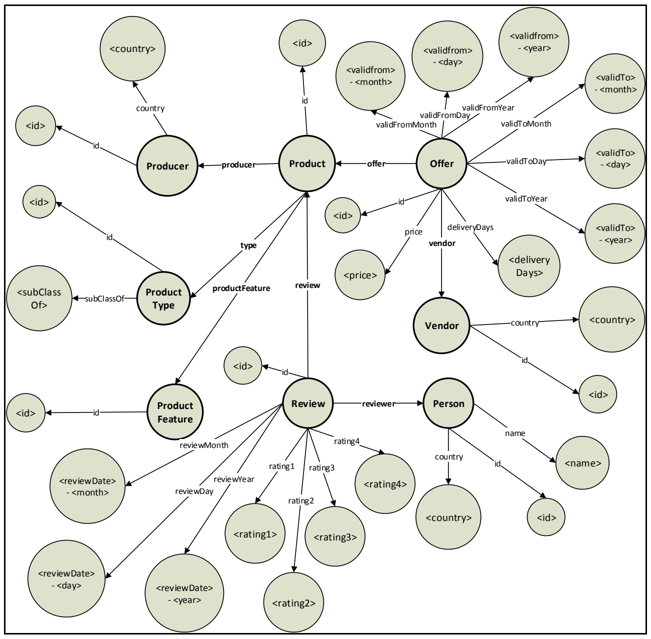

Figure 1. Graph representation of Berlin data.

[2] http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/

6.2 LSBench

The Linked Stream Benchmark (LSBench)[3] data generator generates data that represents users as the static data, and their actions as a data stream, including GPS locations, posts, and photo albums, as well as "like"s and "know"s. We generate RDF triples of varying sizes and time periods, that contain user information, and their associated locations, devices used, postings, photos, likes, and whom they know.

We represent the LSBench objects as vertices, and actions as edges. Figure 2 presents the graph topology of the streaming data. Varying the number of users between 1000 and 4000 results in 1-2M edges per batch, depending on the time period of the captured data, which varies from every day to every minute. In order to evaluate our approach on a volume that is more representative of a graph stream, we specify increasing numbers of users. And, similar to the Berlin data, as there are no intrinsic anomalies created by the LSBench generator, we will seed various types of anomalies, including unusual likes (e.g., a person liking a group that does not fit their usual profile) and unusual follows (e.g., a person joining a forum that is different from the forums they typically follow).

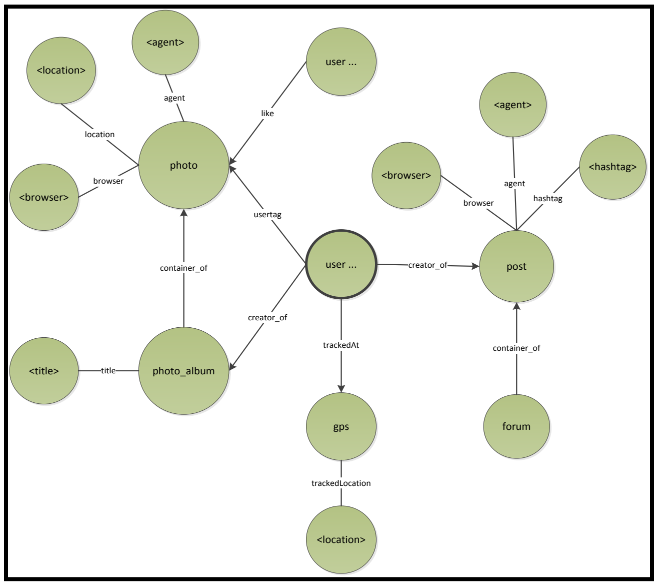

Figure 2. Graph topology of streaming LSBench data.

6.3 CAIDA

The CAIDA (Center for Applied Internet Data Analysis) AS data set[4] represents the topology of the internet as the composition of various Autonomous Systems. Each of the AS units represents routing points through the internet.

One example of network traffic collected by CAIDA represents a dynamic denial-of-service (DDOS) attack at a single location. We represent this data as a graph composed of ~25K vertices and ~100K edges, with each AS depicted as a vertex and an edge indicating a peering relationship between the AS nodes. Figure 3 shows a portion of the AS graph, where the rectangle indicates the normative pattern and the emboldened edge indicates the anomalous structure found by GBAD. However, even if we represent the data as a graph stream, we are only processing a total of ~100K edges. In order to evaluate our approach, we need to scale up to graph streams that may produce millions, if not billions, of edges. For this particular data set, we accomplish this by replicating the data. Then, to experiment using the approach identified in Algorithm 1, we use the METIS tool in order to partition the graph into smaller, roughly equal-sized subgraphs [16]. These subgraphs are then streamed in to our approach as batches of edges.

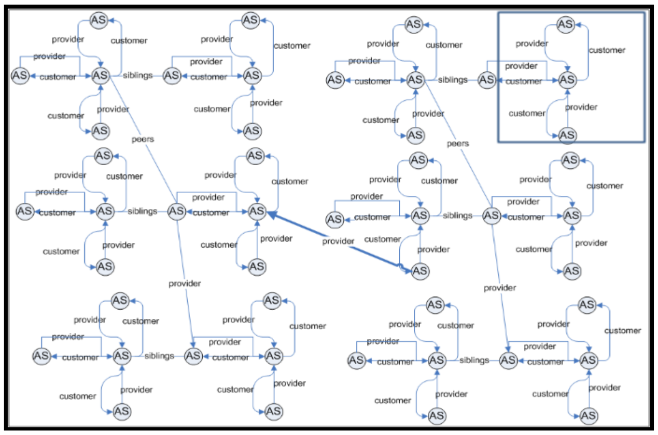

Figure 3. Normative pattern (square) and anomaly (bold) discovered in the CAIDA dataset.

7. EXPERIMENT RESULTS

First, we will present summarized results across the different data sets, and then we will present some detailed results where we vary the values for N, M, and Z, as well as the hardware, to gain some further insights into our proposed approach.

Table 1 shows a summary of the results from Algorithm 1 (shown as "baseline") versus the different change detection approaches using Algorithm 2, for stream rate (edges per second), recall, and false-positive-rate, with best approach in **bold**. In these results we calculate "Recall" as the percentage of targeted anomalous subgraphs that are discovered (i.e., accuracy), and "False Positive Rate" (FP Rate) as the number of reported anomalies minus the true positives, divided by the number of reported anomalies, multiplied by 100 (i.e., the inverse of precision). In addition, it should be noted that we chose a *TD* value of 0.2 and *TP* is bound by the number of instances of the *SA* as compared to instances of other subgraphs (per *Definition 3.2*). Again, the choice of *TD* is important because a value too small may miss the anomalous substructures, and a value too large may result in too many false positives. All of the results shown in Table 1 were performed using a Debian 6.0, 64-bit server with 24GB of RAM and 8 cores @ 2.27 GHz.

First, one notices that all of the change detection approaches implemented in Algorithm 2 are faster than the baseline approach (i.e., able to process more edges per second), which was to be expected as fewer re-evaluations of the normative pattern occur. In some instances, we realize over a 96% speedup. However, for the Berlin data set, while the *clustering coefficient* and *triangles* approaches are faster than the baseline, this is because there are never any future normative pattern evaluations beyond the initial set of N batches. The *clustering coefficient* never deviates enough to warrant a re-evaluation of the normative pattern, and there are no *triangles* in the data. This could be addressed by a re-evaluation of the normative pattern after X iterations (or X amount of time) even if the change detection approach is not

[3] https://code.google.com/archive/p/lsbench/

[4] www.caida.org/data/active/as-relationships

detecting any change. (It should also be noted that the overhead of calculating the graph properties is included within the overall edges/second rate – but will be further reported shortly in a more detailed analysis.) Second, none of the change detection approaches used in Algorithm 2 achieve 100% recall across all of the data sets. While none achieve 100% recall in the Berlin data, *density* and *entropy* discover all of the anomalous subgraphs with the LSBench data, and *connectedness*, *density*, and *community* have 100% recall with the CAIDA data. Third, no change detection approach has the lowest FP Rate across all of the data sets. The *clustering coefficient* and *triangles* approaches have lower FP Rate than the baseline on the Berlin data. In fact, all but *community* achieve a 0% FP Rate on the CAIDA data.

Table 1. Summary of results.

Data Source	Approach	Edges/Sec	Recall	FP Rate
berlin	baseline	141	**100**	2.62
	connected	216	93	5.35
	density	221	90	5.05
	clustering	257	46	**1.40**
	eigenvalue	224	86	5.02
	community	217	83	5.04
	triangles	**277**	46	**1.40**
	entropy	220	93	4.25
lsbench	baseline	157	**100**	**17.82**
	connected	216	96	18.67
	density	196	**100**	19.31
	clustering	**241**	96	19.28
	eigenvalue	232	88	21.82
	community	241	96	18.76
	triangles	239	80	21.06
	entropy	191	**100**	20.39
caida	baseline	168	**100**	1.72
	connected	243	**100**	**0**
	density	**274**	**100**	**0**
	clustering	229	75	**0**
	eigenvalue	243	75	**0**
	community	227	**100**	0.03
	triangles	264	75	**0**
	entropy	235	**100**	**0**

In terms of accuracy, we miss some targeted anomalous subgraphs because of our definition of an anomaly, which is based upon subgraph differences when compared to the normative pattern. In this case, sometimes the structure of a graph does not deviate enough between batches, leading to an older normative pattern being used for anomaly detection, when in reality the normative pattern has changed and was needed to discover the targeted anomalous subgraph. Across all of the change detection methods, we realize an average reduction of normative pattern discoveries of ~75% (i.e., only ¼ of the number of normative subgraph discoveries takes place).

In terms of the high false-positive rate when analyzing the LSBench data, this can be attributed to the graph structure of that particular domain. The CAIDA data set is fairly consistent in structure, with only a few unique labels and subgraphs. The Berlin data generator produces a consistent structure, with growth of vertices and edges linear to the user-specified number of products. However, the LSBench data generator is much more random, as not every user gets the same number of connections to forums, user groups, etc. In addition, the generator is very parametric, where the user must specify a level of noise greater than zero (i.e., there will always be noise).

We now take a closer look at more detailed results using just the LSBench data. Here we vary the number of batches in the sliding window (N) from 3 to 7. The total number of edges processed in a complete run range from 3,868,915 up to 13,296,977. The batch sizes (Z) are based upon the total number of edges divided by the size of an individual window, for which we chose 4 minute windows and 6 minute windows, and varied the overlap between each window from no overlap to 5 minutes of overlap (for a 6 minute window). We also varied M (number of normative patterns to retain) from 3 to 20, with no perceivable differences.

Figures 4-8 represent the results of these experiments on the LSBench data, averaged over all values of Z. We also used a different set of hardware (48 cores from two Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, and a total of 125GB RAM), so as to measure the effect that more memory and faster CPUs would have on this approach.

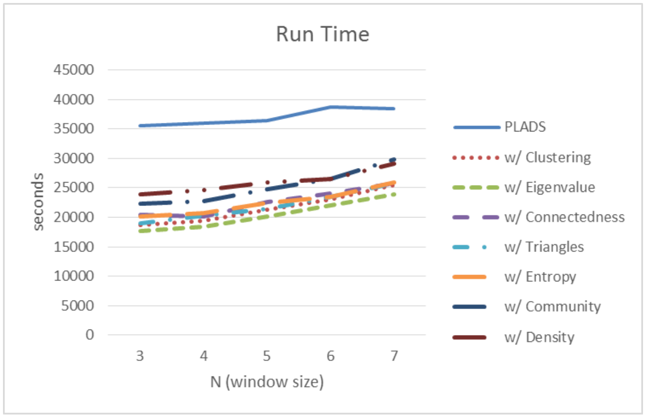

Figure 4. Average run time.

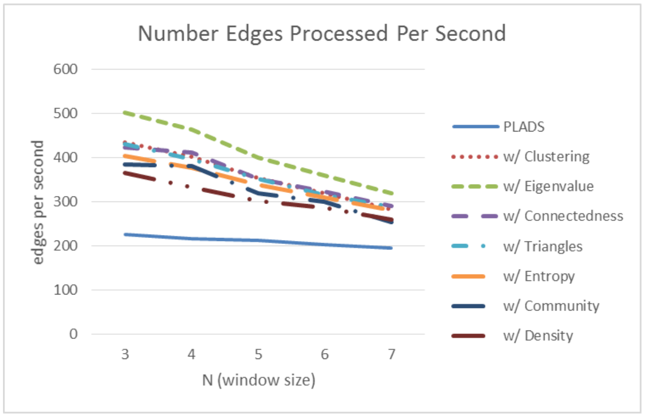

Figure 5. Average number of edges processed per second.

From these detailed results, we observe the following. First, as shown in Figure 4 and Figure 5, when N is small, our approach detects anomalous subgraphs in almost half the time. However, as N increases, the running times and edge processing rates converge (i.e., running times are similar). There are several reasons for this. Intuitively, as the number of batches in the window (N) increases, more and more of the complete graph is being processed. In addition, our implementation of Algorithm 2 is file-based, rather than memory-based, and the more batches there are, the more normative patterns and anomalous subgraphs need to be compared (file I/O overhead). Finally, in addition to implementing a memory-based version, we could also add comparison checks such that if the normative pattern has not changed between batches, a graph match is not performed again, thereby avoiding redundant work.

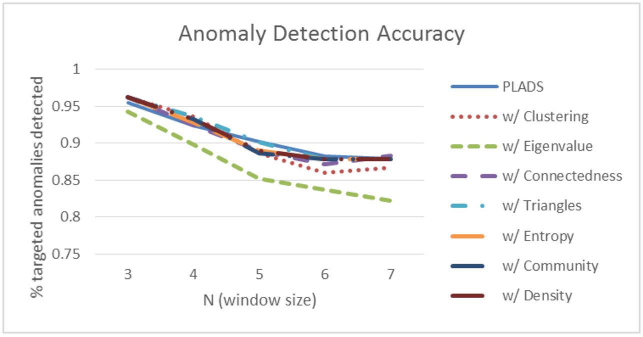

Figure 6. Average % of targeted anomalies discovered.

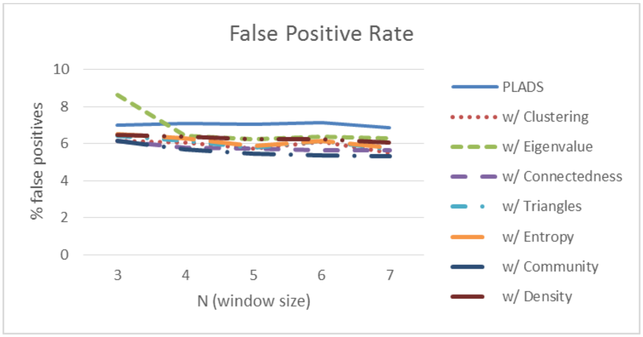

Figure 7. Average false positive rate.

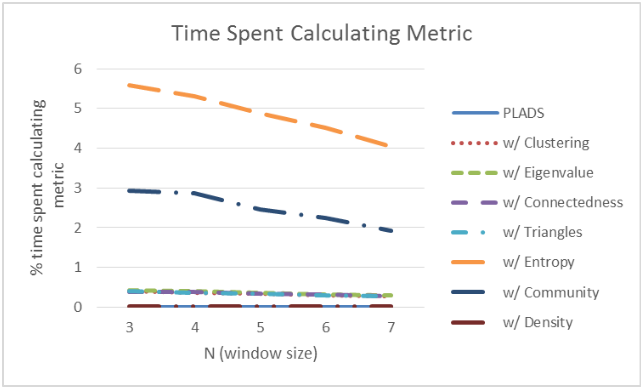

Figure 8. % of total time spent calculating metric.

Second, in terms of discovering our targeted anomalies (as shown in Figure 6), except for when we use *eigenvalue* and *clustering* as our metric for determining when the normative pattern needs to be re-evaluated, accuracy is similar to PLADS. In other words, using a graph property metric does not significantly affect our ability to discover anomalies.

Third, except for when the *eigenvalue* metric is used with small values for *N*, the false positive rate is better than the baseline PLADS approach, and fairly consistent, no matter what value for *N* is chosen (Figure 7). This was one of the more surprising results. What this tells us is that when we calculate the normative pattern with every new batch (as is done with the PLADS approach), noise is affecting the normative pattern discovery. The result is a "local" normative pattern, leading to the non-discovery of the targeted anomalous subgraph. Whereas, using the graph metric to determine when to re-evaluate for a normative pattern, our approach recognizes that the overall graph structure has not really changed, and thus the "global" normative pattern is retained for anomaly detection.

Fourth, Figure 8 shows the percentage of the total time that was spent calculating each metric. Obviously, calculating *entropy* took the longest, followed by *community*. However, the other metrics took less than 0.5% of the total time, and calculating density was less than 0.02% on average. So, in general, little overhead is needed to achieve a significant speed-up towards the discovery of anomalous subgraphs.

Finally, we did observe a significant speed-up between the two platforms – one with 24GB of RAM and 2.27 GHz cores, and one with 125 GB of RAM and 2.50 GHz cores. Runs on the latter platform were significantly faster, with an average speed-up of ~20%, and up to 36% faster in the best case. We also notice that (regardless of the hardware) more edges are processed per second with a larger batch size (i.e., going from a batch size of 4 minutes up to 6 minutes). This is probably due to less overhead because the *number* of batches decreases as the *size* of the batches increases. In addition, the more overlap there is between batches, the faster it runs, albeit at the sacrifice of accuracy. The *speed-up* is probably due to less volatility in the overall graph structure, thus less normative pattern evaluations; while the decrease in *accuracy* could be related to multiple anomalous subgraphs appearing in the same batch, making them less anomalous due to their repetition.

In general, using change detection within a parallel, batch streaming approach like PLADS allows us to maintain similar anomaly detection accuracies with low false positive rates. Based upon our results, in order to significantly reduce the run-time and thus increase the edge processing throughput, we need to minimize the number of batches (*N*). Whereas, accuracy and false-positive rates appear to be unaffected by the choice of *N*, with accuracy results that are linear once *N* reaches 5 and false positive rates that are consistent for any value of *N*.

8. CONCLUSIONS AND FUTURE WORK

Integrating change detection into a streaming approach for graph-based anomaly detection results in a significant speed-up in processing, with most instances resulting in slightly less accuracy, and a minor increase in false-positive rates. However, for several of the change detection methods, as the size of the data sets increase, accuracy reaches baseline levels of 100%, and false-positive rates are actually better than the baseline. Considering that our target is real-world graph streams (i.e., continuous, unending data), these results are promising. In the future, we will develop an incremental approach that processes only the stream of *graph changes* over time, where normative patterns and anomalies are updated only as necessary based on the impact of the changes. This will allow us to remove the current "boundary issues" associated with anomalous subgraphs that could span graph batches. Further speedup is possible by exploiting inherent parallelism across batches. We plan to evaluate our algorithms on other datasets, including a Nokia mobile data set, and actual traffic flows collected from our institution-level network, especially in terms of "keeping up" with such high-speed streams.

Another future objective is to investigate other approaches that can better handle noise – i.e., non-targeted anomalies. As discussed earlier, if we increase the size of the processing window, we can reduce the number of false positives albeit at the sacrifice of speed. In addition, since results vary across the different data sources, we will also explore additional metrics and combination of metrics to ideally identify an indicator that works across domains.

9. ACKNOWLEDGMENTS

This material is based on work supported by the National Science Foundation (NSF) under Grant Nos. 1318913 and 1318957. Support for CAIDA's Internet Traces is provided by the NSF, the US Department of Homeland Security, and CAIDA Members.

10. REFERENCES

[1] Aggarwal, C. and Wang, H. 2010. *Managing and mining Graph Data*. vol. 40, pp. 487–513.

[2] Aggarwal, C., Zhao, Y., and Yu, P. 2011. Outlier detection in graph streams. *In Proceedings of the 27th International Conference on Data Engineering (ICDE)*, Hannover, Germany, pages 399–409.

[3] Ahmed, N., Neville, J., and Kompella, P. 2014. Network sampling: from static to streaming graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, Vol 8, Issue 2, No.7, pp. 1-56.

[4] Albano, J. and Messinger, D. 2012. Euclidean commute time distance embedding and its application to spectral anomaly detection. *SPIE Defense, Security, and Sensing*. International Society for Optics and Photonics.

[5] Andersen, R., Chung, F., and Lang, K. 2006. Local graph partitioning using pagerank vectors. *In Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 475–486. IEEE Computer Society.

[6] Boykin, P.O., and Roychowdhury, V.P. 2005. Leveraging social networks to fight spam. *Computer (Long. Beach. Calif)*. vol. 38, pp. 61–68.

[7] Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R, Tomkins, A., and Wiener, J. 2000. Graph structure in the web. *Computer Networks*, Volume 33. pp. 309-320.

[8] Chin, G., Marquez, A., Choudhury, S., and Feo, J. 2012. Scalable triadic analysis of large-scale graphs: multi-core vs. multi-processor vs. multi-threaded shared memory architectures. In *Proceedings of the 2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD '12). IEEE Computer Society, Washington, DC, USA, 163-170.

[9] Chung, F., Lu, L., and Vu, V. 2003. Eigenvalues of random power law Graphs. *Annals of Combinatorics*, volume 7. pp. 21-33.

[10] Clauset, A., Newman, M., and Moore, C. 2004. Finding Community Structure in Very Large Networks. *Physical review E*, Volume 70, Issue 6, American Physical Society (publisher).

[11] Dasgupta, A., Kumar, R., and Sivakumar, D. 2012. Social sampling. In *KDD*, pp. 235-243.

[12] Eberle, W. and Holder, L. 2007. Anomaly detection in data represented as graphs. *Intelligent. Data Analysis*, vol. 11, pp. 663–689.

[13] Eberle, W. and Holder, L. 2015. Scalable anomaly detection in graphs. *Intelligent Data Analysis*, vol. 19, no. 1, pp. 57–74.

[14] Gao, X., Xiao, B., Tao, D., and Li, X. 2010. A survey of graph edit distance *Pattern Anal. Appl.*, vol. 13, pp. 113–129.

[15] Gustafsson, F. 2000. *Adaptive filtering and change detection*, John Wiley & Sons, Ltd.

[16] Karypis, G. 2011. *METIS: A software package for partitioning unstructured graphs*. Version 5.0.

[17] Khoa, N. and Chawla, S. 2010. Robust outlier detection using commute time and eigenspace embedding. *Advances in Knowledge Discovery and Data Mining*. Springer Berlin Hiedelberg. p422-434.

[18] Krempl, G., Spiliopoulou, M., Stefanowski, J., Žliobaite, I., Brzeziński, D., Hüllermeier, E., Last, M., Lemaire, V., Noack, T., Shaker, A., and Sievi, S. 2014. Open challenges for data stream mining research. *ACM SIGKDD Explor. Newsl*. vol. 16, no. 1, pp. 1–10.

[19] Kukluk, J., Holder, L., and Cook, D. 2004. Algorithm and Experiments in Testing Planar Graphs for Isomorphism. *Journal of Graph Algorithms and Applications*, Volume 8, Number 3.

[20] Levi, M. 2001. Money laundering and its regulation. *The ANNALS of the American Academy of Political and Social Science*, vol. 582. pp. 181–194.

[21] Papadimitriou, P., Dasdan, A., and Garcia-Molina, H. 2010. Web graph similarity for anomaly detection. *J. Internet Serv. Appl.*, vol. 1, pp. 19–30.

[22] Pavan, A, Tangwongsan, K., Tirthapura, S., and Wu, K.L. 2013. Counting and sampling triangles from a graph stream. *VLDB* 6, 14, pp. 1870-1881.

[23] Priebe, C.E., Conroy, J.M., Marchette, D.J., and Park, Y. 2005. Scan statistics on Enron graphs. *Comput. Math. Organ. Theory*, vol. 11, pp. 229–247.

[24] Scott, J. 2000. *Social Network Analysis: A Handbook.* SAGE Publications, Second Edition. pp 72-78.

[25] Sole, R. and Valverde, S. 2004. *Information theory of complex networks: on evolution and architectural constraints*. Springer-Verlag, Lecture Notes Physics. vol 650, pp. 189–207.

[26] Sricharan, K. and Das, K. 2014. Localizing anomalous changes in time-evolving graphs. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM.

[27] Sun, J., Faloutsos, C., Papadimitriou, S., and Yu, P.S. 2007. Graph-Scope: parameter-free mining of large time-evolving graphs. *In Proceedings of the13th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, San Jose, CA, pages 687–696. ACM.

[28] Tran, D.-H., Gaber, M.M., and Sattler, K.-U. 2014. Change detection in streaming data in the era of big data: models and issues. *In ACM SIGKDD Explorations Newsletter - Special issue on big data*. no. 1, pp. 30–38.

[29] Viera, V., Xavier, C., Ebecken, N., and Evsukoff, A. 2014. Performance evaluation of modularity based community detection algorithms in large scale networks. *In Mathematical Problems in Engineering*, Volume 2014.

[30] Wang, H., Tang, M., Park, Y., and Priebe, C.E. 2014. Locality statistics for anomaly detection in time series of graphs. *IEEE Trans. Signal Processing*, vol. 62, pp. 703–717.