# Analysis of Pattern Learning and Anomaly Detection on Streams

Ramesh Paudel
Tennessee Technological University
Cookeville, Tennessee 38501
rpaudel42@students.tntech.edu

***Abstract*— Graph represents complex relationships among heterogeneous data. Using this underlying complex relationships we can find anomaly in the data. The anomaly detection in static and relatively small graph is done by using a compression-based measure. It first finds normative patterns, and then analyzed the close matches to the normative patterns to indicate potential anomalies. However, while this approach has demonstrated its effectiveness in a variety of domains, the issue of scalability has limited this approach when dealing with domains containing millions of nodes and edges. Pattern Learning and Anomaly Detection on Streams, or PLADS is scalable and effective algorithm for detecting anomalies in real-world streaming data. It has developed a partitioning approach that partitions the graph as it streams in over time and maintains a set of normative patterns and anomalies biased toward the set of normative patterns found in the current time window. In this paper, we will analyze PLADS algorithm against know parameter and verify thus obtained theoretical model by implementing it to detect anomaly in health care data set.**

***Index Terms*— Anomaly detection, graph mining, dynamic graphs, streaming data, parallel graph mining**

## I. INTRODUCTION

Popular methods for discovering patterns are usually various machine learning approaches that use techniques such as classification, clustering, nearest neighbors, and statistics. These methods focuses mostly on the attributes of entities and ignored the relationship between entities. Recent research efforts have involved the representation of complex data as a graph, in order to analyze the relational structure in the data. More importantly, most of these data are huge and required scalable approach to mine and learn pattern and anomalies. Graph-based anomaly detection, called GBAD [1], uses compression- based measure to find normative patterns, and then analyzed the close matches to the normative patterns to determine if there is an unexpected deviation to that normative pattern. These unexpected deviation are called anomaly. However, while this approach has demonstrated its effectiveness in a variety of domains [2][22][23][24], the issue of scalability has limited this approach when dealing with domains containing millions of nodes and edges.

Also there are many domain where graphs are dynamic, i.e. changes to the graph are streaming in over time. This further complicates the analysis, because we cannot just analyze a static graph, but would need to analyze snapshots of the graph over time. This streaming graph scenario not only changes the structure of graph but also change the anomaly along with time, i.e. both the structure of the graph and the anomaly are constantly evolving. This streaming graph scenario can be tracked by updating the current set of patterns and anomalies based on only the changes to the graph, rather than repeated analyses on the large graph snapshots. Pattern learning and anomaly detection in streams (PLADS) described in detail in section III address this problem.

Pattern Learning and Anomaly Detection on Streams, or PLADS [3], is scalable to real-world data that is streaming, as well as maintains reasonable levels of effectiveness in detecting anomalies. It has developed a partitioning approach that partitions the graph as it streams in over time and maintains a set of normative patterns and anomalies biased toward the set of normative patterns found in the current time window [3]. In this paper, we will analyze the cost model of PLADS algorithm and verify this theoretical model by implementing it for detecting anomaly in health care data set.

## II. BACKGROUND

The PLADS approach we are discussing here is based on static graph-based anomaly detection (GBAD) [1]. GBAD is an unsupervised approach, that use SUBDUE, graph- based knowledge discovery system[5] to discover the substructure. To fully understand PLADS we have to understand the mechanism of substructure discovery in SUBDUE and how GBAD use it to find the anomalous substructure.

### *A. SUBDUE*

Input to SUBDUE is in the form of a labeled, directed multi-graph. A substructure is a connected subgraph within the graphical representation. The substructure discovery algorithm used by SUBDUE is a computationally constrained beam search [6]. The algorithm begins with the substructure matching a single vertex in the graph. Each iteration through the algorithm selects the best substructures and expands the instances of these substructures by one neighboring edge in all possible ways. The algorithm retains the best substructures in a list, which is returned when either all possible substructures have been considered or the total amount of computation exceeds a given limit. The best substructure is the one that minimizes the dataset description length i.e. that minimizes [5]

$$DL(S) + DL(G|S)$$

Here $S$ is the discovered substructure, $G$ is the input graph, and $DL(S)$ is the number of bits required to encode the discovered substructure. $DL(G|S)$ is the number of bits required to encode $G$ after it has been compressed using $S$. Thus discovered substructure is also called normative substructure.

### B. GBAD

The idea behind GBAD approach is to find anomalies in graph-based data where the anomalous substructure in a graph is part of (or attached to or missing from) a normative substructure. The anomaly in graph can be defined as follows [7].

**Definition:** Given a graph $G$ with a normative substructure $S$, a substructure $S\prime$ is considered anomalous if the difference d between $S$ and $S\prime$ satisfies $0 < d <= X$, where $X$ is a user- defined threshold and $d$ is a measure of the unexpected structural difference between two sub-graphs of a graph.

The above definition have a real importance in its relationship to fraud detection where a person or entity attempting to commit fraud, will do so in a way to hide their illicit behavior by conveying their actions as close to legitimate actions as possible [7]. There are three general categories of anomalies that GBAD finds: insertions, modifications and deletions. Insertions is when an extra vertex or edge appears; modifications is when a vertex or edge that is different than was expected; and deletions is when the expected vertex or edge is missing[7]. GBAD uses SUBDUE to discovers the best substructure, or normative pattern, in an input graph. GBAD then compresses the graph using the normative pattern, i.e., replacing each instance of the normative pattern with a single node with a new label. Then, GBAD is executed on this compressed graph to again find normative patterns and anomalies. This process can continue for multiple iterations to find more and more normative patterns, and anomalies to them, throughout the graph, and at different levels of abstraction as the graph is further compressed. The input of GBAD is a single static graph file.

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. This is very important aspect of GBAD. 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 [3]. In summary, GBAD looks for those activities that appear to match normal/legitimate/expected transactions, but in fact are structurally different.

## III. Pattern Learning and Anomaly Detection on Streams

The advantage of GBAD is that it provide the efficient approach for discovering structural and relational anomalies. However, when it comes to big data, the scalability is an issue. GBAD is limited to static domains (where the data point does not change over time), and data sets that are relatively small in size. To address the issue of scalability of GBAD in dynamic streams and in big graphs, a novel approach called *Pattern Learning and Anomaly Detection on Streams*, or PLADS is purposed. PLADS is not only scalable to real-world data that is streaming, but also maintains reasonable levels of effectiveness in detecting anomalies. PLADS take smaller, individual partitions (i.e., a segment of the data that is processed individually, in parallel with other partitions). PLADS can not only provide similar accuracy but do it in a fraction of the time [3]. The PLADS algorithm accepts the input as a set of N graph partitions either by partitioning a static graph (that could have been the input of GBAD), or fed in over time.

### A. PLADS (input graph partitions) [3]

1) Process $N$ partitions in parallel
    a) Each partition discovers top $M$ normative patterns
    b) Each partition waits for all partitions to discover their normative patterns.
2) Determine best normative pattern $P$ among $NM$ possibilities.
3) Each partition discovers anomalous substructures based upon $P$.
4) Evaluate anomalous substructures across partitions and report most anomalous substructure(s).
5) Process new partition
    a) If oldest partition(s) has exceeded a threshold $T$ (based upon criteria such as the number of available partitions or the time-stamped-age of the partition), remove partition(s) from further processing.
    b) Determine top $M$ normative patterns from new partition.
    c) Determine best normative pattern $P\prime$ among all active partitions.
    d) If ($P\prime \neq P$), each partition discovers new anomalous substructures based upon $P\prime$.
    e) Else, only new partition discovers anomalous substructure(s).
    f) Evaluate anomalous substructures across partitions and report the most anomalous substructure(s).
    g) Repeat.

Figure 1 shows the visual representation of PLADS.

## IV. Related Work

Anomaly detection in large graph is a complex problem, specially when it comes to scalability. And even more complex when we are talking about a ”stream” where processing of the graph has to be done on one or few edges at a time. Since, they are the only edges that changes with time. The earlier work on anomaly detection in large static graph by Cook and Noble [10] defined anomalies as

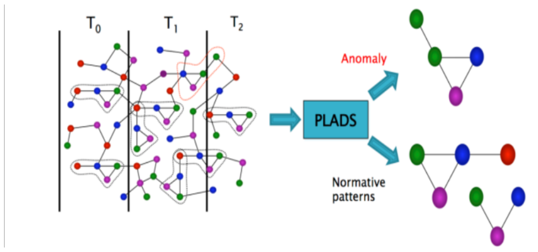

Fig. 1. Working scenario of PLADS over time [25]

structural outliers. They proposed that after compressing the graph based on normative patterns, the remaining structure was considered anomalous. The work by Akoglu et al. [9] addressed anomaly detection in large, weighted graph. They propose the algorithm called oddball. This approach discover several new rules (power laws) in density, weights, ranks and eigenvalues that seem to govern the so-called neighborhood sub-graphs and use these rules for anomaly detection. It carefully choose features, and design oddball, so that it is scalable (for a graph up to 1.6 million nodes) and also it is a unsupervised approach. But their target was to identify only anomalous nodes. There have been some work on the area of big graph mining by using MapReduce technology. One such approach called Pegasus, addresses the problem of graph mining using a distributed approach, but not in the context of anomaly detection [11]. All of the above works are done in static graph.

One potential solution to handling very large graphs is to view the graph as a stream and processing the graph one, or a few edges, at a time [3]. Previous work in this area has provided a few different approaches to handle graph streams. One approach is to use what is called a semi-streaming model as a way of studying massive graphs whose edge sets cannot be stored in memory. For example, Feigenbaum et al.s work presents semi-streaming constant approximation algorithms for unweighted as well as weighted matching problems, along with a further algorithmic improvement for the bi-partite graphs [12]. By considering a set of classical graph problems in their semi-streaming model, they were able to demonstrate that certain approximations to the problems can be achieved. There are work that has generalized this approach to different graph problems, such as the shortest paths in directed graphs, and used intermediate temporary streams as a means of resolving the space issues. For e.g. Demetrescu et al. [13] showed that the use of intermediate temporary streams is powerful enough to provide effective space passes tradeoffs for natural graph problems. Sun et al. [15] proposed algorithm for identifying similar node (Neighborhood formation) and finding abnormal nodes in bipartite graphs. Their algorithms compute the neighborhood for each node using random walk with restarts and graph partitioning; and use neighborhood information to identify abnormal nodes. Basically, these approaches propose a trade off between the available internal memory and the number of passes it requires.

Early work also included the statistical analysis of data stream for the detection of anomalies. Duffield et al. limited the data flow, in order to deal with the issue of scalability, through sampling, which has the potential for missing anomalies [14]. Cormode et al. focused their work on the goal of heavy-change detection (i.e., drastic flow changes in a network), which misses the small changes/deviations characteristic when one is attempting to go unnoticed in their illicit behavior [16]. In addition, neither of these previous approaches deal with data represented as a graph.

Clustering massive graph streams poses significant challenges because of the complex structures which may be present in the underlying data. The massive size of the underlying graph makes explicit structural enumeration very difficult. Aggarwal et al. use a technique for creating hash-compressed micro-clusters from graph streams [17]. The compressed micro-clusters are designed by using a hash-based compression of the edges onto a smaller domain space. There have been attempts to mine frequent closed subgraphs in non-stationary, time varying data streams. Bifet et al. proposed an approach called AdaGraphMiner, maintains only the current frequent closed graphs, utilizing estimation techniques with theoretical guarantees [18]. Empirical experiments have demonstrated the effectiveness of this approach on graph streams representing chemical molecules and structural representations of cancer data.

There are also work done to discover outliers in massive network streams. Using what is called a structural connectivity model, some researchers have attempted to handle the issue of sparseness in massive networks by dynamically partitioning the network [19], while others have exploited the graph structure of a network using various partitioning approaches [20]. Using techniques such as reservoir sampling methods that compress a graph stream, one can search for structural summaries of the underlying network. The goal of this type of outlier detection is to identify graph objects which contain unusual bridging edges, or edges between regions of a graph that rarely occur together. However, all of the approaches so far have not addressed the issue of scalability associated with performing graph- based anomaly detection on big data graphs represented as streams. While some approaches have detected outliers in graph streams, their objective is to identify unusual clusters of subgraphs in the graph by analyzing the statistical nature of the existence of edges, as opposed to discovering anomalies in the structure of a graph, or graph stream. In addition, while some work has attempted to discover anomalous subgraphs using an ensemble-based approach [21] based on the GBAD approach [1], that type of approach does not address the issue of scalability.

## V. Task Decomposition

The main work of task decomposition is done before running the algorithm. We make the smaller, individual partitions of the single graph by dividing it into segment of the data. And each partition will be feed into PLADS based on partition creation time stamp. Based on the algorithm in section III, we can see that at most $N$ partition can be run in parallel, where $N$ is equal to available processor, and each of these partition discovers top $M$ normative patterns. Based on $NM$ possible normative pattern the best normative pattern $P$ is chosen and send it back to all partition. Now, each partition will find the anomalous structure and PLADS will evaluate all the anomalous substructure and report the most anomalous substructure. This process can be summarized in the task dependency graph in fig 2.

–

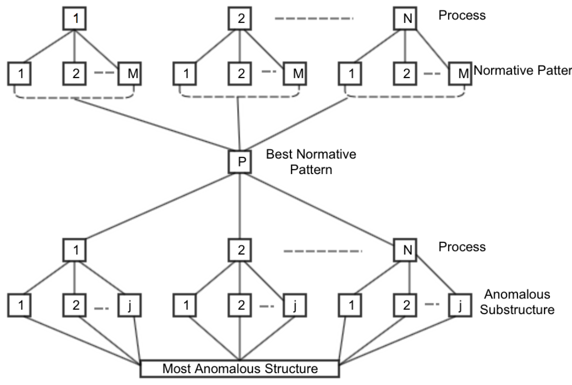

Fig. 2. Task Dependency Graph for PLADS

Once PLADS find most anomalous substructure from N partition, PLADS then processes one partition at a time, handling the data as a sliding time window, removing the oldest partition and processing a new partition. The new partition will find its set of $M$ normative patterns. Now, from all active partition (old and the new one) the new best normative pattern $P\prime$ is determined. If $P\prime$ is not equal to old normative pattern $P$, each active partition will discovers anomalous substructure based on $P\prime$. These anomalous substructure are evaluated and most anomalous substructure is returned like on the previous iteration. If $P\prime$ is same as $P$ then, only new partition find the anomalous substructure. Every time new normative pattern is discovered, the algorithm update the anomalous substructure also. If we have unlimited amount of processor then $N$ will be equal to number of available processor. Hence, the maximum concurrency of PLADS is equal to the number of partitions.

## VI. Theoretical Computational Complexity Analysis

PLADS is based on GBAD for graph substructure discovery and GBAD is computationally expensive [4]. GBAD is a sequential version of PLADS. To understand the computational complexity of PLADS we have to understand GBAD algorithm, since each process will run GBAD and communicate the result of GBAD to each other.

### A. Sequential Computational Complexity

GBAD, the sequential version of PLADS, spends most of its time performing graph matching (SUBDUE graph match). The unconstrained graph match is exponential in the number of graph vertices but GBAD employs constraints that make program more scalable. The total running time of GBAD can be expressed as the number of search nodes expanded during graph matches throughout the entire discovery process [4]. Let,

- $L$ = user-defined limit on the number of substructures considered for expansion, for the experimentation the value of $L$ is kept constant in 16.
- $nv$ = number of vertices in the input graph
- $nsub$ = total number of substructures that can be generated
- $gm$ = user-defined maximum number of partial mappings considered during each graph match
- $n_{inst}$ = total number of instances of a give substructure

Since the algorithm spends most of its time performing graph matching, the total running time of the algorithm can be expressed as

$$N1 = nsub \times n_{inst} \times gm$$

Considering an upper bound time complexity, assume the input graph is a fully connected graph, where the number of neighbors for a given vertex is $(nv - 1)$, the maximum size of a substructure generated in iteration $i$ of the algorithm is $i$ vertices, and the number of vertices which have already been considered in previous iteration $(i-1)$. Hence, the total number of vertices that can be expanded is $((nv-1)-(i-1))$. Therefore, the total number of substructures that can be generated is

$$nsub = \sum_{i=1}^{L} i \times ((nv-1)-(i-1))$$

The total number of instances needed to be compared for a given substructure involves the instances of the substructure itself and the instances of the substructure's parent. For a substructure with $i$ vertices, the maximum number of non-overlapping instances is $\frac{nv}{i}$. Since we consider an upper bound case, let the maximum number of non-overlapping instances be $nv$. Hence, the total number of instances needed to be compared for a given substructure is

$$n_{inst} = nv \times (L-1)$$

This shows that by placing a limit on maximum number of partial mappings considered during each graph match i.e. $gm$ and number of substructures considered for expansion i.e $L$, the time complexity for the graph matching is polynomial in number of vertices i.e. $nv$. If either of the two limits $L$ or $gm$ are removed, the complexity of the discovery algorithm becomes exponential in number of vertices $nv$. This proves that upper bound on the complexity of GBAD is a function of the number of vertices in the input graph.

### B. Parallel Computational Complexity

The parallel implementation in PLADS is done using forking. Since fork is done in the same physical machine the communication cost is ignored. Also the graph partitioning is done before the algorithm starts so the overhead of task decomposition is also not considered. The parameters to the PLADS algorithm are defined as follows [3]:

- $N$ number of partitions in the sliding window. This will be the initial number of graph partitions processed in parallel, and the number of partitions considered for determining the normative pattern and the substructures that are anomalous as each subsequent partition is processed.
- $M$ number of normative patterns to retain. This will be the number of normative patterns saved from each graph partition to compare against other graph partitions.

It should be mentioned that the size of each partition (i.e., number of vertices and edges) is not necessarily the same, but the ideal case of graph partitioning should have even size across all the partition. Since PLADS employ a sliding window approach for processing each partition, each iteration of the algorithm will have $N$ number of partition. The first iteration of a sliding window, considering the algorithm in section III (A) is step(1) to step(4). Each process in this iteration run graph matching algorithm across the partition to find the $M$ normative substructure (Step 1), anomalous substructure across the same partition (Step 3). There is a barrier synchronization after step (1) and step (3) so the maximum amount of time required to complete those steps is equal to the time required by the process that finishes last. Usually, the partition which have the highest number of vertices should take the most time, since the upper bound on the complexity of GBAD is a function of the number of vertices in the input graph. If we are able to partition the graph in equal number of vertices and edges then the wait time due to synchronization will be zero. In addition to that, Step(2) have to do $N \times M$ graph matches to find the best normative substructure $P$ among the possible $N \times M$ possible substructure, and step(4) do graph match on all anomalous substructure across $N$ partition. We cannot really tell the number of anomalous substructure discovered by each process. Hence, cost of the starting iteration of PLADS is given by

$$nsub \times n_{inst} \times M$$

In ideal scenarios the number of vertices in each partition are equally divided. In that case we can express $nsub$ and $n_{inst}$ in terms of N as follows:

$$nsub = \sum_{i=1}^{L} i \times ((\frac{nv}{N} - 1) - (i-1))$$

$$n_{inst} = \frac{nv}{N} \times (L-1)$$

After completing step (4), in each iteration oldest partition (in terms of time it arrived in the sliding window) is removed and the new partition is considered i.e. the number of partition in each iteration is alway N. Master process run GBAD to find the top $M$ normative patterns in the partition and best normative pattern $P\prime$ among all active partition is calculated. If $P\prime \neq P$, each partition discover new anomalous substructure based on $P\prime$, else only new partition discovers anomalous substructure. If $P\prime = P$, master process don't have to create a child processes. It can discover anomalous substructure itself. Either way the cost for this iteration is similar to the cost of first iteration. Because no matter how many process are running in parallel, the runtime to complete this step is upper bound of the number of vertices in each partition. This whole process is repeated until there are no partition left. So the total cost of PLADS is given by:

$$T_{PLADS} = N_{iter} \times (nsub \times n_{inst} \times M)$$

where, $N_{iter}$ is the total number of iteration required until no partition are left and is given by

$$N_{iter} = (TotalPartition - N) + 1$$

After each process finishes finding the normative substructure they have to wait till the other finishes and the they can communicate each other. The best runtime is directly affected by the processor that take longest time. Usually if we see GBAD, the runtime is directly affected by the number of vertices. So we have to make sure each partition gets equal number of vertices to get best concurrency in runtime among all the processes. The ideal time process endure due to load imbalance is the main overhead. Based on above equation, to see how the change in value of $N$ affects the runtime, we have plotted the graph which is shown in fig. 3. Similarly,

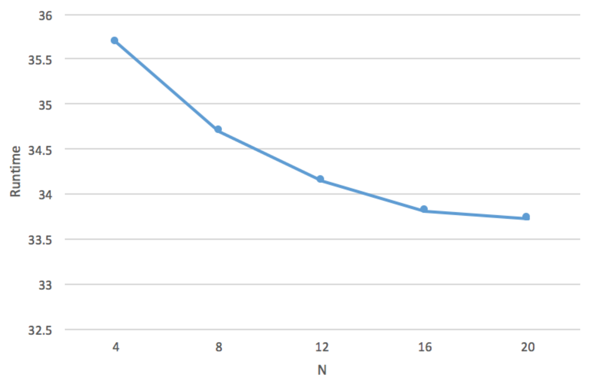

Fig. 3. Theoretical PLADS runtime for different value of N

we have plotted the graph for different value of M. This is shown in fig. 4.

## VII. Experiment using Medicare Data

We apply PLADS to CMS Linkable 20082010 Medicare Data Entrepreneurs Synthetic Public Use File (DE-SynPUF) dataset [8]. Our experimental setup consists of parsing the required data from the DE-SynPUF dataset, constructing a graph that contains the data for each beneficiary from all the claim tables, and processing the resulting graph with a PLADS tool. In order to create the graph input file, we will create a parser (written in the python programming language) that will read the CMS data and build the graph. The snapshot of the sample graph we generated is shown in fig. 5. All of our experiments are performed using a Debian 6.0, 64-bit server with 24GB of RAM and 8 CPUs. We

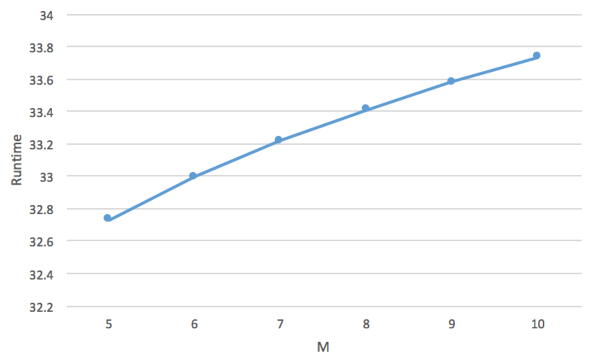

Fig. 4. Theoretical PLADS runtime for different value of M

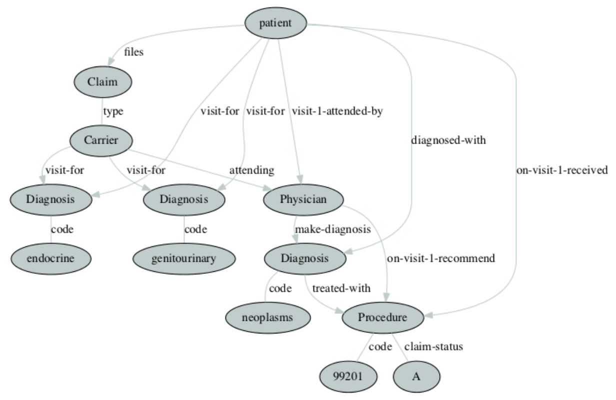

Fig. 5. Sample graph used for testing

created two different graphs, one has data for diabetic patient from February 1, 2009 to February 25 2009, over the period of 25 days as a test dataset. The single graph have 79,111 vertices and 118,067 edges. We created one partition per day so that the single graph will be divided into 25 partition. Again it should be noted that the size of each partition (i.e., number of vertices and edges) is not necessarily the same. Another dataset has data for diabetic patient from January 18, 2009 to March 08, 2009, over the period of 50 days. The single graph have 161,273 vertices, and 240,990 edges. We created one partition per day so that the single graph will be divided into 50 partition in this case. The choice of data is just random. The sample of anomalies reported by PLADS is shown in fig. 6.

We first ran PLADS using different value of $N$ while keeping the normative pattern $M$ constant i.e. 10. With the increase of number of process in a sliding window ($N$), the runtime for the algorithm will reduce. We ran PLADS in two different problem size as mentioned above. First one has 50 partitions. The second one, has almost half number of vertices and edges in the graph and is divided into 25 partitions. We choose the above two dataset to see if the algorithm show similar behavior with different number of partitions. The runtime for both problem size are shown in table I. For 50 partitions it almost took twice to finish the algorithm in comparison to the problem size of 25 partition. The table shows varying numbers of partitions (25 and 50)

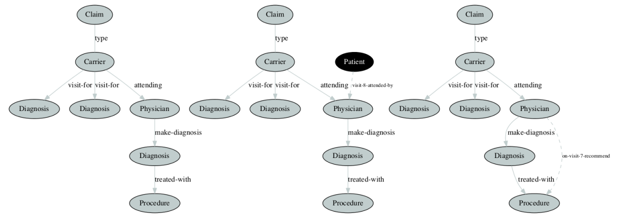

Fig. 6. PLADS Result showing normative substructure, anomaly with extra vertex and edge

and varying numbers of partition retained in the sliding window ($N$), while normative pattern $M$ is always 10. The data of table I are plotted in a graph and the result are shown in fig. 7. With both problem size, PLADS shows similar characteristics.

TABLE I
PLADS Runtime for different N

| Partition | M | N | Runtime (sec) |
|---|---|---|---|
| 50 | 10 | 4 | 8639 |
| | | 8 | 8261 |
| | | 12 | 7842 |
| | | 16 | 7532 |
| | | 20 | 7230 |
| 25 | 10 | 4 | 4033 |
| | | 8 | 3191 |
| | | 12 | 3032 |
| | | 16 | 2311 |
| | | 20 | 1877 |

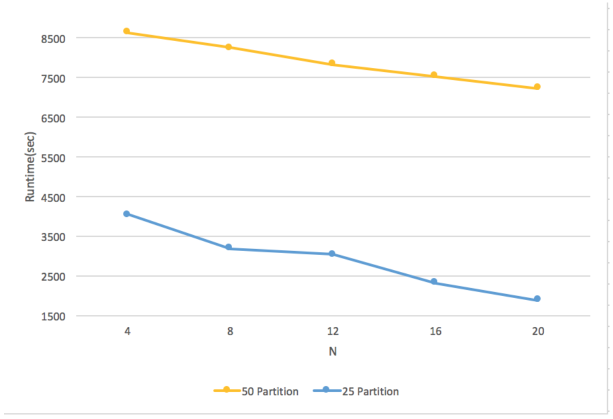

Fig. 7. PLADS runtime for different N

Also we also ran PLADS using different value of M in the graph with 25 partition to see how the number of normative pattern retained affect the runtime. The results of PLADS are shown in table II. It should be noted that we observed that varying the value for M does have some effect on the PLADS performance. With less number of normative retained the runtime of algorithm improves, but this might have affect in the accuracy of the algorithm to discover the anomaly. There is a trade off between the accuracy and the choice of $N$ and $M$. Figure 8 illustrates the scalability improvement of PLADS while reducing the value of $M$.

TABLE II
PLADS RUNTIME FOR DIFFERENT M

| Partition | Runtime(Seq) | N | M | Runtime(PLADS) | Speedup |
|---|---|---|---|---|---|
| 25 | 56215 | 20 | 5 | 1550 | 36.26 |
| | | | 6 | 1537 | 36.57 |
| | | | 7 | 1655 | 33.96 |
| | | | 8 | 1684 | 33.38 |
| | | | 9 | 1684 | 33.38 |
| | | | 10 | 1877 | 29.94 |

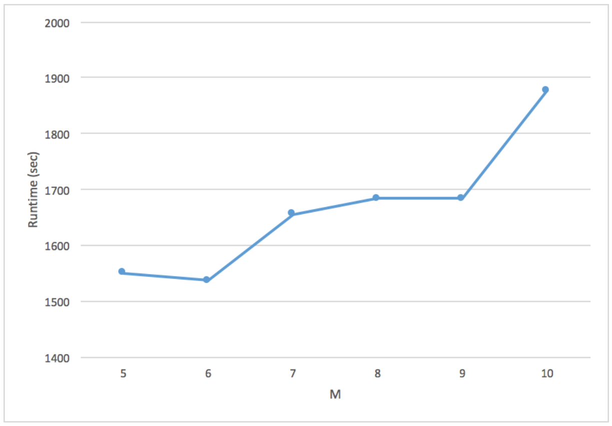

Fig. 8. PLADS runtime for different value of M

This particular work is attempting to analyses the performance of PLADS based on it's parameter $N$ and $M$. If we compare the runtime give by theoretical model as shown in figure 3 and 4 with figure 7 and 8 obtained from running anomaly detection in DE-SynPUF dataset we can see the similar behavior.

## VIII. CONCLUSIONS

The approach used in Static graph analysis are not good enough to handle streaming graph. They are not scalable to handle the bigger size as well as the changing nature of the graph. PLADS is a method for analyzing graphs using a parallel partitioning approach that can discover anomalous substructures. We implemented PLADS ourselves and have demonstrated that running-time shows the huge improvement in comparison to serial graph-based anomaly detection approach. Analyzing the algorithm using real-time Medicare claim scenario we found that after each process finishes discovering the normative or anomaly substructure, all the process have to wait till the other finishes their work. This explicit synchronization is one of the major factor slowing down the algorithm. The best runtime is directly affected by the process that take longest time i.e. the process that is handling the partition with most number of vertices finishes last (Because the upper bound complexity of GBAD is equal to the number of vertices). So it is highly recommended that each partition gets equal number of vertices to get best concurrency in runtime among all the processes. There is no standard way to do the partition in PLADS, we recommend that there should be the standard way for task decomposition.

$N$ and $M$ are the parameter that directly affect the runtime of PLADS. However, the choice of $N$ and $M$ also affect the accuracy of PLADS for discovering anomalies. So the choice of $N$ and $M$ should be done to get the right balance between accuracy and speedup.

## REFERENCES

[1] Eberle, William, and Lawrence Holder. ”Anomaly detection in data represented as graphs.” Intelligent Data Analysis 11.6 (2007): 663-689.
[2] Eberle, William, Jeffrey Graves, and Lawrence Holder. ”Insider threat detection using a graph-based approach.” Journal of Applied Security Research 6.1 (2010): 32-81.
[3] Eberle, William, and Lawrence Holder. ”A partitioning approach to scaling anomaly detection in graph streams.” Big Data (Big Data), 2014 IEEE International Conference on. IEEE, 2014.
[4] Cook, Diane J., Lawrence B. Holder, and S. Djokok. ”Scalable discovery of informative structural concepts using domain knowledge.” IEEE Expert 11.5 (1996): 59-68.
[5] D. Cook and L. Holder. Graph-based data mining. IEEE Intelligent Systems 15(2), 32-41, 1998
[6] Holder, Lawrence B., Diane J. Cook, and Surnjani Djoko. ”Substucture Discovery in the SUBDUE System.” KDD workshop. 1994.
[7] Eberle, William, and Lawrence Holder. ”Discovering structural anomalies in graph-based data.” Data Mining Workshops, 2007. ICDM Workshops 2007. Seventh IEEE International Conference on. IEEE, 2007.
[8] CMS. 2016a. Medicare claims synthetic public use files. [Online; accessed 11-August-2016].
[9] Akoglu, Leman, Mary McGlohon, and Christos Faloutsos. ”Oddball: Spotting anomalies in weighted graphs.” Advances in Knowledge Discovery and Data Mining (2010): 410-421.
[10] Noble, C. and Cook, D. 2003. Graph-Based Anomaly Detection. SIGKDD.
[11] Kang, U. and Faloutsos, C. 2012. Big Graph Mning: Algorithms and Discoveries. SIGKDD Explorations, Volume 14, Issue 2.
[12] Feigenbaum, J., Kannan, S., McGregor, A., Suri, S. and Zhang, J. 2005. On Graph Problems in a Semi-Streaming Model. Theoretical Computer Science.
[13] Demetrescu, Camil, Irene Finocchi, and Andrea Ribichini. ”Trading off space for passes in graph streaming problems.” ACM Transactions on Algorithms (TALG) 6.1 (2009): 6.
[14] Duffield, Nick, Carsten Lund, and Mikkel Thorup. ”Properties and prediction of flow statistics from sampled packet streams.” Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment. ACM, 2002.
[15] Sun, Jimeng, et al. ”Neighborhood formation and anomaly detection in bipartite graphs.” Data Mining, Fifth IEEE International Conference on. IEEE, 2005.
[16] Cormode, Graham, and S. Muthukrishnan. ”What's new: Finding significant differences in network data streams.” IEEE/ACM Transactions on Networking (TON) 13.6 (2005): 1219-1232.
[17] Aggarwal, Charu C., Yuchen Zhao, and Philip S. Yu. ”On clustering graph streams.” Proceedings of the 2010 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics, 2010.
[18] Bifet, Albert, et al. ”Mining frequent closed graphs on evolving data streams.” Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2011.
[19] Aggarwal, Charu C., Yuchen Zhao, and S. Yu Philip. ”Outlier detection in graph streams.” Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011.
[20] Leskovec, Jure, et al. ”Statistical properties of community structure in large social and information networks.” Proceedings of the 17th international conference on World Wide Web. ACM, 2008.
[21] Parveen, P., Evans, J., Thuraisingham, B., Hamlen, K., Khan, L. 2011. Insider Threat Detection Using Stream Mining and Graph Mining. Privacy, Security, Risk and Trust (PASSAT), IEEE International Conf. on Social Computing (SocialCom), pp.1102-1110.
[22] Mookiah, Lenin, William Eberle, and Lawrence Holder. ”Detecting suspicious behavior using a graph-based approach.” Visual Analytics Science and Technology (VAST), 2014 IEEE Conference on. IEEE, 2014.

[23] Eberle, William, Lawrence B. Holder, and Beverly Massengill. ”Graph-Based Anomaly Detection Applied to Homeland Security Cargo Screening.” FLAIRS Conference. 2012.

[24] Chaparro, Cameron, and William Eberle. ”Detecting Anomalies in Mobile Telecommunication Networks Using a Graph Based Approach.” FLAIRS Conference. 2015.

[25] Eberle, William, and Lawrence Holder. ”Incremental Anomaly Detection in Graphs.” Data Mining Workshops (ICDMW), 2013 IEEE 13th International Conference on. IEEE, 2013.