Outlier Detection in Graph Streams

Charu C. Aggarwal [*1], Yuchen Zhao [#2], Philip S. Yu [#3]

[*] *IBM T. J. Watson Research Center*
Hawthorne, NY 10532, USA
[1] charu@us.ibm.com

[#] *University of Illinois at Chicago*
Chicago, IL, USA
[2] yzhao@cs.uic.edu
[3] psyu@cs.uic.edu

***Abstract*—A number of applications in social networks, telecommunications, and mobile computing create massive streams of graphs. In many such applications, it is useful to detect structural abnormalities which are different from the "typical" behavior of the underlying network. In this paper, we will provide first results on the problem of structural outlier detection in massive network streams. Such problems are inherently challenging, because the problem of outlier detection is specially challenging because of the high volume of the underlying network stream. The stream scenario also increases the computational challenges for the approach. We use a structural connectivity model in order to define outliers in graph streams. In order to handle the sparsity problem of massive networks, we dynamically partition the network in order to construct statistically robust models of the connectivity behavior. We design a reservoir sampling method in order to maintain structural summaries of the underlying network. These structural summaries are designed in order to create robust, dynamic and efficient models for outlier detection in graph streams. We present experimental results illustrating the effectiveness and efficiency of our approach.**

I. Introduction

Many communication applications such as social networks, IP networks and internet applications lead to the creation of massive streams of graph data. This has lead to an increasing interest in the problem of mining dynamic graphs [1], [2], [4], [16], [21]. In a graph stream, we assume that individual graph objects are received continuously over time. Some examples of such application scenarios are as follows:

- Many information network objects can be expressed as graph objects [19]. For example, a bibliographic object from the DBLP network may be expressed as a graph with nodes corresponding to authors, conference, or topic area. The graph for the object may be represented in a variety of ways, depending upon application-specific requirements. In general, many objects in information networks are represented as entity-relation graphs which are drawn from a particular kind of schema. For example, the internet-movie database can be represented as an entity-relation graph, in which the structure of the object corresponds to the relationships between the different information elements.
- Social networks [2] typically contain massive patterns of activity among the different users. Each user may be considered a node, and the activity between them may be considered an edge in the graph stream. The pattern of interactions within a small time window, or within users of a particular type may be considered a structural network stream. Similarly, certain events in social networks may lead to local patterns of activity, which may be modeled as streams of graph objects.
- The user browsing pattern at a web site is a stream of graph objects. Each object corresponds to the browsing pattern of a particular user. The edges represent the path taken by the user across the different objects.

The afore-mentioned examples typically correspond to scenarios in which the graphs are defined over a *massive domain of nodes*. The node labels are typically drawn over a universe of distinct identifers, such as the URL addresses in a web graph, an IP-address in a network application, or a user identifier in a social networking application. For example, in the case of a graph with more than 10^7 nodes, the potential number of edges could be of the order of 10^{13}. In such a case, the number of distinct edges in the stream is so large that it becomes impossible to store them for offline processing. The massive graph size also creates a huge challenge for dynamic mining applications of the kind discussed in this paper.

In this paper, we will study the problem of outlier detection in graph streams. Such outliers represent significant deviations from "normal" structural patterns in the underlying graphs. Some example of such deviations are as follows:

- Consider the case where the objects correspond to entity-relation graphs for a movie. The nodes could correspond to actors that are drawn from very diverse genres or regions which do not normally occur together. In such cases, the object may be considered an outlier. This is also an interesting event, because it is likely that the movie has some unusual characteristics which are usually not present in most networks.
- In an academic network, two co-authors which are drawn from groups that usually do not work together may sometimes publish together, especially when one of the authors changes fields or if an interesting cross-disciplinary paper is written. This is an interesting event from the perspective of outlier detection.

In general, *unusual relationships* in the graphs may be represented as *edges between regions of the graph that rarely occur together*. Therefore, the goal of a stream-based outlier detection algorithm is to *identify graph objects which contain such unusual bridging edges*. Such unusual objects need to be identified in the graph stream dynamically, as they are received over time. The dynamic nature of the problem is particularly challenging in the context of graph stream processing, because of the high rate of the incoming stream, and the large number of distinct edges received over time.

In order to address these challenges, we propose a probabilistic algorithm for maintaining summary structural models about the graph stream. These models are maintained with the use of an innovative reservoir sampling approach for *efficient structural compression of the underlying graph stream*. An additional product of the results of this paper is that we show how to dynamically maintain reservoir samples of the graph *with specific structural criteria* satisfying a general condition referred to as *set monotonicity*. While reservoir sampling is typically used in the context of the stream scenario [5], [20] with particular *temporal* statistical properties, there is no known research on the topic of maintaining reservoir samples which *leverage the underlying structural properties* of graph streams. This kind of reservoir sampling is likely to be useful for other applications which leverage particular structural properties during graph sample maintenance.

This paper is organized as follows. The remainder of this section presents related work. In section II, we define the outlier detection problem for the graph scenario. In section III, we discuss the approach of structural reservoir sampling. We show how to use structural reservoir sampling in order to solve the outlier detection problem for the graph stream scenario. Section IV contains the experimental results. Section V presents the conclusions and summary.

A. Related Work and Contributions

The problem of outlier detection has been studied extensively in the context of multi-dimensional data [7], [9], [17], [18]. These techniques are mostly either distance-based [17], [18] or density-based [7], [9] methods. However, these methods cannot be easily generalized to non-spatial networks. The problem of graph outliers presents a special challenge, because the linkage structure in graph objects can be arbitrary, and outliers can only be determined by examining the behavior of the edges in the individual graph objects. Furthermore, the stream scenario presents a special challenge, because the stream objects can be examined at most once during the computation. The related work of clustering has also been studied in the context of graphs and graph streams [1], [2], [4], [16], [19], [21]. A number of different methods for network outlier detection have been discussed in [8], [11], [13]. However, these methods are only applicable to static networks, and not generally applicable to the case of dynamic graph streams. The dynamic nature of graph streams presents a special challenge which cannot be easily addressed in graph mining techniques such as outlier detection, which require intricate structural analysis.

In this paper, we will design the first known *real-time* and *dynamic* method for outlier detection in graph streams. We also design a structural reservoir sampling approach, which designs the structural summarization required for such a dynamic approach. This stream-based structural sampling method is an additional useful by-product of our approach because it satisfies a broad class of desirable properties known as the *set-monotonicity property*. The broad class of properties satisfied by the structural reservoir sample ensures that it is likely to be useful for a variety of other applications beyond the outlier detection problem.

II. Probabilistic Outlier Modeling for Graph Streams

We will first introduce some notations and definitions. We assume that we have an *incoming stream of graph objects*, denoted by $G_1 \ldots G_t \ldots$. Thus, the ith graph in the stream is denoted by G_i. Each graph G_i has a set of nodes, which are drawn from the node set N. Each node is associated with a unique identifier, which may be a string, such as the IP-address in a computer network, or the user-identifier in a social network. The set N may be very large in web-scale applications. For example, in a networking application, the node set may consist of millions of IP-addresses, whereas in a social network, it may consist of millions of users. We formally define the node set N as the *base domain* of nodes.

Definition 1 (Base Node Domain): The base node domain defines the node set N from which the nodes of all the graphs in the stream are drawn.

We note that the base node domain may vary dynamically over time. For example, in a bibliographic information network application, new nodes may be added to the base domain continuously, as new authors (or other entities) are added to the network. Each graph G_i is associated with a set of nodes $N_i \subseteq N$ and a set of edges A_i. In many real applications, we work with the sparsity assumption, in which the node set N_i of each graph G_i is small subset of base node domain N. Next, we formally define the concept of a *graph object* in the data stream:

Definition 2 (Stream Graph Object): Each graph object $G_i = (N_i, A_i)$ is defined as the set of nodes N_i and edges A_i. The set N_i is a subset of the base domain node set N. Each edge in the set A_i is an (undirected) edge between two nodes in N_i.

We use anomalous edge structure in individual graphs (with respect to the overall stream structural patterns) in order to determine outliers. In such cases, outliers may be defined as graph stream objects which have *unusual connectivity structure* among different nodes. Since the unusual connectivity structure can be defined only with respect to the historical connectivity structure, we need some way of representing these structural statistics. The large size of the underlying graph, and the massive number of distinct edges precludes the storage of the entire network stream history explicitly in order to track this structural statistics. Furthermore, the decision on whether an incoming graph object is an outlier needs to be performed in real time. In order to achieve this goal, we will dynamically maintain node partitions which can expose abnormalities in the connectivity structure, and will use it in order to make real-time decisions about the incoming graph objects. Such partitions should ideally represent the dense regions in the graph, so that the rare edges across these dense regions are exposed as outliers.

One challenge is that such a node partitioning needs to be *dynamically maintained for the graph stream*, and a statistical model needs to be concurrently maintained for outlier determination. In order to achieve this goal, we will design a structural reservoir-sampling approach in order to dynamically maintain the partitioning. Such a structural reservoir sampling has the property that it is biased towards creating node groups which are densely connected. This bias towards maintaining the dense regions is useful, because it helps expose abnormal bridge edges across different partitions of the graph. This also ensures that we do not need to use an approach which maintains the optimum partitioning into dense regions of the graph. We will defer the description of structural reservoir sample maintenance (and corresponding node partitioning) to a future section. First, in this section, we will show how to *use the node partitioning information* for probabilistic modeling of outliers.

In order to define the abnormality of edge behavior we define the *likelihood fit of an edge* with respect to a *partition* of the nodes $\mathcal{C} = C_1 \dots C_{k(\mathcal{C})}$. The number of node partitions in $\mathcal{C}$ is denoted by $k(\mathcal{C})$. Each set C_i represents a disjoint subset of the nodes in N, and $\mathcal{C}$ is the notation representing the partitioning induced by the sets $C_1 \dots C_{k(\mathcal{C})}$. It is assumed that the union of the sets $C_1 \dots C_{k(\mathcal{C})}$ is the base node domain N. First, we define the *structural generation model* of edges with respect to node partitioning $\mathcal{C}$. This defines the probability of an edge between a pair of partitions in the incoming graph stream. This structural generation model will be useful for probabilistic modeling of outliers, because a lower probability edge provides evidence of outlier-like characteristics of the underlying graph object.

Definition 3 (Edge Generation Model): The structural generation model of a node partitioning $\mathcal{C} = \{C_1 \dots C_{k(\mathcal{C})}\}$ is defined as a set of $k(\mathcal{C})^2$ probabilities $p_{ij}(\mathcal{C})$, such that $p_{ij}(\mathcal{C})$ is the probability of a randomly chosen edge in the incoming graph object to be incident on partitions i and j.
The edge generation model discussed above includes the case in which $i = j$ and therefore both end points of the edge may lie in the same partition. We also note that since we are working with the assumption of undirected graphs, we have $p_{ij}(\mathcal{C}) = p_{ji}(\mathcal{C})$. For a given node i, we denote the partition identifier for i by $I(i, \mathcal{C})$, which lies between 1 and $k(\mathcal{C})$, depending upon the membership of i in one of the $k(\mathcal{C})$ partitions. The $k(\mathcal{C})$th group of nodes is considered to be special, and accommodates isolated nodes or outlier subgraphs. We will discuss more about this partition in a later section on structural reservoir sampling.

Definition 4 (Edge Likelihood Fit): Consider an edge (i, j), a node partition $\mathcal{C}$, and edge generation probabilities $p_{ij}(\mathcal{C})$ with respect to the partition. Then, the likelihood fit of the edge (i, j) with respect to the partition $\mathcal{C}$ is denoted by $\mathcal{F}(i, j, \mathcal{C})$ and is given by $p_{I(i,\mathcal{C}),I(j,\mathcal{C})}$.
It is important to note that the likelihood fit of an edge *can be defined with respect to any partition in the network*. The purpose of using partitions for the generation model instead of the individual nodes is to allow for a higher level of granularity in the representation, so that enough statistical information can be collected about the edges between the partitions in the underlying stream. While it may not be possible to construct a statistically robust model on the basis of edge presence or absence between individual nodes, it may be possible to create a statistically more robust model for *groups of* nodes which correspond to the partitions. This is because a given stream may not contain an edge between an arbitrary pair of nodes, and therefore a sufficient amount of data is not available to estimate the underlying probabilities between individual nodes. Furthermore, the compression of the statistics into a smaller number of partitions also reduces the space-requirement of maintaining these probabilities tremendously, since the number of such probabilities is governed by the square of the number of partitions.

We maintain *multiple models* in order to increase the robustness of likelihood estimation. This corresponds to multiple ways of partitioning the nodes, and the corresponding statistics also need to be maintained simultaneously. Since each partitioning provides a different way to construct the generation model, it provides a different way of estimate the edge generation probabilities. This smooths out the local variations which are specific to a given partitioning. By combining these different ways of estimation, we can provide a more robust estimate of the underlying probabilities. We denote the r different ways of creating the partitions by $\mathcal{C}_1 \dots \mathcal{C}_r$. Specifically, the ith *partitioning* $\mathcal{C}_i$ contains $k(\mathcal{C}_i)$ different *partitions*. The composite edge-likelihood fit from these r different ways of creating the partitions is defined as the *median* of the edge like-likelihood fits over the different partitions.

Definition 5 (Edge Likelihood Fit (Composite)): The composite edge likelihood fit over the different partitionings $\mathcal{C}_1 \dots \mathcal{C}_r$ for the edge (i, j) is the median of the values of $\mathcal{F}(i, j, \mathcal{C}_1) \dots \mathcal{F}(i, j, \mathcal{C}_r)$. This value is denoted by $\mathcal{MF}(i, j, \mathcal{C}_1 \dots \mathcal{C}_r)$.
The likelihood fit for a graph object G is the product of the likelihood fits of the edges in G. In order to fairly compare between graphs which contain different numbers of edges, we put the fraction $1/|G|$ in the exponent, where $|G|$ is the number of edges in the incoming graph stream object. In other words, we use the geometric mean of the likelihood fits of different edges in the incoming graph stream object. Therefore, we define the object likelihood fit as follows.

Definition 6 (Graph Object Likelihood Fit): The likelihood fit $\mathcal{GF}(G, \mathcal{C}_1 \dots \mathcal{C}_r)$ for a graph object G with respect to the partitions $\mathcal{C}_1 \dots \mathcal{C}_r$ is the geometric mean

```
Algorithm DetectGraphOutliers(Incoming Object: $G$
          Stream Samples: $S_1 \dots S_r$, Partitions: $\mathcal{C}_1 \dots \mathcal{C}_r$
          Probability Statistics: $p^n_{ij}(\cdot)$,$p^s_{ij}(\cdot)$, $p_{ij}(\cdot)$)
begin
  Compute likelihood probabilities of incoming graph
    object $G$ with the use of the stored probability
     statistics in $p_{ij}(\cdot)$;
  if likelihood probability is $t$ standard deviations below
      average of all graphs received report $G$ as outlier;
  Update stream samples $S_1 \dots S_r$ by adding the
    edges in incoming object $G$; (Structural reservoir
     sampling technique of section 3)
  Update partitionings $\mathcal{C}_1 \dots \mathcal{C}_r$ from the
    updated samples $S_1 \dots S_r$ and the corresponding
     probability statistics $p^n_{ij}(\cdot)$,$p^s_{ij}(\cdot)$, and $p_{ij}(\cdot)$;
end
```

Fig. 1. Outlier Detection Framework (Update and Computation Process for Incoming Graph Object)

of the (composite) likelihood fits of the edges in G. Therefore, we have:

$$\mathcal{GF}(G, \mathcal{C}_1 \dots \mathcal{C}_r) = \left[\prod_{(i,j) \in G} \mathcal{MF}(i, j, \mathcal{C}_1 \dots \mathcal{C}_r) \right]^{1/|G|} \quad (1)$$

The process of determination of the outliers uses the maintenance of a group of r different *partitions which are implicitly defined by dynamically maintained reservoirs from the graph stream.* At a given time, the algorithm maintains r different reservoir samples, which are denoted by $S_1 \dots S_r$ respectively. We will discuss the process of maintaining these reservoir samples in the next section. These samples are essentially sets of edges which are picked from the stream graph objects. Each set S_m induces a different partitioning of the nodes in the underlying network, which is denoted by the notation $\mathcal{C}_m$ defined earlier. Specifically, this partition is induced by determining the connected components in the subgraph defined by the edges in S_m. Then, the statistical behavior of the edges in the set corresponding to $Q_m = \cup_{j=1}^r S_j - S_m$ is used in order to model the edge generation probabilities. Thus, we use a *cross-validation approach* in order to model the likelihood statistics in a more robust way, and avoid overfitting of the likelihood statistics to the particular structure induced by a reservoir sample. Since many of the partitions may not contain edges between them, we need a way to smooth out the probability estimation process (rather than simply assigning zero probabilities to such partition pairs). A straightforward way of estimating smoothed edge edge probabilities is as follows:

- We model the *non-sample-estimated* probability of an edge between partitions i and j by assuming that the probability of an edge between any pair of nodes is equal. Therefore, the *non-sample-estimated probability* probability between any pair of partitions is proportional to the product of the number of nodes between these two partitions. This computation is performed for each partition $\mathcal{C}_m$ corresponding to sample S_m. We denote this value by $p^n_{ij}(\mathcal{C}_m)$.
- We model the *sample-estimated* probability of an edge between partitions i and j as the fraction of the number of edges in Q_m which exist between partitions i and j. While this estimation is much more accurate than the previous case above, the problem with it is that it is rather inaccurate for partition pairs (i, j) which contain a small number (or none) of the edges from Q_i. Therefore, the uniform assumption above is helpful for smoothing, when the data available for estimation is small. We denote this value by $p^s_{ij}(\mathcal{C}_m)$.
- The *estimated* probability $p_{ij}(\mathcal{C}_m)$ of an edge between partitions i and j is a weighted combination of the above two factors. Therefore, for some small value of $\lambda \in (0, 1)$ we have:

$$p_{ij}(\mathcal{C}_m) = \lambda \cdot p^n_{ij}(\mathcal{C}_m) + (1 - \lambda) \cdot p^s_{ij}(\mathcal{C}_m) \quad (2)$$

For each incoming graph stream object, we compute the likelihood fit with the use of the above-mentioned estimated probabilities. Those objects for which this fit is t standard deviations below the average of the likelihood probabilities of all objects received so far are reported as outliers. Then, we will use the reservoir sampling approach discussed below in order to update the edges in the different samples $S_1 \dots S_r$. These are also used in order to update the partitions $\mathcal{C}_1 \dots \mathcal{C}_r$. At the same time the values of $p^n_{ij}(\cdot)$, $p^s_{ij}(\cdot)$, and $p_{ij}(\cdot)$ are dynamically updated. More details on the methodology for defining effective partitions, maintaining them with a reservoir sampling approach, and updating underlying probability statistics for model estimation are discussed in the next section. The overall framework for outlier detection is illustrated in Figure 1.

In order to determine whether the likelihood probability of an object is t standard deviations below the likelihood probability of all objects received so far, we need to dynamically maintain the mean and standard deviation of the likelihood probabilities of all objects maintained so far, in a way which can be *additively achieved* for a data stream. For this purpose, we maintain three values: (1) The sum of squares of likelihood probabilities (second moment), (2) the sum of the likelihood probabilities (first moment), and (3) the number of graph objects received so far (zeroth moment).
It is easy to see that all of the above values can be maintained efficiently and additively for a data stream. The mean and standard deviation can be directly computed from the above values, because both of the these statistical quantities can be expressed as a closed function of moments of order at most 2 [6].

III. Structural Reservoir Sampling: Methodology and Applications

In this section, we will study the methodology of structural reservoir sampling, and its applicability to the problem of model estimation for outlier detection. The aim of the partitioning process is to construct clusters of dense nodes, which would expose the outliers well. Since clustering is a very challenging problem for the stream scenario, we will create node partitions from samples of edges; it is well known that the use of edge sampling [3] to create such partitions are biased towards creating partitions which are dense. In order to further increase the likelihood of outlier exposure, we will use multiple choices of edge samples, in order to create different kinds of node partitions. This is used in order to improve the robustness of abnormality estimation. The sampling methods discussed in [3], [14] are not applicable to the case of the stream scenario and are also not designed for *maintaining specific structural properties* in the underlying partitions. In our particular case, the structural constraint is that we would like each partition to have a certain minimum number of points, or to constrain the total number of partitions (in order to ensure robust statistics maintenance). Therefore, we will design a structural reservoir sampling method for graph streams. Furthermore, we anticipate that such a sampling method is likely to have structural partitioning applications beyond those discussed in this paper. Reservoir sampling [20] is a methodology to dynamically maintain an unbiased sample from a stream of elements. In this case, we extend the approach to an unbiased sample of a structured graph with a *structural stopping criterion*. We will impose an elegant solution in which many natural and desirable structural properties of the sample are maintained with the help of a *monotonic set function* of the underlying edges in the reservoir. We define a monotonic set function of an edge sample as follows:

Definition 7 (Monotonic Set Function): A monotonically non-decreasing (non-increasing) set function is a function $f(\cdot)$ whose argument is a set, and value is a real number which always satisfies the following property:

- If S_1 is a superset (subset) of S_2, then $f(S_1) \geq f(S_2)$

We note that the monotonic set function can be useful for regulating the structural characteristics of the graph over a given set of edges. Some examples of a monotonic set function with corresponding structural properties are as follows:

- The function value is the number of connected components in the edge set S (monotonically non-increasing).
- The function value is the number of nodes in the the largest connected component in edge set S (monotonically non-decreasing).

Properties such as the above are very useful for inducing the appropriate partitions with robust structural behavior. In some cases, we can use *thresholds* on the above properties, which are also referred to as *stochastic stopping criteria*. Next, we define the concept of a sort sample with a such a stopping criterion. We will first define this sample on a static data set $\mathcal{D}$, and then show how to generalize it to a data stream with a reservoir sampling approach. We examine a restricted bunch of subsets of $\mathcal{D}$, in which the edges are sorted, and can be added to S only in **sort order priority**; in other words, an edge cannot be included in a subset S, if all the elements which occur before it in the sort order are also included. Clearly, the number of such subsets of $\mathcal{D}$ is linear in the size of $\mathcal{D}$.

Definition 8 (Sort Sample with Stopping Criterion): Let $\mathcal{D}$ be a set of edges. Let $f(\cdot)$ be a monotonically non-decreasing (non-increasing) set function defined on the edges. A sort sample S from $\mathcal{D}$ with stopping threshold α is defined as follows:

- We sort all edges in $\mathcal{D}$ in random order.
- We pick the *smallest* subset S from $\mathcal{D}$ **among all subsets which satisfy the sort-order priority**, such that $f(S)$ is at least (at most) α.

This means that if we remove the last element which was added to the set, then that set (and all previous subsets) will not satisfy the stopping criterion. As a practical matter, the set which is obtained by removing the last element which was added is the most useful for processing purposes. For example, if $f(S)$ is the size of the largest connected component, the stopping criterion determines the smallest sample S, such that the size of the largest connected component is at least a user-defined threshold α. By dropping the last edge (v, w) which was added to S, we are guaranteed that the size of the largest connected component in $S - \{(v, w)\}$ is less than α. This ensures that none of the connected components is too large to use for processing. Correspondingly, we define a *penultimate set* for a sort sample.

Definition 9 (Penultimate Set): The penultimate set for a sort sample S is obtained by removing the last element in the sort order of sample S.

For the case of a *fixed data set*, it is fairly easy to create a sort sample with a structural stopping criterion. We achieve this by sorting the edges in random order and adding them sequentially, until the stopping criterion can no longer be satisfied. However, in the case of a data stream, a random sample or reservoir needs to be maintained *dynamically*. Once edges have been dropped from the sample, how does one compare their sort order to the incoming edges in the stream, and correspondingly update the sample?

The key idea is use a *fixed random hash function*, which is computed as a function of the node labels on the edge, and remains fixed over the entire stream. This hash function is used to create a sort order among the different edges. This hash function serves to provide *landmarks* for incoming edges when they are compared to the previously received edges from the data stream. Furthermore, the use of a hash function *fixes the sort order among the edges throughout the stream computation*. The fixing of the sort order is critical in being able to design an effective structural sampling algorithm. Therefore, for an edge (i, j) we compute the hash function $h(i \oplus j)$, where $i \oplus j$ is the concatenation of the node labels i and j. We note that the use of a sort on the hash function value induces a random sort order on the stream elements. Furthermore, a stopping criterion on the sorted data set *translates* to a threshold on the hash function value. This provides an effective way to control the sampling process. We make the following observation:

Observation 1: Let $\mathcal{D}$ be a set of edges. let $f(\cdot)$ be a monotonically non-decreasing (non-increasing) set function defined on the edges. A sort sample S from $\mathcal{D}$ with stopping threshold α is equivalent to the following problem:

- Apply a uniform random hash function $h(\cdot)$ to each edge (i, j) in $\mathcal{D}$.
- Determine the smallest threshold q, such that the set S of edges which have hash function value at most q satisfy the condition that $f(S)$ is at least (at most) α.

We denote the corresponding threshold value with respect to set function f, hash function h. data set $\mathcal{D}$ and stopping threshold criterion α by $H(f, h, \mathcal{D}, \alpha)$, and refer to it as the *stopping hash threshold* for data set $\mathcal{D}$. We make the following observation about the stopping hash threshold:

Lemma 1: The stopping hash threshold exhibits a version of set monotonicity with respect to the underlying data set. Specifically, let us consider the monotonic set function $f(\cdot)$, hash function $h(\cdot)$, stopping threshold α. Let us consider two data sets $\mathcal{D}_1$ and $\mathcal{D}_2$, such that $\mathcal{D}_2 \supseteq \mathcal{D}_1$. Then, the stopping hash threshold $H(f, h, \mathcal{D}_2, \alpha)$ is at most equal to $H(f, h, \mathcal{D}_1, \alpha)$. In other words:

$$H(f, h, \mathcal{D}_2, \alpha) \leq H(f, h, \mathcal{D}_1, \alpha) \tag{3}$$

Proof: The set monotonicity of the hash threshold follows directly from the set monotonicity of the function $f(\cdot)$. Since $\mathcal{D}_2$ is a superset of $\mathcal{D}_1$, it follows that the sets of edges W_2 and W_1 derived by using the same hash function and hash threshold on both data sets $\mathcal{D}_2$ and $\mathcal{D}_1$ would result in the former set W_2 being a superset of the latter set W_1. As a result, the set W_2 may not be the minimal set to satisfy the stopping criterion, if the hash threshold is chosen in order to make W_1 satisfy the stopping criterion. Thus, the hash threshold for the former set may (or may not) need to be further reduced in order to make it satisfy the stopping criterion, though it never needs to be increased. The result follows. ■

We note a stream can be viewed as a continuously increasing set of edges. Therefore, the result above can be directly used to conclude that the stopping hash threshold is monotonically non-increasing over the progress of the data stream.

Corollary 1: The stopping hash threshold is monotonically non-increasing over the life of the data stream.

This is a critical result, because it implies that edges which have not been included in the current sample will never be relevant for sampling over the future life of the data stream. Therefore, the current sample is the only set we need for any future decisions about reservoir sample maintenance. Furthermore, the result also implies a simple algorithm in order to maintain the reservoir dynamically. We dynamically maintain the *current hash threshold* which is used to make decisions on whether or not incoming elements are to be included in the reservoir. For each incoming graph, we apply the hash function to each of its edges, and we add the edge to the reservoir, if the hash function value is less than the current threshold value. We note that the addition of these edges will always result in the stopping criterion being met because of set function monotonicity. However, the set may no longer be the *smallest sort-ordered set* to do so. Therefore, edges may need to be removed in order to make it the smallest sort-ordered set to satisfy the stopping criterion. In order to achieve this goal, we process the edges in the reservoir *in decreasing order of the hash function value*, and continue to remove edges, until we are satisfied that the resulting reservoir is the smallest possible set which satisfies the stopping constraint. For each incoming graph, this process is repeated by first selectively adding the edges for which the hash function meets the threshold, and then removing edges if necessary. The corresponding hash threshold is then reduced to the largest hash function value of the *remaining edges in the reservoir* after removal. Clearly, the removal of edges may result in a reduction of the hash threshold in each iteration. However, it will never result in an increase in the threshold, because all the added edges had a hash function value lower than the threshold in the previous iteration. It is important to note that *we always use the penultimate set derived from the sort sample* for the purposes of partitioning.

The above description explains the maintenance of a single reservoir (and corresponding partition). For robustness, we need to maintain r different reservoirs. Therefore, we need to use r different hash functions, and corresponding reservoir samples. These are used to define the r different partitionings of the nodes as required by the outlier modeling algorithm. We denote the *penultimate sets* of the r different reservoirs as $S_1 \ldots S_r$. These will be used for the purpose of inducing the r different partitionings denoted by $\mathcal{C}_1 \ldots \mathcal{C}_r$.

A. The Edge Sampling Approach to Partitioning: Discussion

The approach discussed above designs the partitions with the use of edge sampling. A key question is as to why this *edge sampling* approach should be useful in exposing outliers in the underlying data. For example, if statistical robustness is the only reason for creating multi-node partitions, then why not design the node partitions randomly by assigning nodes randomly to partitions? However, such an approach does not expose the key edges which bridge significant partitions in the underlying data, because we are specifically looking for partitions which are statistically biased towards having a small number of bridge edges among them. *An edge sampling approach is naturally biased towards creating partitions in which individual components are statistically much denser than the rest of the data.* In fact, it has been shown in [14], how edge sampling approaches have much higher likelihood of containing cuts with lower value. A natural solution would be to use a graph-partitioning technique [15], but this cannot be implemented efficiently for the case of graph stream. We will provide an intuitive qualitative argument as to why such an edge sampling technique should work well, in addition to its natural efficiency for a data stream.

The quality of outliers are clearly sensitive to the nature of the partitions, because we would like the partitions to represent densely connected nodes with a small number of bridge edges. Such partitions expose regions of the graph between which there are very few edges. Therefore, when an incoming graph object contains edges which are bridges across such partitions, the low likelihood of this occurrence is reflected in the quantification of the corresponding likelihood fit. The use of *edge sampling* probabilistically biases the partitions to correspond to dense local regions of the graph, as discussed in [14]. This is because connected regions in an edge sample will typically be biased towards dense regions in the graph. In fact, it has been shown in [14], that the use of repeated edge samples will always retain the minimum cut across the partitions with high probability. Such cuts are invaluable in exposing the key bridge edges across the partitions in the incoming stream objects. The use of multiple reservoir samples provides robustness to the likelihood fit estimation process.

B. Choosing the Structural Function

Another key question which affects the quality of the partitions is the choice of the function $f(S)$, which is used to control the partitions. Clearly, we would like to have a compact number of partitions, so that the summary statistics about the likelihood fits can be maintained compactly. One possible approach is that we set $f(S)$ to be the number of connected components in the corresponding graph induced by edge sample S. However, this approach is not likely to result in discriminative creation of partitions, as most networks have variations in structural density, and a sampling approach will result in a vast majority of components to very small, and a single component to be very large. Components which are either too small or too large are not very helpful in exposing anomalous structural behavior in a robust way. Therefore, it is useful to set the function in such a way so as to control the sizes of the underlying partitions from the reservoir. Therefore, an effective alternative is to set $f(S)$ to the number of nodes in the largest connected component induced by the reservoir edge sample S. A corresponding stopping threshold of α is used on the function $f(S)$. Thus, the stopping criterion ensures that the sort sample defines the *smallest* set of edges, such that the largest connected component is *at least* α. This ensures that the largest partition has at most α nodes, *when the penultimate set derived from* S *is used*.

C. Additional Implementation Details

While the use of a maximum partition size, improves the selectivity of the larger connected components, there may still be many partitions which are too small to provide robust information about the likelihood fits of edges emanating from such partitions. For example, newly encountered nodes in the data stream are likely to be isolated nodes in partitions of their own. Therefore, all partitions which contain less than min_thresh nodes are grouped into a single outlier partition, which is treated as a unit during processing. For each of the r different reservoirs, we also need to maintain the estimated information about the probabilities $p_{ij}^n(\mathcal{C}_m)$ and $p_{ij}^s(\mathcal{C}_m)$. The values of $p_{ij}^n(\mathcal{C}_m)$ need not be maintained explicitly because it can be derived directly from the number of nodes in the partitions. The values of $p_{ij}^s(\mathcal{C}_m)$ need to be maintained explicitly. This is done by using the samples $S_1 \ldots S_r$, which are defined as the *penultimate sets* of the corresponding sort samples for the different reservoirs. For each partition $\mathcal{C}_m$ induced by sample S_m, we *dynamically maintain* the statistics of the edges in $\cup_{i=1}^r S_i - S_m$. The values of $p_{ij}^s(\mathcal{C}_m)$ are maintained indirectly as the *total number of edges among all graph objects received so far* which are incident between partitions i and j (for partitioning $\mathcal{C}_m$) using all the edges in $\cup_{i=1}^r S_i - S_m$. The corresponding probability can be estimated by dividing this value by the total number of edges in all graphs received so far in the stream. We note that the process of dynamic maintenance may require the merging or division of the statistics across different partitions, when new edges are added to or deleted from the reservoirs. Furthermore, some connected components may move into or out of the outlier partition set, when new edges are added or deleted. These operations may affect the value of $k(\mathcal{C}_m)$ over the course of the data stream. For example, when two connected components are merged, the corresponding rows and columns in the statistics need to be merged as well. The reverse argument applies to a split in the underlying components. Correspondingly, the size of the maintained statistics $k(\mathcal{C}_m) \times k(\mathcal{C}_m)$ may vary over the course of algorithm execution. However, for large enough values of α, the maintained statistics is compact enough to be maintained dynamically. Furthermore, when the same edge occurs multiple times across a partition, it is counted an equal number of times in the corresponding statistics.

In order to track the statistics above, it is necessary to maintain the information about the connected components *dynamically* during stream processing. The connected components are tracked by using the *spanning forests* of each of the edge samples dynamically. Thus, we need to maintain r different sets of spanning forests corresponding to the r different partitioning structures induced by the different reservoirs. Efficient algorithms for dynamically maintaining spanning forests of incrementally updated sets of edges are discussed in [10], [12].

IV. Experimental Results

In this section, we tested our outlier detection approach for effectiveness and efficiency on a number of real and synthetic data sets. We refer to our approach as the the *GOutlier* method, which corresponds to the fact that it is a Graph Stream Outlier Detection Method. Since there is no known method for outlier detection in graph streams, we used the graph clustering framework $GMicro$ in [4] as a baseline. An outlier was defined as a point which does not naturally belong to any of the clusters. If an incoming graph lies outside the structural spread (defined in [4]) of any of the current clusters, then we assume that this graph is an outlier.

A. Data Sets

We used a combination of real and synthetic data sets in order to test our approach. The data sets used were as follows:

(1) DBLP Data Set: The DBLP data set contains scientific publications in the computer science domain. We further processed the data set in order to compose author-pair streams from it. All conference papers ranging from 1956 to March 15th, 2008 were used for this purpose. There are $595,406$ authors and $602,684$ papers in total. We note that the authors are listed in a particular order for each paper. Let us denote the author-order by $a_1, a_2, \ldots, a_q$. An author pair $\langle a_i, a_j \rangle$ is generated if $i < j$, where $1 \leq i, j \leq q$. There are $1,954,776$ author pairs in total. Each conference paper along with its edges was considered a graph.

(2) Internet Movie Database Data Set: The Internet Movie Database(**IMDB**) is an online collection of movie information. We obtained five-year movie data from 2001 to 2005 from IMDB in order to generate the graph stream. The stream generation methodology is similar to the one used for the DBLP data set. Each movie was considered a graph, and actor and director pairs within a movie were considered edges. The IMDB data set contained a total of $117,856$ movies and $16,191,159$ actor-director pairs.

(3) Synthetic Data Set: We used the R-Mat data generator in order to generate a base template for the edges from which all the graphs are drawn. The input parameters for the R-Mat data generator were $a = 0.5$, $b = 0.2$, $c = 0.2$, $S = 17$, and $E = 508960$ (using the CMU NetMine notations). If an edge is not present between two nodes, then the edge will also not be present in *any graph* in the data set. Next, we generate the base partitions. Suppose that we want to generate κ base partitions. We generate κ different zipf distributions with distribution function $1/i^\theta$. These zipf distributions will be used to define the probabilities for the different nodes. The base probability for an edge (which is present on the base graph) is equal to the product of the probabilities of the corresponding nodes.

Next, we determine the number of edges in each graph. The number of edges in each of the generated graph is derived from a normal distribution with mean $\mu = 10$ and standard deviation $\sigma = 2$. The proportional number of points in each partition is generated using a uniform distribution in $[\alpha, \beta]$. We used $\alpha = 1$, and $\beta = 2$. In order to generate a regular graph, we first determine which partition it belongs to by using a biased die, and then use the probability distributions to generate the edges. In order to generate an outlier graph, we first randomly pick two different partitions, and then select one node from each and create an edge between these two nodes as a bridge between those two partitions. In order to add correlations, we systematically add the probabilities for some of the other distributions to the ith distribution. In other words, we pick r other distributions and add them to the ith distribution after adding a randomly picked scale factor. We define the distribution $\mathcal{S}_i$ from the original distribution $\mathcal{Z}_i$ as follows:

$$S_i = Z_i + \alpha_1 \cdot \text{(randomly picked } Z_j) + \ldots$$
$$\ldots + \alpha_r \cdot \text{(randomly picked } Z_q)$$

$\alpha_1 ... \alpha_r$ are small values generated from a uniform distribution in $[0, 0.1]$. The value of r is picked to be 2 or 3 with equal probability. We use $S_1 \ldots S_r$ to define the node probabilities. We used a clustering input parameter of $\kappa = 10$.

B. Evaluation Metrics

We used a variety of metrics for the purpose of evaluation. For the case of the synthetic data set (in which the "ground truth" corresponding to the true outliers were known), we used the **false positive rate** and the **false negative rate**. These metrics are defined as follows:

- **False Positive Rate**: This is defined as the fraction of normal graphs classified as outlier graphs.
- **False Negative Rate**: This is defined as the fraction of outlier graphs classified as normal graphs.

It is clear that both metrics are fractions, and would lie in the range $(0, 1)$. The lower the value of each metric, the better the quality of outlier detection. We further note that it is possible to achieve different tradeoffs between the false positives and the false negatives with the use of different thresholds on the variable used for determining the outlier quality. For example, in the case of the *GOutlier* method, this variable is the likelihood probability, whereas in the case of the *GMicro* method, this variable is the structural spread. By using different thresholds on these variables, it is possible to achieve different tradeoffs between false positives and false negatives.

The case of real data is much more challenging, because such metrics cannot be defined when the "ground truth" is not available. Therefore, we will conduct some case studies for real data sets by showing some interesting outliers which are generated by the *GMicro* method, especially those which are missed by the baseline approach. For efficiency, we tested the processing rate with the stream progressing for both synthetic data set and real data sets.

Unless otherwise mentioned, the default value of number of samples was 10, and the stopping threshold of the number of nodes in the largest connected component was set to 70. In the baseline approach, the length of sketch table was set to 1000, and the number of hash functions was set to 15.

C. Case Studies for Real Data Sets

We will first present some examples of results obtained by our algorithm on DBLP and IMDB data sets. We will only pick the outliers discovered by our proposed algorithm but ignored by the clustering algorithm as examples. This provides intuition and insight about the effectiveness of the scheme, and why it is able to determine interesting outliers.

Example 1: [DBLP] Yihong Gong, Guido Proietti, Christos Faloutsos, *Image Indexing and Retrieval Based on Human Perceptual Color Clustering*, CVPR 1998: 578-585.

As mentioned earlier, each graph in the DBLP data set represents a paper, and the nodes represent the authors in that paper. One of the papers discovered as an outlier is shown above. This was because of the presence of authors who naturally belonged to different partitions. Specifically, the first author Yihong Gong and the third author Christos Faloutsos belong to different partitions in our experiment. This characteristic was observed over many different instantiations of the partitioning. The common nodes which co-occur with each of these authors were as follows:

- Nodes in **Partition 1A**: Yihong Gong, Chau Hock Chauan, Masao Sakauchi, etc.
- Nodes in **Partition 2A**: Christos Faloutsos, Rakesh Agrawal, Jiawei Han, Philip S. Yu, etc.

We notice that Partition 1A is a group of researchers in computer vision and multimedia processing. On the other hand, Partition 2A is a relatively large and growing community which represents researchers in the database and data mining area. In the above example, the first author and the third author come from different communities, but they co-authored an interdisciplinary work in database and computer vision corresponding to an image indexing and retrieval paper. This naturally represents an interesting anomaly, since such papers are unusual in terms of the interactions of researchers from diverse communities. The *GOutlier* Algorithm was able to determine such outliers, because it observed that the co-authors of this paper naturally belonged to many different partitions in different instantiations of reservoir sampling. As a result, a low likelihood probability was assigned to the corresponding edges. Therefore, the algorithm (appropriately) discovered this paper as an outlier.

Outlier 2: [DBLP] Natasha Alechina, Mehdi Dastani, Brian Logan, John-Jules Ch Meyer, *A Logic of Agent Programs*, AAAI 2007: 795-800.

We notice that all authors in this example are interested in artificial intelligence, and especially agent systems. Then, why is this paper classified as an outlier, because it would seem that the authors are drawn from related groups? By analyzing the partitioning information from the algorithm, we further noticed that the four authors often fall into two partitions (over different reservoir samples), and these samples frequently contain the following other nodes:

- Nodes in **Partition 2A**: Natasha Alechina, Mark Jago, Michael Lees, Brian Logan, etc.
- Nodes in **Partition 2B**: John-Jules Ch Meyer, Mehdi Dastani, Frank Dignum, Rogier M. van Eijk, etc.

Through further understanding of the partitions, we realized that the co-authorship behavior of these cohorts was defined by geographical proximity. The first partition includes a group of researchers in the United Kingdom, while the second partition is composed of researchers in the Netherlands. Before this paper was published, these two communities were growing independently (likely because of geographical separation), though they were working in the same area. This paper is an unusual bridge between the two communities, and is therefore an outlier. Such outliers can be very useful, because they provide us with information about important and interesting events in the underlying data.

Outlier 3: [IMDB] *Movie Title:* Cradle 2 the Grave (2003)

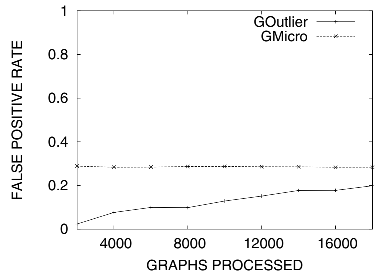
This movie was directed by Andrzej Bartkowiak, and the actors include Jet Li, DMX (I), etc. Jet Li is a Chinese actor who is known for his kung-fu films. As shown by the outlier detection algorithm, he belongs to a partition which also contain many Chinese actors and directors due to his frequent appearances in many Chinese movies and shows. DMX (I), on the other hand, appears more frequently in various American TV shows and music rewards, since he is an American rapper and actor. This movie was starred by two actors from totally different backgrounds, and their profile also did not fit the other set of actors often picked by Andrzej Bartkowiak. Thus, this movie brings an unusual cast of directors and actors together and is reasonably discovered as an outlier.

Outlier 4: [IMDB] *Movie Title:* Memoirs of a Geisha (2005)

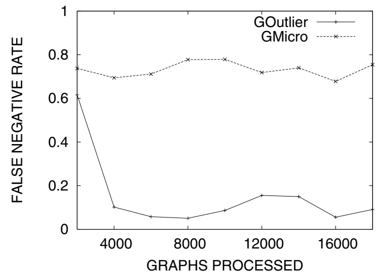
This movie was directed by Rob Marshall, who is an American film director. The cast of this movie is international and includes Youki Kudoh, Gong Li, Suzuka Ohgo, Michelle Yeoh, and Ziyi Zhang. This movie was quite different from previous movies directed by Rob Marshall, which were of a more American nature (such as, for example, Chicago[2002]). Furthermore, the actors of this movie come from different backgrounds. Actors such as Youki Kudoh and Suzuka Ohgo are from Japan, and they have little appearances in American movies. Some other actors such as Ziyi Zhang, Gong Li and Michelle Yeoh are Chinese actors. Because of the unique combination of backgrounds of the directors and actors, this unusual movie is captured by the outlier detection scheme.

D. Effectiveness Results

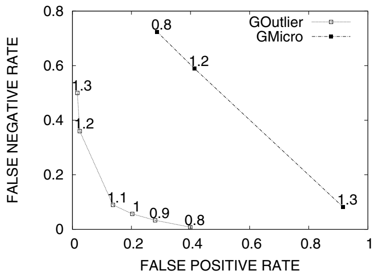
While the real data sets provide interesting insights in terms of the kinds of outliers the *GOutlier* algorithm can discover, we also provide validation with synthetic data sets in which the ground truth about the underlying outliers is also available. As mentioned earlier, we use the false-positive and false-negative rates in order to declare data points as outliers. For the case of the *GOutlier* method, the value of t was set at 1.0, whereas for the case of the *GMicro* method, the threshold on the spread was also set at 1.0. The results for the false positive rate with stream progression are illustrated in Figure 2(a). The number of graphs processed is illustrated on the X-axis, whereas the false positive rate is illustrated on the Y-axis. It is clear that the proposed *GOutlier* scheme outperforms the baseline by a wide margin in terms of the false positive rate. In fact, the error of the *GOutlier* scheme was approximately half the error of the *GMicro* scheme. In Figure 2(b) we have illustrated the false negative rate for both approaches for the same setting of parameters. As in the previous case, the progression of the stream is illustrated on the X-axis, and the false negative rate is illustrated on the Y-axis. One interesting trend for the proposed *GOutlier* approach is that the false negative rate is somewhat high in the initial phases of the scheme (though still lower than *GMicro*), but it drops rapidly in steady state. The reason is that the proposed scheme has not obtained enough data in the beginning, and therefore it is more likely to miss some real outliers. As more graphs are added, the constructed



(a) False Postives

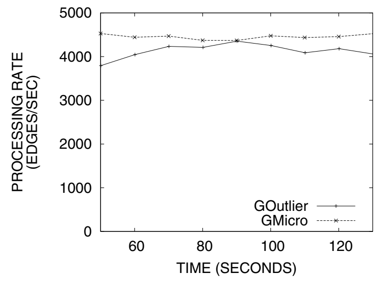


(b) False Negatives

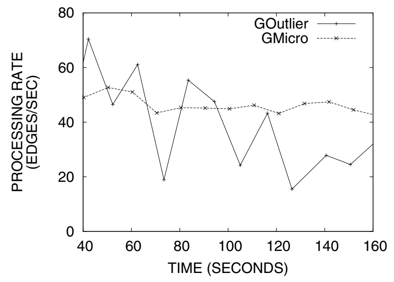


(c) False Pos/Neg Tradeoff

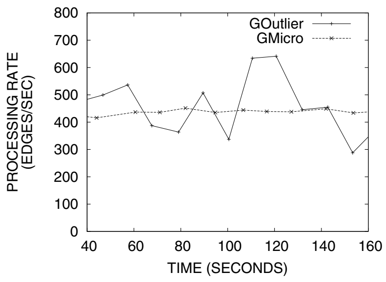
Fig. 2. Effectiveness on Synthetic Data Set



(a) Synthetic Data Set

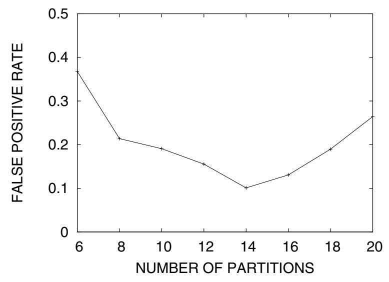


(b) DBLP Data Set

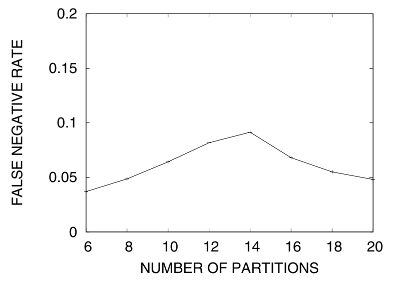


(c) IMDB Data Set

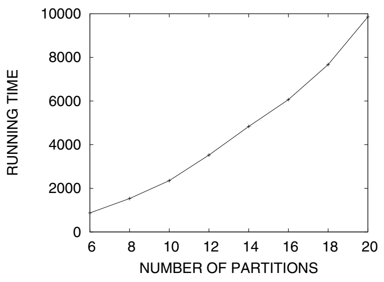
Fig. 3. Efficiency Results



(a) False Positives



(b) False Negatives



(c) Running Time

Fig. 4. Sensitivity with Increasing Number of Partitions (Synthetic Data)

partitions represent the underlying data more accurately. As a result, the false negative rate drops and is maintained at a low rate. In comparison to the baseline approach, we also find that the false negative rate of our proposed approach is significantly lower.

We further examined the false positive rate versus the false negative rate by varying the threshold t over which a graph is reported as an outlier. In the outlier detection algorithm, if the likelihood probability of the incoming graph is t standard deviations below average of all graphs received, this graph will be reported as an outlier. Therefore, the false positive rate will increase but the false negative rate will decrease if we decrease the value of threshold t, and vice versa. In this case, we computed this metric over a fixed set of 10,000 graphs. The tradeoff between the false positive rate and the false negative rate is presented in Figure 2(c). The false positive rate is presented on the X-axis, whereas the false negative rate is presented on the Y-axis. The value of threshold t of *GOutlier* and spread threshold of *GMicro* are also shown next to each data point within the figure. It once again clearly shows that *Goutlier* greatly outperforms *GMicro* on outlier detection in terms of effectiveness. Based on our experiments, we found that by using a value of t in the range $[1.0, 1.1]$, we obtain a good tradeoff between false negatives and false positives. This can of course be varied depending upon different application-specific scenarios.

E. Efficiency Results

We also tested the efficiency of the proposed approach and the baseline approach for both synthetic data set and real data sets. The results for the synthetic, DBLP and IMDB data sets are illustrated in Figures 3(a), (b) and (c) respectively. In each figure, the X-axis represents the progression of the stream, whereas the Y-axis illustrates the stream processing rate (in terms of the number of edges processed per second). We noticed that the processing rate of our proposed approach fluctuates somewhat with progression of the stream. This is due to the maintenance of the edge sampling structure. When more graphs are added, some edges have to be removed in order to keep the number of nodes in the largest connected component within the stopping threshold. The number of such edges to be removed may vary with time, and in some cases, no edges may need to be removed at all. As a result, the processing rate can vary somewhat over time. It is evident that the processing rate of our scheme is comparable with that of the *GMicro* scheme. This is quite acceptable, considering the tremendous qualitative advantages of our approach.

F. Sensitivity Analysis

It is also valuable to test the effectiveness and efficiency of the proposed approach over different number of partitions. The Figures 4(a), (b), and (c) show the results for the false positive rates, false negative rates and running time respectively by varying the number of partitions from 6 to 20. In all figures, the X-axis illustrates the number of partitions, and the Y-axis represents the different metrics. From Figure 4(a), it is evident that setting the number of partitions either too small or too large will increase the false positive rate. When the number of partition is too few, it will not be a good representation of underlying data, hence will not result in good discrimination. Therefore, small random variations are sometimes reported as positives. On the other hand, too many partitions will result in difficulty in statistics estimation in any robust way, which will also increase the false positive rate. In general, the number of positives reported for too few or too many partitions is larger, and therefore the number of false positives is also larger. An inverse result to the false positive rate was observed for the case of false negatives. In Figure 4(c). This is an artifact of the natural tradeoff between false positives and false negatives, because when the number false positives are larger, the number of false negatives are generally fewer and vice-versa. Based on the running time shown in Figure 4(c), the running time of *GOutlier* scales almost linearly with increasing number of partitions. This suggests that the *GOutlier* method is an extremely efficient and scalable algorithm over a wide range of parameter settings.

V. Conclusions and Summary

Streaming applications have become more common in the graph domain because of numerous social networking applications which naturally generate such data. Such data may often have anomalies in the form of unusual connections between entities that are rarely linked together. In this paper, we presented a method for outlier detection in graph streams with the use of a structural reservoir sampling approach. The reservoir sampling approach is designed to capture a summary representation of the graph, while guaranteeing certain structural properties in this summary. The method proposed for reservoir sampling is interesting in its own right as a general method for sampling graphs, and may be useful for a number of different graph applications which require summarization. We present case studies of the outlier detection method on the DBLP and IMDB data sets, and provide some interesting examples of outliers. We also test the method on synthetic data sets in which we show that our method is more effective than competing methods. Furthermore, it is also extremely efficient in terms of the speed of stream processing. Thus, this paper presents an effective and efficient method for outlier detection in graph streams.

Acknowledgement

This work is supported in part by NSF through grants IIS-0905215, OISE-0968341, and OIA-0963278.

References

[1] C. Aggarwal and H. Wang, *Managing and Mining Graph Data*, Springer, 2010.

[2] C. Aggarwal, *Social Network Data Analytics*, Springer, 2011.

[3] C. Aggarwal, Y. Xie, and P. Yu, "Gconnect: A connectivity index for massive disk-resident graphs," in *PVLDB*, vol. 2(1), pp. 862–873, 2009.

[4] C. Aggarwal, Y. Zhao, and P. Yu, "On clustering graph streams," in *SIAM Conf. on Data Mining*, pp. 478–489, 2010.

[5] C. Aggarwal, "On biased reservoir sampling in the presence of stream evolution," in *VLDB Conf.*, pp. 607–618, 2006.

[6] C. Aggarwal, J. Han, J. Wang, and P. Yu, "A framework for clustering evolving data streams," in *VLDB Conf.*, pp. 81–92, 2003.

[7] C. Aggarwal and P. Yu, "Outlier detection for high dimensional data," in *SIGMOD Conf.*, pp. 37–46, 2001.

[8] L. Akoglu, M. McGlohon, and C. Faloutsos, "Oddball: Spotting anomalies in weighted graphs," in *PAKDD Conf.*, pp. 420–421, 2010.

[9] M. Breunig, H.-P. Kriegel, R. Ng, and J. Sander, "Lof: Identifying density-based local outliers," in *SIGMOD Conf.*, pp. 93–104, 2000.

[10] G. N. Frederickson, "Data structures for on-line updating of minimum spanning trees, with applications," in *SIAM J. Comput.*, vol. 14(4), pp. 781–798, 1985.

[11] J. Gao, F. Liang, W. Fan, C. Wang, Y. Sun, and J. Han, "On community outliers and their efficient detection in information networks," in *ACM KDD Conf.*, pp. 813–822, 2010.

[12] M. Henzinger and V. King, "Randomized fully dynamic graph algorithms with polylogarithmic time per operation," in *J. ACM*, vol. 46(4), pp. 502–516, 1999.

[13] W. Jin, Y. Jiang, W. Qian, and A. K. H. Tung, "Mining outliers in spatial networks," in *DASFAA*, pp. 156–170, 2006.

[14] D. R. Karger, "Random sampling in cut, flow, and network design problems," in *STOC*, pp. 648–657, 1994.

[15] B. W. Kernighan and S. Lin, "An efficient heuristic for partitioning graphs," in *Bell Systems Technical Journal*, vol. 49, pp. 291–307, 1970.

[16] M.-S. Kim and J. Han, "A particle-and-density based evolutionary clustering method for dynamic networks," in *PVLDB*, vol. 2(1), pp. 622–633, 2009.

[17] E. Knorr, R. Ng, and V. Tucakov, "Distance-based outliers: Algorithms and applications," in *VLDB J.*, vol. 8(3-4), pp. 237–253, 2000.

[18] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *SIGMOD Conf.*, pp. 427–438, 2000.

[19] Y. Sun, Y. Yu, and J. Han, "Ranking-based clustering of heterogeneous information networks with star network schema," in *ACM KDD Conf.*, pp. 797–806, 2009.

[20] J. S. Vitter, "Random sampling with a reservoir," in *ACM Trans. Math. Softw.*, vol. 11(1), pp. 37–57, 1985.

[21] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," in *PVLDB*, vol. 2(1), pp. 718–729, 2009.