2013 IEEE 13th International Conference on Data Mining

On Anomalous Hotspot Discovery in Graph Streams

Weiren Yu
SKLSDE Lab
Beihang University, China
yuweiren@act.buaa.edu.cn

Charu C. Aggarwal
IBM Research
Yorktown, NY, USA
charu@us.ibm.com

Shuai Ma
SKLSDE Lab
Beihang University, China
mashuai@buaa.edu.cn

Haixun Wang
Google Research
California, USA
haixun@google.com

***Abstract*—Network streams have become ubiquitous in recent years because of many dynamic applications. Such streams may show localized regions of activity and evolution because of anomalous events. This paper will present methods for dynamically determining anomalous hot spots from network streams. These are localized regions of sudden activity or change in the underlying network. We will design a localized principal component analysis algorithm, which can continuously maintain the information about the changes in the different neighborhoods of the network. We will use a fast incremental eigenvector update algorithm based on von Mises iterations *in a lazy way* in order to efficiently maintain local correlation information. This is used to discover local change hotspots in dynamic streams. We will finally present an experimental study to demonstrate the effectiveness and efficiency of our approach.**

***Keywords*-graph streams; anomaly detection**

I. Introduction

Network streams have become very popular in recent years because of an increasing number of domains in social and communication network analysis which continuously generate data about interactions between network participants. In such cases, it may be desirable to determine anomalous hot spots, which show abrupt changes both in the magnitudes and patterns of interaction. Such hotspots are often indicative of critical events in the network. This problem is related to that of *abrupt or surprising change detection*, which has been studied in more conventional time-series settings [1], [4], [11].

Anomalous events may arise in a graph stream in two different ways. In the first kind, significant changes in the *magnitudes* of the interaction may occur in the neighborhood of a node in the network. For example, unusual events in the neighborhood of a node in the network may lead to sudden and abrupt changes in the *frequency of* activity levels of the edges. In the second kind, anomalous events may also cause sudden changes in the *structural edge patterns* at a particular node. Specifically, the correlations between the nodes, as measured by the edges between the different nodes in the network, may change significantly. For example, in a bibliographic network, the sudden movement of a faculty member from one university to another may affect local edge correlation patterns. Both these different types of changes may provide valuable insights about the anomalies in the underlying network, and the precise importance of each is application-dependent.

We will see that a localized principal component analysis (PCA) of the edge correlation structure can be very useful from an analytical perspective. Changes in the dominant (local) *eigenvectors* provide useful information on the local edge correlation patterns, whereas changes in the (local) *eigenvalues* provide insights on changes in absolute levels of activity. Of course, a major challenge is that such analytics are traditionally considered too computationally intensive to be applied in an online setting. In this paper, we will use an approach that performs the updates *minimally as required*. Further, since the changes in eigenvectors are significant only for outliers (which by definition are rare), we can reuse the results from previous time periods to efficiently update most of these values incrementally in a stream setting.

Another challenge is that changes could occur over different horizons in time. In some cases, significant anomalies can only be detected over a long period in time, whereas in other cases, they may be detected over a short period of time. There is typically no "optimal" temporal granularity at which the analysis needs to be performed. Hence, we will design an approach that can analyze streams automatically at multiple levels of temporal granularity in an efficient way. Such an analysis can be used to provide *simultaneous* insights over short, medium and long-term horizons.

Related work. The problem of change detection [1], [4], [11] has been studied widely in the literature. Most of the well known work on change detection in graphs measures similarities between successive graph snapshots [7] or uses matching [8] between successive snapshots for anomaly detection [8]. The work in [5] uses spectral methods in order to determine anomalies in time-series of graphs. Summarization-oriented methods for anomaly detection in bipartite graphs are presented in [9], [10]. The problem of anomalous temporal changes in graphs is addressed in [6]. These methods are designed for snapshots of graphs (rather than streams), and are not specifically designed for the problem of hotspot discovery. The works in [2], [3], [12] are designed for various mining applications in graph streams, but are not designed for node-centric hotspot discovery.

1550-4786/13 $31.00 © 2013 IEEE
DOI 10.1109/ICDM.2013.32

IEEE computer society

II. Anomalous Hot Spot Detection: The Model

We assume that we have a network stream, which is defined by a continuously arriving sequence of edges. Since some edges may contain new nodes, the set of nodes also changes over time, though much more slowly. Therefore, the network stream implicitly defines a temporal network $G(t) = (N(t), A(t))$, where $N(t)$ and $A(t)$ are the nodes and edges in the network at time t. We note that $N(t)$ is the *set* of all distinct nodes in the stream at time t, and $A(t)$ is a *sequence* of edges corresponding to all edges received so far. This sequence of edges may possibly contain repetitions, since the same edge may be received multiple times in the network stream. It is assumed that the set $A(t)$ contains the edge (i, j) a total of n_{ij}^t times. $A(t)$ also contains the explicit time-stamps at which the edges were received, since it is maintained as a sequence. Therefore, we assume that the time stamps at which the edge (i, j) is received in $A(t)$ are denoted by $T(i, j, 1) \ldots T(i, j, n_{ij}^t)$. Since the graph is assumed to be undirected, the value of $T(i, j, r)$ is the same as $T(j, i, r)$.

The period over which trends change is often unknown in advance. For example, in a network intrusion detection application, if a sudden attack occurs in the locality of a node, a smaller period is more appropriate. This is clearly application-specific. Therefore, we will design multi-granularity methods which can automatically detect the most relevant periods over which the changes occur in the graph stream. For simplicity, we will first use a fixed period for analysis, and later incorporate multi-granularity.

We note that the relative edge frequencies of the different edges will stabilize over a long period of time (and will not show much relative change in the presence of unusual events), unless the frequencies of the edges are counted by providing greater importance to recent edges. In order to model the temporal aspect of edge frequencies, we assume a decay factor λ associated with each edge. This weight regulates the rate at which the importance of an edge decays over time. We will discuss later, how multiple values of λ are picked in the context of a multi-granularity approach. We define the weight of an edge as follows:

Definition 1: At current time t, the weight of an edge, which arrived at time t_s, is given by $2^{-\lambda \cdot (t - t_s)}$. The half-life of the edge is $1/\lambda$, since its weight reduces by a factor of 2 in that period.

Because of the use of decay weighting, the correlations between the nodes in the network structure will change significantly, as the network evolves over time. Of course, since the sequence $A(t)$ contains *multiple* occurrences of the edge (i, j) at time t, we need to define the weighted frequency $F(i, j, t)$.

Definition 2: The weighted frequency $F(i, j, t)$ of an edge (i, j), which has arrived at multiple times $T(i, j, 1) \ldots T(i, j, n_{ij}^t)$ is defined as the sum of its decay weighted frequencies over all instances of its arrival. Therefore, we have:

$$F(i, j, t) = \sum_{k=1}^{n_{ij}^t} 2^{-\lambda \cdot (t - T(i,j,k))} \quad (1)$$

Since the network is assumed to be undirected, the value of $F(i, j, t)$ is the same as $F(j, i, t)$. The sudden changes in this frequency can be very useful in determining the anomalies from the underlying stream. An immediate consequence of this way of defining exponential decay for $F(i, j, t)$, is that it is bounded above by $1/(1 - 2^{-\lambda})$, as long as each edge is received at most once in a time-stamp. This follows immediately from the bound on the geometric series $\sum_{t=0}^{\infty} 2^{-\lambda \cdot t}$.

Observation 1: The frequency $F(i, j, t)$ is bounded above by $1/(1 - 2^{-\lambda})$.

We note that the value of the frequency is often dominated by the relative recency of arrivals. Therefore, in hotspots of suddenly increasing activity, the corresponding edges may show high frequency. On the other hand, a sudden unexpected decrease in the activity will also show up as a reduction in the underlying frequencies.

At time t, we define the locality $S(i, t)$ of the node i as the nodes which are directly connected to node i with an edge. Since most networks are sparse, the value of $S(i, t)$ is typically much smaller than the number of nodes in the network. For example, in a bibliographic network such as DBLP, the number of nodes may be of the order of a million, whereas the average degree of a node is less than 20. Therefore, these sets are fairly compact, and can be analyzed easily in practice. However, the value of $S(i, t)$ will change over time, as new edges are added to the network. We assume that the index of the nodes in $S(i, t)$ are denoted by $\{j_1^i(t) \ldots j_{|S(i,t)|}^i\}$.

Next, we will define methods for computing the temporal correlations between edges in the network stream. This is quite tricky, because the edges in the stream may not appear at *exactly* the same time, and yet we would somehow like to capture the temporal correlation between them. It turns out that the decay weighting provides a powerful way of computing the correlations, at a level of granularity which is regulated by the decay rate. For a pair of edges (i, j) and (k, l), the correlations between the decay weighted frequencies provide an interesting way to measure the correlations between the attributes.

Definition 3 (Decay-based Frequency Product): The decay-based frequency product $P(e_1, e_2)$ of a pair of edges $e_1 = (i, j)$ and $e_2 = (k, l)$ at time t is defined as the sum of the pairwise products of the aggregate frequencies associated with the edges at each time stamp.

$$P(e_1, e_2) = \sum_{r=1}^{t} F(i, j, r) \cdot F(k, l, r) \cdot 2^{-2 \cdot \lambda \cdot (t - r)} \quad (2)$$

We note that $P(e_1, e_2) = P(e_2, e_1)$. Furthermore, $P(e_1, e_1)$ is defined by matching each edge with itself. Intuitively,

the additional decay term is added to account for a lower importance to the older frequencies $F(i,j,r)$. The decay-based frequency product captures the correlations between edges very well (relative to the individual edge frequencies). This is because when the edges arrive together, this product is usually much higher than if the edges arrive individually. The decay-based frequency product can be used in order to define a decay-based frequency matrix as follows:

Definition 4 (Decay-based Product Matrix): The decay-based product matrix at a node i is the $|S(i,t)| \times |S(i,t)|$ matrix $M(i,t)$, in which each row or column k corresponds to a node $j_k^i(t)$ in the locality of i. The value of the (k,l)th entry is equal to the value of the decay-based frequency product between $(i, j_k^i(t))$ and $(i, j_l^i(t))$.

An immediate observation is that the decay-based product matrix is positive semi-definite.

Lemma 1: The decay-based product matrix $M(i,t)$ is positive semi-definite.

Proof: Omitted. ■

Since the matrix $M(i,t)$ represents the correlation structure of the locality of a given node, its largest eigenvectors and eigenvalues provide key insights about the underlying edge-correlation dynamics. We define the characteristic vector and value of the locality of the node i as follows:

Definition 5 (Characteristic Vector/Value): The characteristic vector $\overline{W(i,t)}$ of node i is the unit eigenvector of its decay-based product matrix $M(i,t)$, which corresponds to the largest eigenvalue. The characteristic value is equal to the largest eigenvalue $\alpha(i,t)$.

What is the significance of the characteristic vector and the characteristic value? The characteristic vector contains one coefficient for each node adjacent to i. This is a summary representation of the correlations of the edges incident on i. If the coefficients for two nodes j and k are both of the same sign, it implies that the edges (i,j) and (i,k) are positively correlated in their arrival times. Once the characteristic vector has been defined, it can be used in order to define the changes both in terms of correlations and absolute values.

Definition 6 (Activity Correlation Change): The activity correlation change $C(t_1,t_2)$ at node i between t_1 and t_2 is defined as $1 - \overline{W(i,t_1)} \cdot \overline{W(i,t_2)}$.

We note that since the characteristic vector is normalized to be a unit vector, the dot product between the vectors at times t_1 and t_2 is less than 1. When the two vectors are identical, the dot product is 1, and therefore the change is 0. On the other hand, when the two vectors are orthogonal, this corresponds to the greatest change, and the change value is 1. Typically, such a measurement is likely to be most informative, when the time period (t_1,t_2) is chosen to have a span which is dependent on the half-life of the data points for the corresponding decay parameter. The corresponding *half-life* activity correlation change at level λ and time t, is defined as the change between the current time t and that at a time $t - 1/\lambda$. We note that $1/\lambda$ is the half-life span of the data point. Therefore, we have:

Definition 7 (Half-life Correlation Change): The half-life correlation change $HC(i,t,\lambda)$ for node i at decay level λ and time t is the change occurring over a half-life period of a data point, and is equal to $C(i,t-1/\lambda,t)$.

The afore-mentioned quantity measures the changes in the correlations. It is also possible to measures the absolute changes in the dominant trend, by examining the largest eigenvalue. Correspondingly, the magnitude change is defined as the absolute change in the characteristic value.

Definition 8 (Activity Magnitude Change): The activity magnitude change between t_1 and t_2 for node i is denoted by $\gamma(i,t_1,t_2)$, and is equal to $\alpha(i,t_2) - \alpha(i,t_1)$.

High positive values correspond to a substantial increase in activity between t_1 and t_2, and high negative values correspond to a substantial decrease in the activity level. We define the half-life magnitude change in a similar way.

Definition 9 (Half-life Magnitude Change): The half-life magnitude change $HA(i,t,\lambda)$ for node i at decay level λ and time t is the change occurring over a half-life period of a data point, and is equal to $\gamma(i,t-1/\lambda,t)$.

Thus, by measuring the change over the half life of the decay parameter λ, it is possible to obtain a good estimate of the level of change occurring within the window. On the other hand, windows much smaller or much larger than the half-life will lose their ability to provide an effective analysis. Thus, the value of λ provides the level of granularity of the analysis. In the next section, we will discuss the efficient implementation of a multi-granularity approach.

III. Efficient Algorithms for Change Monitoring

In this section, we will study efficient algorithms for change monitoring in network streams. A number of computational and effectiveness challenges arise in this context.

- The determination of principal components need to be *locally performed over all nodes in the network*. While each local matrix is relatively small, and dependent only on the node-degree, this can be a problem for a large network containing many nodes, especially since eigenvector computation is expensive.
- A decay-based approach ensures that the relative frequencies change at *each time-stamp*. This implies that *all matrices, eigenvectors, and eigenvalues* need to be updated at each time-stamp. In a fast stream setting, this can clearly be impractical.
- The anomalous trends may show up over different time-horizons. If the half-life of the analysis is chosen to be too short, it will result in highly changing set of characteristic vectors. On the other hand, if the half-life of the analysis is chosen too long, the characteristic vectors will be too stable. In other words, *multi-granularity analysis* needs to be performed in an efficient way.

In order to analyze the nature of the variations in the patterns, we will examine the case where a single edge (k, l) is added to the network at current time t. Let us also assume that the previous edge added to the network was at time $t' < t$. It is evident that the changes to the eigenvectors and eigenvalues of the different nodes are impacted *by a combination of decay-based activity and the addition of the new edge*. As long as $t' < t$, it is certain that every tracked quantity $M(i, t)$ and $F(i, j, t)$ has changed for each node and edge. The number of entries for these variables alone is greater than the number of edges in the network. Therefore, it would seem, at least at first sight, that even the update process in a fast network stream is likely to be intractable from a computational perspective.

However, it turns out that many of the temporal updates at nodes can be performed in a lazy fashion, and maintained implicitly, as long as there are no changes in the locality of the node because of arrivals of edges.

Observation 2: As long as there are no new arrivals of the edge (i, j) in (t', t), we have $F(i, j, t) = F(i, j, t') \cdot 2^{-\lambda \cdot (t-t')}$.

The correctness of this observation follows almost immediately from how $F(i, j, t)$ is defined. The behavior of the decay-matrix is slightly more involved. By combining the definition of each entry of $M(i, t)$ in Equation 2, and Observation 2, the following may be obtained:

Lemma 2: As long as there are no arrival of edges incident on node i in (t', t), the (k, l)th entry of the matrix $M(i, t)$ can be expressed completely in terms of the corresponding quantities at t' as follows:

$$[M(i,t)]_{kl} = [M(i,t')]_{kl} \cdot 2^{-2\lambda \cdot (t-t')} + \\ +(t-t') \cdot 2^{-2 \cdot \lambda \cdot (t-t')} \cdot F(i, j_k^i(t'), t') \cdot F(i, j_l^i(t'), t')$$

Thus, the updates to these quantities are *expressed purely as a function of the values of the quantities at t', and the value of $(t - t')$*. Therefore, as long as we know the last time-stamp t' at which the matrix was updated together with the corresponding values of $F(\cdot)$ and $M(\cdot)$, we do not need to explicitly update its values. This saves a tremendous computational burden, especially when updates to specific nodes are much more rare than a single node.

Four pieces of information was maintained for each node i. Specifically, the information maintained are (i) the locality set $S(i, t)$, (ii) the last time stamp $L(i)$ at which an edge was received incident at node i, (iii) the values of the decay matrix $M(i, L(i))$, and (iv) the value of $F(i, j, L(i))$ for each edge (i, j) incident at node i.

Note that if d_i be the degree of node i, then the bottleneck space requirement here is given by d_i^2 for the decay based product matrix. Therefore, the total space requirement of this maintenance is given by the sum of the squares of the degrees of the nodes in the network. Since real matrices are quite sparse, the total space requirement of this approach may be quite modest. Furthermore, because of the power-law property of networks, most of the space requirements will be consumed by a small number of nodes with high-degree. If desired, the statistics of such nodes can be maintained on disk, and most of the other nodes can be maintained in main memory. Thus, the approach may require occasional updates to disk, but most of the relevant updates will be performed in main memory. This can work effectively in the streaming scenario.

Updates to the afore-mentioned statistics are performed only when new edges arrive. An immediate observation is the following, which follows directly from Lemma 2:

Observation 3: When edge (i, j) arrives, the summary statistics of only nodes i and j need to be updated.

The actual update process first uses the results of Observation 2 and Lemma 2 in order to make the statistics current till time t from a decay perspective. Then, the effect of the arrival of the new edge is incorporated into $F(\cdot)$ by a simple additive operation of adding 1. At that point, the corresponding change in the entries of $M(\cdot)$ because of the change in $F(\cdot)$ is computed and added to the relevant entries of the decay matrix. If desired, it is also possible to re-compute the statistics of nodes at which no new edge has been incident for a long time, in order to be able to estimate significant changes arising out of unusual *inactivity*. However, since such nodes are relatively easy to identify, our focus will be on the more complex case, where we wish to identify anomalous nodes because of large *activity or changes in activity patterns*.

A. Computing Anomalous Changes

A variety of options are available for computing anomalous changes, depending on whether online or offline analysis is desired. If all nodes need to be tracked at the same time, it is advisable to periodically store all the statistics for offline analysis. However, if only a subset of the nodes need to be tracked, online analysis is much more simpler.

For online monitoring, we maintain the time-series values of $HA(i, t, \lambda)$ and $HC(i, t, \lambda)$ continuously over time. A variety of auto-regressive models may be maintained in order to compute the unusual deviations among these values. However, in the streaming scenario, we choose a much simpler solution in which the mean and standard deviations of these values are maintained continuously over time. Specifically, the square sum of these values, sum of these values and the number of values are continuously maintained. This is easy to do in a data stream. The mean and standard deviations can be easily computed from these values. For example, let $\mu^A(i, t, \lambda)$ and $\sigma^A(i, t, \lambda)$ be the mean and standard deviation of $HA(i, t, \lambda)$. Then, the Z-number of the quantity can be computed as follows:

$$ZValue = \frac{HA(i,t,\lambda) - \mu^A(i,t,\lambda)}{\sigma^A(i,t,\lambda)} \tag{3}$$

When the value of this deviation is larger than 3, it is flagged as an anomaly.

If desired, this approach can also be used for more detailed offline analysis by dumping all the frequencies and decay-based matrices to disk. In order to compute the anomalous changes, we always dump all the statistics to disk, when the time-stamp is divisible by $1/\lambda$. Recall that the half-life represents the level of granularity at which the analysis is performed. Depending upon the space availability, either all the dumped statistics may be maintained, or only the last k dumps may be maintained.

B. Multi-Granularity Analysis

The previous section assumes that a particular value of λ is used for analytical purposes. However, in practice, the value of λ may not be known in advance. The value of λ really depends upon the expected time periods over which the significant changes in the network structure are expected. For example, in a DBLP network, this value could range anywhere between 2 and 20 years, and in a social network, this value may range from a month to a year. However, for a given application, it is reasonable to assume that an approximate idea of the approximate *ranges* in which the changes could occur are known. Therefore, we assume that these minimum and maximum ranges are denoted by t_{min} and t_{max} respectively. Correspondingly, we choose the minimum and maximum half lives of the decay-based approach to t_{min} and t_{max} respectively. Therefore, we have $\lambda_{min} = 1/t_{max}$, and $\lambda_{max} = 1/t_{min}$. We choose $\log_2(\lambda_{max}/\lambda_{min})$ different values of λ denoted by λ_{max}, $\lambda_{max}/2$, $\lambda_{max}/4$ and so on, till the value falls below λ_{min}. For each of these logarithmic values the model is maintained, and the change values are independently monitored. Note that smaller values of λ require disk statistic dumps which are much more infrequent. In fact, since the storage requirement of a successively smaller value of λ requires at most half as much space, the total storage requirement is dominated by that required by the case for the largest value of λ. Furthermore, the change values are reported more infrequently for smaller values of λ. In this multi-granularity approach a change is reported to be significant, if it is found by *any of the detectors* for different values of λ. This approach increases the performance and main memory requirements by a factor of at most $\log_2(\lambda_{max}/\lambda_{min})$. In many applications, this may be a small number, e.g., 3 or 4.

IV. Experimental Study

In this section, we will present an experimental study of our HotSpot framework on real data sets.

A. Experimental Settings

We first introduce the settings of our experimental study. **Data Set**. We chose Internet Movie Database (IMDB) to test our approach. IMDB is an online collection of movie data, which contains data from 1892 to 2012 from the IMDB database to generate the graph stream. We selected works with director and actor information. A piece of work is considered a graph, director-actor pairs being edges. The IMDB data set contains a total of 1,008,978 records, and a graph with 2,214,210 nodes and 13,529,524 edges.

Algorithms and Implementation. HotSpot algorithm was implemented with Microsoft Visual C++. We adopted Intel Math Kernel Library(Intel MKL) 11.0 update 1 (http://software.intel.com/en-us/intel-mkl) for eigenvalue computation. All experiments were run on a PC with an Intel Core i5-2400 CPU@3.10GHz and 16GB of memory.

B. Experimental Results

We next present our findings. In the experiments, we used four half-life values: 1, 2, 4 and 8 years (*i.e.,* λ = $1/1$, $1/2$, $1/4$, and $1/8$) to detect anomalous changes. We used all these half-life values in the multi-granularity analysis.

1. Case Study. A hot spot is essentially a node such that its interaction with neighbors has significant changes or its structural edge patterns have a sudden change. We report an example hot spot discovered by algorithm HotSpot, which provide intuition and insights why it is able to effectively determine anomalous hot spots.

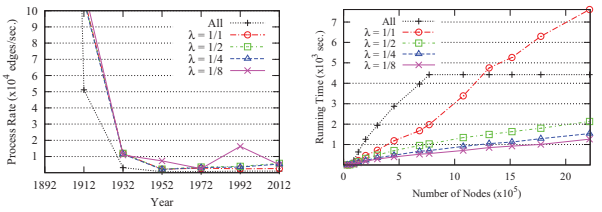
Hot Spot: David Butler, Director, who was one of the earliest movie artists, started his career as an actor in 1910 when he was 16 and shot his first film as a director in 1927. In this example, HotSpot algorithm found several interesting turning points and remarkable periods during his career.

(1) When the half-life was set to 1 year, David Butler was detected as a hot spot in years 1929, 1934, 1943, 1949, 1956 and 1962. We next give an analysis.

For example, year 1929 marked his first success as a director. In 1927 he started as a director in a movie with 6 of the actors represented in the data. According to IMDB, the activity level was almost as stable as his actor records before 1927. In 1928 he directed 3 films with a small crew of around 5 people, and acted in an additional film. In 1929, he was again detected as a hot spot for directing 6 films with many actors, which caused sudden changes of activity levels and structural edge patterns. In a director-actor relationship, a director forms edges with all the actors in one movie, while an actor forms edges with each director only. This typically results in sudden changes on directors.

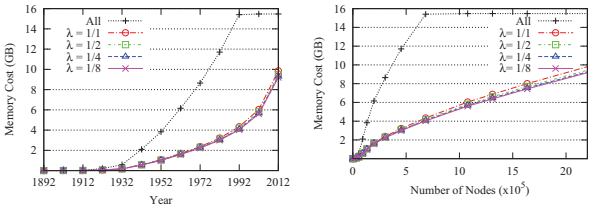
(2) When half-life is 2 years, the algorithm tells us an artist's take-off period. For David Butler, the detected period was 1956-1957, 1962-1963, corresponding to the start and peak of his TV series career. When half-life period is longer than 1 year, the activity level is balanced within the period. For example year 1943 was prominent when half-life is 1 year but it was balanced and hence not detected due to inactivity in 1942 when half-life is 2 years.

(4) When half-life is 4 years, the algorithm tells us an artist's peak period of his entire career. For David Butler, the detected peak period was 1956-1963, which was almost his entire TV series career. He retired in 1967.



(a) IMDB (b) IMDB

Figure 1: Processing Speed Evaluation



(a) IMDB (b) IMDB

Figure 2: Space Overhead Evaluation

(5) When half-life is 8 years, David Butler was not detected as a hot spot as this half-life is too large to detect effective changes due to the change balance.

2. Performance Evaluation. In this section, we report the performance of HOTSPOT in terms of efficiency. The results for IMDB are illustrated in Figure 1. Figure 1(a) shows the processing rate, in terms of the number of edges that HOTSPOT processed per second, where the X-axis represents the progression $w.r.t$ years. Figure 1(b) shows the total processing time, where the X-axis represents the progression $w.r.t$ the number of nodes.

The processing rate of the HOTSPOT algorithm drops with the progression of the stream. This is because for any node i, its locality set $S(i,t)$ grows and the decay-based product matrix grows larger in scale. As a result, it takes more time to solve the eigenvalue problem. Moreover, the eigenvalue computation often took up to 80% of the entire time.

The processing rate fluctuated with the progression of time. The sudden drops reflect the time when the eigenvalue computation was performed. The processing time for each half-life value is different, since for smaller half-life values, the statistics update is more frequent.

Figure 1 shows that the algorithm has a performance of 980 edges per second for IMDB, which is quite significant from a practical perspective in many specific applications.

3. Space overhead evaluation. We report the memory cost of HOTSPOT $w.r.t$ the progression of time and the number of nodes. The results are illustrated in Figure 2.

Figure 2(a) shows that the usage of memory increases with the progression of time, and 2(b) shows that the usage of memory increases with stream length. This is because the locality set $S(i,t)$ of each node i grows constantly. It took 16GB from years 1892 to 1995. The memory usage for distinct half-life values are almost the same since identical set of information is maintained.

Summary. In summary, we conclude the following.
(1) Algorithm HOTSPOT is effective in finding anomalous hot spots in graph streams. Sudden structural edge pattern changes and significant activity level changes are reflected by changes in the eigenvector system. For small half-life values, the algorithm detects temporary activity bursts. For medium half-life values, it detects the activity level increments during a period. For large half-life values, it could tell a period of a peak activity level in a long span of time.

(2) Algorithm HOTSPOT is efficient, and can process several hundred edges per second. Its bottleneck lies in the hardness of the computation of eigenvalues. The reduction of space overhead will be addressed by our future optimization.

V. CONCLUSIONS

The problem of hotspot discovery can help identify important localized regions of change in a fast graph stream. Such regions of change are often associated with critical events. In practice, such hot-spots may be difficult to find both because of the stream setting, and the transient nature of hot spots. We used a dynamic PCA-based method in order to identify significant neighborhoods of change. The experimental results show the effectiveness in discovering useful hot-spots from the underlying data stream.

Acknowledgments. Research of C. Aggarwal was sponsored by the Army Research Lab. and was accomplished under Coop. Agreement No. W911NF-09-2-0053. Shuai is supported in part by NSFC grant 61322207, NGFR 973 grant 2014CB340304 and 863 grant 2013AA01A213, and MOST grant 2012BAH46B04.

REFERENCES

[1] C. Aggarwal, *Outlier Analysis*, Springer, 2013.
[2] C. Aggarwal, On classification of graph streams, *SDM*, 2011.
[3] C. Aggarwal, Y, Zhao, P, Yu. Outlier detection in graph streams. *ICDE*, 2011.
[4] T. Fawcett, F. Provost. Activity Monitoring: Noticing Interesting Changes in Behavior. *KDD*, 1999.
[5] T. Ide, H. Kashima. Eigenspace-based Anomaly Detection in Computer Systems. *KDD*, 2004.
[6] M. Mongiovi, P. Bogdanov, R. Ranca, A. Singh, E. Papalexakis, C. Faloutsos. NetSpot: Spotting Significant Anomalous Regions on Dynamic Networks. *SDM*, 2012.
[7] B. Pincombe. Anomaly Detection in Time Series of Graphs using ARMA Processes. *ASOR Bulletin*, 24(4): 2–10, 2005.
[8] P. Showbridge, M. Kraetzl, D. Ray. Detection of Abnormal Change in Dynamic Networks. *IDC*, 1999.
[9] J. Sun, S. Papadimitriou, P. Yu, C. Faloutsos. Graphscope: Parameter-free Mining of Large Time-Evolving Graphs. *KDD*, 2007.
[10] J. Sun, H. Qu, D. Chakrabarti, C. Faloutsos. Neighborhood Formation and Anomaly Detection in Bipartite Graphs. *ICDM*, 2005.
[11] K. Yaminshi, J. Takeuchi. A Unified Framework for Detecting Outliers and Change Points from Time Series Data. *KDD*, 2002.
[12] P. Zhao, C. Aggarwal, M. Wang. Gsketch: On query estimation in graph streams, *VLDB*, 2012.