2014 ACM SIGKDD INTERNATIONAL CONFERENCE ON
KNOWLEDGE DISCOVERY AND DATA MINING

ODD^2 Workshop on Outlier Detection & Description under Data Diversity

August 24th, 2014

New York City, USA

Editors:
Charu Aggarwal
IBM T. J. Watson Research Center
Leman Akoglu
Stony Brook University
Emmanuel Müller
Karlsruhe Institute of Technology & University of Antwerp
Raymond T. Ng
University of British Columbia

Preface

We consider outlier mining as the threefold challenge of **outlier detection** (quantitative analysis), **outlier description** (qualitative analysis), outlier models for **diverse data** (relational databases, data streams, text documents, heterogeneous graphs, ...). Each of these topics is of great interest by itself, however, they are rarely considered in unison, and literature for these tasks is spread over different research communities. Especially, if we consider heterogeneous data sources with different data types users are overwhelmed by complex data. They do not only require a raw detection of outlier, they require a deeper description of why an objects seems to be outlying. Further, outlier models published for different data types proposed by different communities could benefit from each other, however, are rarely published in the same venues (e.g. outliers in text data published in *information retrieval community*, outliers in high dimensional data in *database community*, or graph outliers in *social network community*). As main contribution of this workshop we aim to bridge this gap between **outlier detection – outlier description – outlier models in diverse data** and provide a venue for knowledge exchange between these different research areas for a corroborative union of analyses in outlier mining.

Following the tradition of the first ODD Workshop at KDD 2013 the technical program of ODD^2 at KDD 2014 again demonstrates the strong interest from different research communities. In particular, we have six peer-reviewed papers covering multiple research directions of outlier mining in diverse data. They passed a competitive selection process ensuring high quality publications. Furthermore, we are pleased to have two excellent speakers giving invited talks that provide an overview on challenges in related fields: Srinivasan Parthasarathy (Ohio State University) and Ambuj Singh (University of California Santa Barbara) contribute with their recent work in this area. And finally, in the spirit of previous workshops, the panel opens for a discussion of state-of-the-art, open challenges, and visions for future research. It wraps up the workshop by summarizing common challenges, establishing novel collaborations, and providing a guideline for topics to be addressed in following workshops and journal special issues.

As organizers of this workshop, we are grateful for the support of the ACM SIGKDD conference, assisting us with all organization issues. Especially we would like to thank the workshop chairs, Alejandro Jaimes and Dragos Margineantu. Furthermore, we also gratefully acknowledge the ODD^2 program committee for conducting thorough reviews of the submitted technical papers. We are pleased to have some of the core researchers of the covered research topics in the ODD^2 committee.

New York City, USA, August 2014

Charu Aggarwal
Leman Akoglu
Emmanuel Müller
Raymond T. Ng

Workshop Organization

Workshop Chairs

Charu Aggarwal	IBM T. J. Watson Research Center
Leman Akoglu	Stony Brook University
Emmanuel Müller	Karlsruhe Institute of Technology & University of Antwerp
Raymond T. Ng	University of British Columbia

Program Committee

Fabrizio Angiulli	University of Calabria
Ira Assent	Aarhus University
Albert Bifet	HUAWEI Noah's Ark Lab
Rajmonda Caceres	MIT Lincoln Laboratory
Varun Chandola	Oak Ridge Nat. Lab.
Polo Chau	Georgia Tech
Sanjay Chawla	University of Syndey
Christos Faloutsos	Carnegie Mellon University
Jing Gao	University of Buffalo
Manish Gupta	Microsoft, India
Bartosz Krawczyk	Wroclaw University of Technology
Arun Maiya	Institute for Defense Analyses
Daniel B. Neill	Carnegie Mellon University
Spiros Papadimitriou	Rutgers University
Joerg Sander	University of Alberta
Thomas Seidl	RWTH Aachen University
Koen Smets	University of Antwerp
Hanghang Tong	CUNY
Ye Wang	The Ohio State University
Osmar Zaiane	University of Alberta
Arthur Zimek	LMU Munich

External Reviewers

Sergej Fries	RWTH Aachen University
Koosha Golmohammadi	University of Alberta
Suchismit Mahapatra	University at Buffalo

Table of Contents

Invited Talks

An ODE (Outlier Detection and Explanation) to Analyzing Data with Anomalies: Algorithms and Applications 1
Srinivasan Parthasarathy

Detection of Significant Network Processes 2
Ambuj Singh

Research Papers

Anomaly Detection in Networks with Changing Trends 3
Timothy La Fond, Jennifer Neville and Brian Gallagher

Change-point detection in temporal networks using hierarchical random graphs 13
Leto Peel and Aaron Clauset

Real time contextual collective anomaly detection over multiple data streams 23
Yexi Jiang, Chunqiu Zeng, Jian Xu and Tao Li

STOUT : Spatio-Temporal Outlier detection Using Tensors 31
Yanan Sun and Vandana Janeja

An Ensemble Approach for Event Detection and Characterization in Dynamic Graphs 41
Shebuti Rayana and Leman Akoglu

Learning Outlier Ensembles: The Best of Both Worlds - Supervised and Unsupervised 51
Barbora Micenkova, Brian McWilliams and Ira Assent

An ODE (Outlier Detection and Explanation) to Analyzing Data with Anomalies: Algorithms and Applications

Srinivasan Parthasarathy
Ohio State University

Abstract: In this talk I will briefly discuss recent advances in outlier detection, with a focus on distance-based techniques and discuss possible future directions in the context of rank-driven interactive analysis and data-guided explanations and visualizations. Time permitting we will examine such techniques in the context of real world analysis of multi-modal data including time series, graphs, text and factorial designs to help understand interactions among various optimizations for such algorithms.

Detection of Significant Network Processes

Ambuj K Singh
Department of Computer Science
University of California Santa Barbara

Abstract: In order to understand complex networks, we need to characterize the dynamic processes occurring within them. Examples of such processes include network congestion in the Internet or a transportation network, or the spread of malware through an online social network. The detection of such dynamic processes requires a model of underlying behavior using which inferences about significance or anomalous behavior can be made. Detecting anomalies in networks is a well-understood problem when restricted to only the graph structure (e.g., communities, structural holes), but there has been limited work on networks with node/edge attributes. When node attributes are allowed to change over time, the smooth evolution of network substructures can be used to detect significant network processes that grow, shrink, and merge over time. I will discuss an approach that compares the value at a node/edge with a background distribution, and uses a positive score to indicate significance. I will discuss methods for detecting highest scoring subgraphs (fixed substructures, varying time intervals), and for detecting smoothly varying substructures. Finally, I will discuss methods that detect significant network substructures over two classes of networks in order to explain their differences. Examples will be drawn from information networks and brain networks to illustrate the methods.

Anomaly Detection in Networks with Changing Trends

Timothy La Fond[1], Jennifer Neville[1], Brian Gallagher[2]
[1]Purdue University, [2]Lawrence Livermore National Laboratory
{tlafond,neville}@purdue.edu, bgallagher@llnl.gov

ABSTRACT

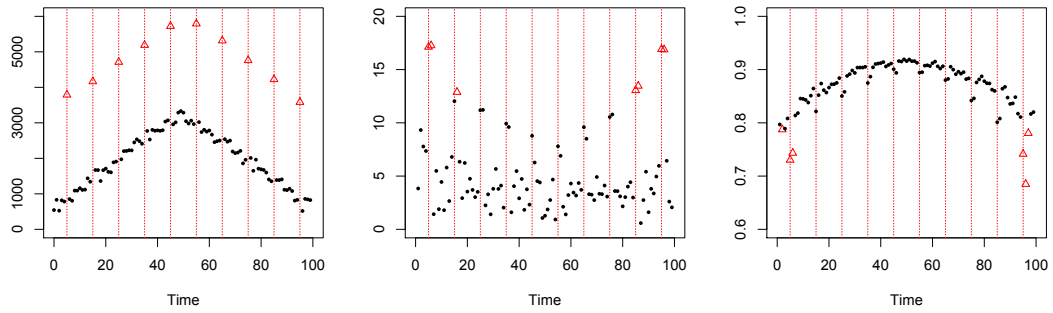
Dynamic networks, also called network streams, are an important data representation that applies to many real-world domains. Many sets of network data such as e-mail networks, social networks, or internet traffic networks have been analyzed in the past mostly using static network models and are better represented by a dynamic network due to the temporal component of the data. One important application in the domain of dynamic network analysis is anomaly detection. Here the task is to identify points in time where the network exhibits behavior radically different from a typical time, either due to some event (like the failure of machines in a computer network) or a shift in the network properties. This problem is made more difficult by the fluid nature of what is considered "normal" network behavior: a network can change over the course of a month or even vary based on the hour of the day without being considered unusual. A number of anomaly detection techniques exists for standard time series data that exhibit these kinds of trends but adapting these algorithms for a network domain requires additional considerations. In particular many different kinds of network statistics such as degree distribution or clustering coefficient are dependent on the total network degree. Existing dynamic network anomaly detection algorithms do not consider network trends or dependencies of network statistics with degree and as such are biased towards flagging sparser or denser time steps depending on the network statistics used. In this paper we will introduce a new anomaly detection algorithm dTrend which overcomes these problems in two ways: by first applying detrending techniques in a network domain and then by using a new set of network statistics designed to be less dependent on network degree. By combining these two approaches into the dTrend algorithm we create a network anomaly detector which can find anomalies regardless of degree changes. When compared to current techniques, dTrend produces up to a 2x improvement in F1 score on networks with underlying trends.

ODD^2*'14,* August 24th, 2014, New York, NY, USA.

1. INTRODUCTION

Statistical network analysis is a domain with a huge range of applications and has seen a great deal of study in recent years. With the rise of social networking sites like Facebook or Twitter and other large networks there has been a push to understand these sorts of networks in more detail either for monetization of the network data or for the detection of malicious or criminal activity. However, most of the work already done has treated the networks as primarily static structures, aggregating all activity within a time period into a single network structure [2, 11]. The reality of these networks is that they are dynamic and evolving entities: every message or action that takes place in the network has a certain time of occurrence associated with it. Thus the introduction of dynamic network models offers the possibility of even greater understanding of the facets of these networks [9]. There are many challenges, of course, associated with dynamic models: the dimensionality of the data is vastly increased making learning models more expensive and difficult, and techniques from standard time series analysis are often not directly applicable to data with a network structure.

In this work we will focus on the task of anomaly detection in dynamic network streams. An anomaly is defined as a point in time where the network behavior deviates radically from the norm. An example of an anomalous event could be a DDoS attack on a computer network or a flurry of e-mail messages between a company's employees as a deadline approaches [13]. The two main difficulties arise from defining what network behavior is exactly and what constitutes normal behavior. As a complex data structure, a wide array of statistics have been proposed to represent the behavior of networks including network degree, degree distribution, diameter, transitivity, and more [11], and selecting the appropriate measure can be difficult. In addition, defining the typical behavior of the network requires a few considerations as well. A network can change as time progresses; networks like Facebook are constantly adding new nodes and connections, and network traffic will increase during peak hours and fall off again afterwards. This requires that our definition of normality change to match the concept drift that the network experiences. While trend shift is not a new concept in time series analysis, network-focused approaches to this problem generally lack this consideration. Figure 1 (a) shows a simple example of a network with a changing trend and a set of anomalies represented by the vertical lines. The Y-axis represents the anomaly score and the red triangles are points flagged as anomalous by our algorithm.



(a) (b) (c)

Figure 1: An example of an underlying network trend confounding anomaly detectors. Subfigure (a) shows the edge count of the network and the anomalies correctly flagged by the dTrend detector. Subfigure (b) shows the Netsimile detector, and (c) is the Deltacon detector.

Subfigures (b) and (c) show the anomaly scores and points identified as anomalous by two current statistics: Netsimile and Deltacon respectively [17, 18]. Clearly the accuracy of these detectors is hindered by the underlying trend, detecting anomalies in the sparser networks (in the beginning and end of the series) but not in the denser ones (the middle of the series). Netsimile primarily has false negatives where it fails to flag the anomaly, while Deltacon has both false positives and false negatives.

We will attempt to combine the complexity of network models with lessons learned from time series analysis in a new anomaly detection algorithm: dTrend. dTrend is a multiple hypothesis test algorithm which applies a detrending step to each set of statistics extracted from the network stream. This detrending step allows us to make accurate detections even as the network evolves over time. In addition, we will define a set of new network statistics which are insensitive to changes in the number of edges in the network. The combination of both these traits allows dTrend to identify a variety of anomalies in networks undergoing change, a case where current algorithms struggle.

Our major contributions are as follows:

- We identify the challenges associated with anomaly detection in networks with trends and demonstrate that current algorithms are unable to accommodate them.
- We adapt detrending, a concept from traditional time series, to the dynamic network domain.
- We define several new network statistics designed to detect anomalies in networks of varying density.
- We combine the two previous contributions into dTrend, a new anomaly detector for dynamic networks using multiple hypothesis tests, and use synthetic data to quantify the advantages over other techniques.
- We apply dTrend to a real-world dataset to show its ability to recover critical events from a network timeline.

2. RELATED WORK

One common dynamic network anomaly detection method is to use network distance metrics to find time slices that are extremely different from the network in the previous time step. Examples of network distance metrics include Netsimile and Deltacon [17, 18]. Netsimile uses aggregates of network statistics and then calculates a distance metric from the aggregates. Deltacon is similar, calculating pairwise node affinity scores and calculating the distance metric from the affinities in two time steps. These metrics can be used for network anomaly detection by comparing the network in each time step with that in the previous and flagging any time where the distance exceeds some threshold. This approach has fewer problems with dynamic networks where there is concept drift compared to algorithms which learn parameters over the whole stream (such as dynamic ERGM discussed below). As each step is compared to the previous, as long as the trend shift is not too rapid the average distance between adjacent time steps is low and there will not be false positives due to the shift. The limitations are that the "memory" of the algorithm is very short, only taking into account the previous time step as a baseline: this can cause anomalies adjacent to similar anomalies to be incorrectly classified as normal.

In addition, the strength of the signal from anomalous time steps can be dependent on the local trends of that time step. Consider graph edit distance, a typical network statistic. The edit distance between two networks with a large network degree has a larger potential maximum than two networks which are sparser: even if the sparse networks have no overlapping edges, the dense networks with their larger edge volume could potentially have a great deal of overlap and still produce a greater edit distance than the completely different sparse networks. The result is that the signal is not consistent with the number of edges in the network: if we select an appropriately sensitive detection threshold for sections of the stream with higher degree, we will not detect anomalies in the sparser sections of the stream vice versa a threshold appropriate for sparse networks will be too sensitive in more dense ones.

Another approach to anomaly detection is a model-based detector. Here testing is done by positing some null model

and reporting a detection if the observed data is unlikely under the model [1]. Some of the most popular models for hypothesis testing are the ERGM family [24], which has been applied to a wide variety of domains [3, 4]. One such example is dynamic ERGM variant for hypothesis testing in a network stream [5, 6, 8, 9, 10, 12]. A standard ERGM model calculates the likelihood for a particular graph configuration using a set of network statistics; a dynamic ERGM simply selects certain temporal network statistics like the edges retained from the previous time step. A dynamic ERGM model can be used with likelihood ratio testing to perform anomaly detection, but the model itself has several major drawbacks. The first is scaleability: the model cannot be learned efficiently on networks with thousands or millions of nodes. The second is an inability to take network trends into account. Only one set of model parameters are learned for the entire network stream, so if the network properties change over time behavior typical for those times will be considered anomalous. One solution is to relearn the model parameters as time progresses in order to keep the model up to date, but this only exacerbates the expensive model learning costs.

A number of algorithms exist for anomaly detection in time series data, including CUSUM-based and likelihood ratio approaches [13, 14, 15, 16, 19, 20]. The basic algorithms will flag any time steps were deviation from a baseline model is detected. To deal with a changing baseline model, the baseline either needs to be re-learned before each detection decision, or a data processing step such as detrended fluctuation analysis (DFA) or detrended moving average (DMA) can be applied to the data as a whole before the detection step [21]. These techniques are sufficient for singular or multivariate time series with trends, but do not directly apply to network data as there is no obvious way to detrend a set of nodes and edges. We will be incorporating techniques from time series and showing how they can be adapted to the domain of dynamic network data.

3. DATA MODEL

Define a dynamic network as a set of graphs $DN = \{G_t\}$, $G_t = [V_t, E_t]$ where V_t is the active node set at time t and E_t is the set of edges at time t. Let t be drawn from the set $\{t_1, t_2 ... t_k\}$ where each time represents a sequential slice of network activity with a uniform width t_Δ and without gaps. We will be using a multigraph representation for the component graphs, so two nodes may have multiple edges between them at a particular time step . Let $e_{ij,t}$ denote the number of edges between nodes i and j at time t.

Define $F(t)$ as a graph generation function which outputs networks in the space of $\{G_t\}$. We assume that the majority of the data is "normal" in the sense that it was created by the $F(t)$ process. We also assume that there are a number of anomalous points in the series where the graph is the output of some alternative model $F^*(t)$. We can now characterize the problem of anomaly detection as a standard hypothesis test where the null hypothesis is that G_t was generated from function $F(t)$ and the alternative that it was generated some other way: $H_0 : F(t) \models G_t$ and $H_1 : \neg(F(t) \models G_t)$. We will evaluate this hypothesis using a network statistic $S(G_t)$. This statistic could be graph edit distance, clustering coefficient, or a more complex statistic like a Netsimile score on the previous network. Once a statistic is selected anomalies can be detected by rejecting the null hypothesis at any time step where $S(G_t)$ is less than some threshold k_s.

3.1 Data Detrending

To incorporate the network trend into our detection, the statistics we selected for dTrend were residuals of linear segmentation fits to common statistics [21]. An example of such a segmentation can be seen in figure 5 (d). The algorithm is as follows: calculate a network statistic S on all times t, then learn a linear segmentation on these statistics $[\alpha_t, \beta_t] = Fit_{LS}(S(G_1, G_2, ...G_{t_k})$, where α_t, β_t are the intercept and slope of the segment that t belongs to. The residual at any time is then given by $R_S(G_t) = S(G_t) - \alpha_t - t * \beta_t$. If we assume that the residuals follow a normal distribution, we can then set a threshold for detection using a p-value of 0.05. Although many different options for the S statistic are available, in the next section we will show that many of these statistics are biased based on the degree of the network even after being detrended.

4. EDGE VOLUME BIASES

As stated before, several of the most common network statistics are biased with respect to the number of edges of the network which can confound the detection process. In this section we describe the most commonly used network statistics and will demonstrate how the range of values are dependent on $|E_t|$.

4.1 Graph Edit Distance

The graph edit distance is defined as the number of edits that one network needs to undergo to transform into another network. In the context of a dynamic network the statistic for edit distance at time t is calculated using the graph in $t-1$ as the comparison. More formally, the graph edit distance at time t is defined as

$$GED(G_t) = |V_t| + |V_{t-1}| - 2*|V_t \cup V_{t-1}| + |E_t| + |E_{t-1}| - 2*|E_t \cup E_{t-1}|$$

The minimum possible edit distance is

$$abs(|E_t| - |E_{t-1}|) + abs(|V_t| - |V_{t-1}|)$$

where the larger graph contains the smaller as a subgraph, and the maximum possible distance is

$$abs(|V_t| - |V_{t-1}|) + min(|E_t| + |E_{t-1}|, min(|V_t|, |V_{t-1}|)^2$$
$$+abs(|V_t| - |V_{t-1}|) * max(|V_t|, |V_{t-1}|) - |E_t| - |E_{t-1}|)$$

The maximum possible edit distance is given by two non-overlapping networks, or in the case of very dense networks, the minimum possible overlap. Consider two slices where the latter slice is generated by moving each edge in the network to an unoccupied node pair with a probability p, which will give an expected edit distance of $2*p*|E|$. So two very large graphs can have a large edit distance just through random chance, while two very small graphs could be almost totally different and still have a lower edit distance than the two large graphs.

4.2 Clustering Coefficient

The clustering coefficient, which captures the transitivity or tendency to form links between mutual friends, is another common network statistic. The typical measure is simply $CC(t) = 3 * tri(t)/we(t)$ where $tri(t)$ is the number of triangular relationships in the network at time t and $we(t)$

is the number of wedges or open triangles. The minimum clustering coefficient occurs when only wedges are formed initially; after $(|V_t| - 1)$ edges with no triangles (a complete star graph) any additional edges inevitably produce triangles. As the number of edges tends towards $|V_t|^2$, the minimum clustering coefficient tends towards 1. In general, as the density of the network increases the clustering coefficient will increase as well. Consider the case of a random Erdos-Renyi graph with an edge probability of p: the expected number of triangles is

$$\binom{N}{3} * p^3$$

as each triple of vertices needs each edge to exist to form a triangle, and the expected number of wedges is

$$\binom{N}{3} * (3 * p^2 * (1-p) + 3 * p^3) = \binom{N}{3} * 3 * p^2$$

This gives us an expected clustering coefficient of

$$3 * \binom{N}{3} p^3 / 3 * \binom{N}{3} * p^2 = p$$

So even in a network model that explicitly has no concept of transitivity, the clustering coefficient is directly proportional to the density of the network and will increase as more edges appear in the network.

4.3 Degree Distribution

The relationship between density and the degree distribution of the network is fairly simplistic. As the density of the network increases, the number of nodes with a higher degree increases as well. This is particularly true of the high degree nodes, as the total number of edges in the network is a hard limit on the largest degree possible. This relationship is true even if the shape of the degree distribution remains in the same form. Consider a power law degree distribution, one of the most common degree distributions in real-world networks. The probability of a node having degree k in a power law distribution is $p(k) \sim k^{-\gamma}$, and theoretically this distribution allows for nodes of any degree. Naturally, $p(k)$ is zero if $k > |E_t|$ and if k is a significant fraction of the network degree the odds of observing such a node is vanishingly small. Therefore even if two networks have identical underlying power law distributions, if the number of edges in each is very different the observed degree distributions will also be different.

5. PROBABILISTIC EDGE MODEL

The statistics we described above are not good candidates for use in anomaly detection due to the differing signal strength in network streams with varying density. We will now introduce a set of statistics designed to generate a consistent signal regardless of the network trends. To do this we update our model definition to include a probabilistic representation of the edge set. Assume that the process by which edges are generated within a time slice is an iterative procedure where each edge is placed between vertices v_i and v_j with probability $p_{i,j}(t)$ which is dependent on the current time slice. The set of all pairwise edge probabilities in a time slice can be represented with an edge probability matrix $P(t)$ where $P(t)[i,j] = p_{i,j}(t)$ and the total probability mass of the matrix is 1. The edge set E_t in any time step is a set of $|E_t|$ independent samples taken using the probabilities defined by $P(t)$. Note that we are assuming that the edge probability matrix is constant within a single time slice and is only an approximation of the true edge formation affinities. P_t can be easily estimated from E_t by normalizing the multigraph adjacency matrix by the number of edges in the network. This effectively converts the E_t adjacency matrix into two values, the total degree of the network D_t and the edge probability matrix P_t. From this edge probability matrix we can derive a complementary set of network statistics which are not directly related to the network degree.

5.1 Probability Mass Shift

The probability mass shift statistic is the edge probability equivalent to the graph edit distance in the fact that it captures amount of change the network experienced between the current and previous time steps. Define the probability mass shift estimate as the total probability mass that changes between P_t and P_{t-1}:

$$MS(G_t) = \sum_{i,j \in V} |P_t[i,j] - P_{t-1}[i,j]| \quad (1)$$

Any vertices which exist in only one of the networks are considered to be unconnected in the other network. Similar to graph edit distance the probability mass shift captures the amount of change between the network and its previous iteration, but since the edge probability matrix is normalized the probability mass shift will always be a value between 0 and 1. In addition, since each edge change represents a greater proportion of the probability mass in a sparse network the probability mass shift of smaller networks is not inherently less than that of large networks. Consider the example used for graph edit distance where each edge changes with probability p: the expected probability mass shift in this case will be

$$E[MS(G_t)] = p * |E| * 1/|E| = p$$

which does not scale with the number of edges in the network.

5.2 Probabilistic Triangles

As the name suggests, the probabilistic triangles is an approach to capturing the transitivity of the network and an alternative to the traditional triangle count or clustering coefficient statistic. Define the probabilistic clustering coefficient as

$$PT(G_t) = \sum_{i,j,k \in V, i \neq j \neq k} P_t[i,j] * P_t[i,k] * P_t[j,k] \quad (2)$$

This is equivalent to the probability of producing a triangle after inserting three edges into the graph. As these probabilities are not dependent on total degree of the network, the density of the network can vary without a change in the probabilistic clustering coefficient.

5.3 Probabilistic Degree Distribution

The probabilistic degree distribution is the equivalent to a degree distribution calculated on P_t rather than an adjacency matrix. Define the probabilistic degree of a node as

$$PD(v_i) = \sum_{j \in V_t, i \neq j} P_t[i,j] \quad (3)$$

This value can be interpreted as the edge potential of the node in question. The probabilistic degree distribution is simply a histogram of the degree potentials in the network. The width of the bins can be arbitrarily selected but should remain consistent throughout the network stream. We also define a degree potential difference score, which is simply the squared distance of a network slice's degree potential distribution from the prior slice:

$$PDD(G_t) = \sum_{bin \in Bins} \Big(\sum_{v_i \in V_t \cap V_{t-1}} I[PD(v_{i,t}) \in bin] - I[PD(v_{i,t-1} \in bin]\Big)^2 \quad (4)$$

6. ANOMALY DETECTION PROCESS

The edge probability matrix P_t can be estimated at each time step by normalizing the adjacency matrix by the number of edges present in the step:

$$P_t = \frac{A}{|A|}$$

Then calculate the total degree, probability mass shift, probabilistic clustering coefficient, and degree potential distribution score at each time step. This generates a set of standard time series which can be analyzed with a traditional time series anomaly detection technique. dTrend runs on a single statistic stream by applying detrended fluctuation analysis with a linear fit to each segmentation as described in the data detrending section and then sets the threshold for detection at a p-value of 0.05. In practice we do not know the exact type of anomalies present in the data, so rather than selecting a single statistic it is more practical to use a joint dTrend algorithm. The joint dTrend applies dTrend to each statistic separately with a p-value of $\frac{0.05}{\#ofdetectors}$ and reports the union of time steps flagged by the individual detectors. This p-value is consistent with a Bonferroni adjustment on multiple hypothesis tests. Probability mass and probability distribution tend to flag the same types of anomalies, so there are some dependencies between these statistics making this p-value a conservative estimate. It may be possible to find a better performing detection threshold through careful analysis of the dependencies of the individual detectors.

If the network stream has slices that are extremely sparse, it may be necessary to introduce a prior probability to the normalized adjacency matrix. In such a sparse network, a handful of edge changes can radically change the calculated statistics as they represent a larger portion of the probability mass. Another way to represent this problem is that in such a sparse network our ability to estimate the true P_t value is limited, and by introducing a prior probability we can incorporate our uncertainty of P_t. If the prior probability is c we add this value to each node pair in the adjacency matrix, making the new normalized adjacency matrix

$$P_t[i,j] = \frac{e_{i,j} + c}{|E| + c * N^2} \quad (5)$$

The value of c should be very small, small enough that $c * N^2 << |E|$ in denser sections of the network stream.

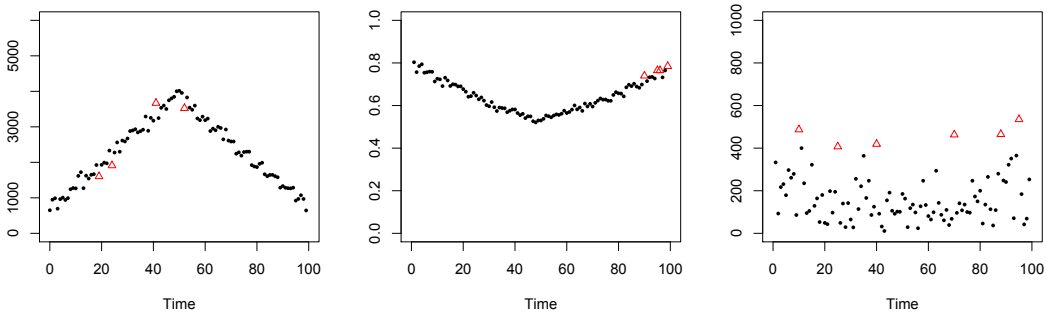
6.1 Complexity Analysis

There are two main components to the dTrend process, the statistic extraction step and the detrending step, and depending on the exact statistics and detrending methods used either one can be the dominating element in the runtime. Statistics like edge count, probability mass, and probability distribution can be calculated at each step in $O(E_t)$ time, making their overall complexity $O(E_t * t)$. Triangle probability, on the other hand, is more expensive as some triangle counting algorithm must be applied. The fastest counting algorithms run in $O(N^2)$ time, making the overall complexity $O(N^2 * t)$ for the whole stream. However, if we make the assumption that the maximum number of neighbors of any node is bounded by k, we can approximate the triangle count with $O(N * k^2 * t)$. Note that any other statistic-based approach such as Netsimile that utilizes triangle count or clustering coefficient must make the same allowances in order to run in linear time.

Linear segmentation in general is super linear in the number of time steps although this can vary with certain allowances. Here we used a top-down approach which runs in $O(t^2 * k)$ where k is the number of segments needed. A bottom-up approach can run in $O(L*t)$ where $L = \frac{t}{k}$, which allows linear in the number of time steps if the number of segments is very large. However, since we expect the number of edges in the entirety of the network stream to be vastly greater than the number of time steps even a quadratic cost in the number of time steps is minimal. Therefore the runtime is generally determined by the cost of the statistic extraction rather than the detrending. In a rare case of a very long, very sparse network stream the cost of detrending can be reduced by using a detrended moving average rather than a linear segmentation, which can run in $O(t)$.

7. SYNTHETIC DATA EXPERIMENTS

These synthetic data experiments will demonstrate that using the set of network statistics defined above can distinguish different types of anomalies when there is a trend of changing network density. The synthetic data trials were created by resampling a stochastic block model [23] a set number of times to create a multigraph. The number of resamples follows a triangular function, starting at 1 and increasing to a maximum of 11 at time 50, then decreasing back down to 1, as shown in figure 2 (a). To simulate a smooth change in the edge count, the exact number of resamples is given by $10 * |1 - (t - 50)/50| + 1 + rand(0, 1)$, so the decimal of the resample value will trigger an actual resampling with probability modulo 1. Into this baseline trend anomalies are introduced at times 5, 15, 25... etc. We created three distinct types of anomalies in separate trials to demonstrate the utility of each statistic in detecting different network changes: edge count anomalies, distribution anomalies, and triangle anomalies. For edge count anomalies the model is a simple three-cluster network with balanced probabilities of intra-cluster links. The class matrix does not change in anomalous time steps but 10 extra resamples are performed on the block model to increase the volume of edges produced. In the distribution trials the baseline blockmodel has the majority of mass concentrated in one cluster. Anomalies are produced by shifting most of this mass to the other two clusters evenly, creating a predominantly two cluster network. For triangle anomalies the baseline model is a strongly wedge-shaped network with few edges within classes. The anomalies are generated by increasing the mass between the two arms of the wedge, significantly increasing the probability of closing triangles.



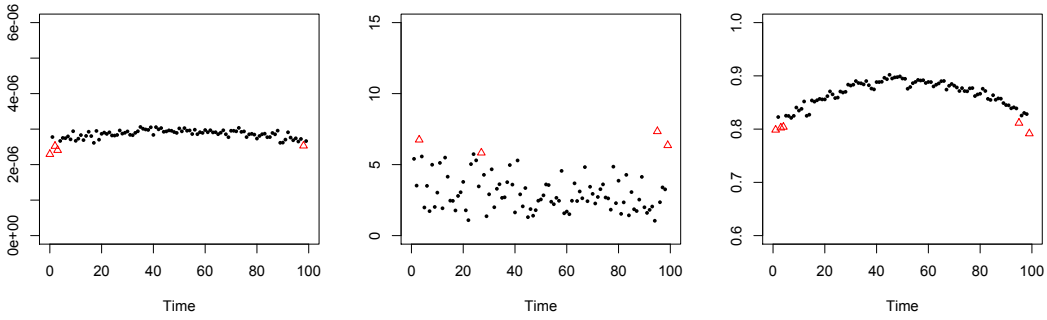


Figure 2: False positives on synthetic network stream with no anomalies and triangular trend. (a) Edge Count Detector, (b) Probability Mass Detector, (c) Probability Distribution Detector, (d) Triangle Probability Detector, (e) Netsimile Detector, (f) Deltacon Detector

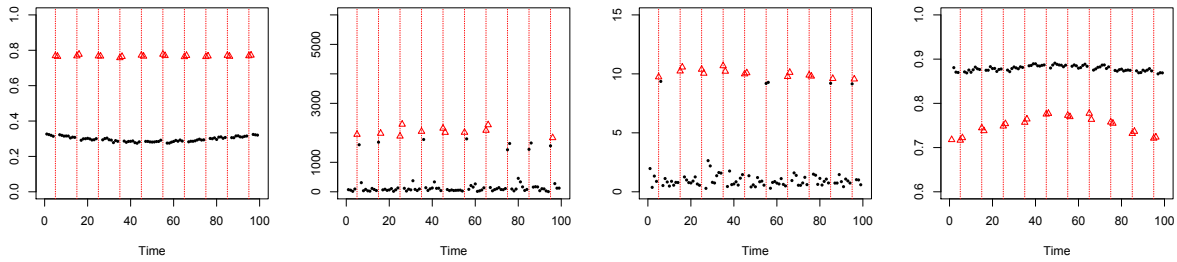


Figure 3: Detection of Distribution Anomalies, anomalies at times 5, 15, 25... (a) Probability Mass Detector, (b) Probability Distribution Detector, (c) Netsimile Detector, (d) Deltacon Detector

The class matrices for the stochastic block model are as follows:

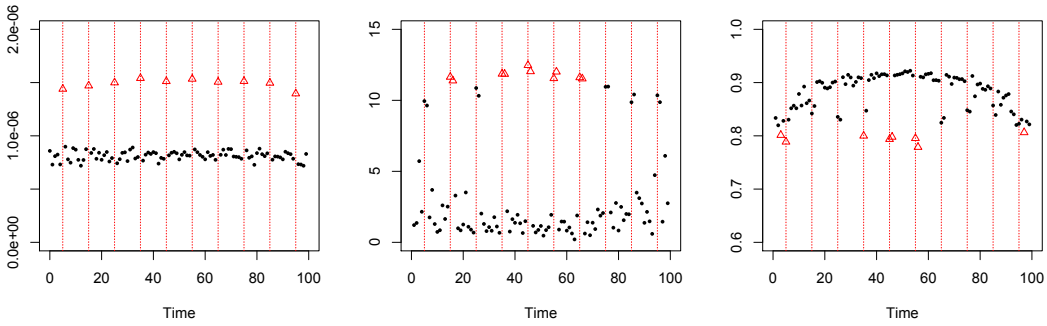
Normal/Anomalous Edge Count
$$\begin{bmatrix} 0.10 & 0.001 & 0.001 \\ 0.001 & 0.10 & 0.001 \\ 0.001 & 0.001 & 0.10 \end{bmatrix}$$

Normal Distribution
$$\begin{bmatrix} 0.20 & 0.001 & 0.001 \\ 0.001 & 0.02 & 0.001 \\ 0.001 & 0.001 & 0.02 \end{bmatrix}$$

Anomalous Distribution
$$\begin{bmatrix} 0.02 & 0.001 & 0.001 \\ 0.001 & 0.11 & 0.001 \\ 0.001 & 0.001 & 0.11 \end{bmatrix}$$

Normal Triangles
$$\begin{bmatrix} 0.01 & 0.13 & 0.13 \\ 0.13 & 0.01 & 0.01 \\ 0.13 & 0.01 & 0.01 \end{bmatrix}$$

Anomalous Triangles
$$\begin{bmatrix} 0.01 & 0.09 & 0.09 \\ 0.09 & 0.01 & 0.09 \\ 0.09 & 0.09 & 0.01 \end{bmatrix}$$



(a) (b) (c)

Figure 4: Detection of Triangle Anomalies, anomalies at times 5, 15, 25... (a) Triangle Probability Detector, (b) Netsimile Detector, (c) Deltacon Detector

To evaluate each statistic used in dTrend, we will be running the anomaly detection process with each statistic in isolation on synthetic data with anomalies that statistic should detect. We will also run on a synthetic dataset with no anomalies to establish an acceptable false positive rate. The edge count statistic should create detections on edge count anomalies, both probability mass and probability distribution statistics should flag distribution anomalies, and the triangle statistic should be able to detect triangle anomalies. In addition, we will be applying Netsimile and Deltacon to the data as a basis for comparison. Netsimile extracts a host of node-level statistics (such as degree, neighbor degree, and local clustering coefficient), generates aggregates such as mean and standard deviation on the statistics, and then produces a distance score between two networks by taking the sum of the Canberra distances on each aggregate. We adapt this process as an anomaly detector by generating a Netsimile distance between each time step and its prior and flagging scores that exceed a p-value of 0.05. We do not add any additional detrending step to this algorithm. Deltacon also generates a distance score between pairs of network time steps but rather than using node or network statistics it generates an affinity matrix out of the random walk probabilities of all pairs of nodes and then reports the distance between two pairs of affinity matrices. When we implemented the detection threshold described in the Deltacon paper, we found it to be too conservative for most of our anomaly examples. Therefore we selected a more sensitive threshold in order to generate a number of detections more in line with the other algorithms.

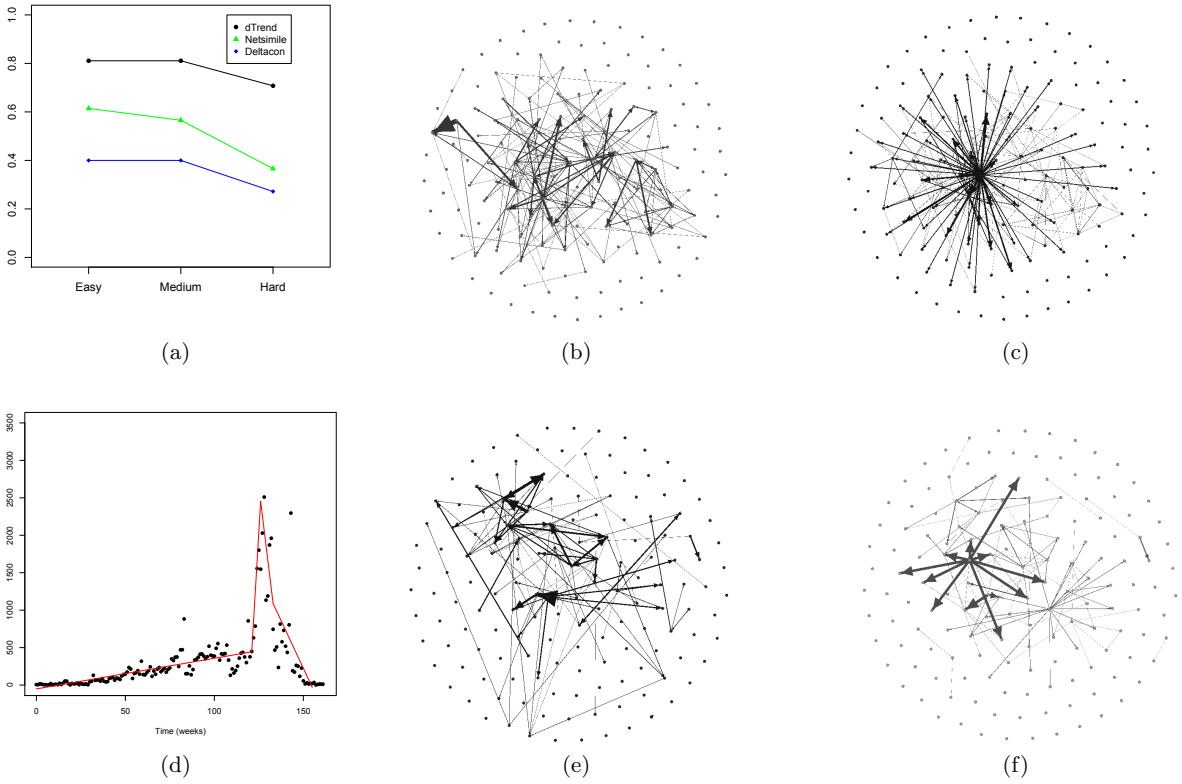
Figure 2 shows the results of applying each detector to synthetic data with no anomalies. Subfigure (a) shows most clearly the triangular baseline behavior of the data. Each detector produces an acceptable level of false positives in this case. Typically the a false positives appear in the sparser parts of the network stream, which means they may be a result of the higher variance in networks generated from fewer model samplings.

Figure 1 shows the results of adding edge anomalies into the synthetic data. Subfigure (a) demonstrates the ability of the edge count dTrend detector to accurately flag each anomaly. Netsimile and Deltacon, on the other hand, tend to miss anomalies in areas where the the underlying trend is at its peak. Netsimile in particular down weights anomalies with a larger underlying trend due to the use of Canberra distance: the distance between statistics x_1 and x_2 is $\frac{|x_1-x_2|}{x_1+x_2}$, which means that for larger values of x_1 and x_2 the difference between an anomalous and normal point (the anomaly signal strength) must be greater to create a detection. This explains the inverse relationship of the anomaly signals and the underlying trend in subfigure (b). Deltacon struggles for a different reason: as a random walk measure, it is most affected by changes in the connectivity of the network. Increasing the number of edges in an already well-connected cluster will have no effect on their random walk distances, thus generating no signal. In sparser networks the additional edges from the edge anomalies can generate detections but this is due to a random extra edge forming a bridge being more likely in an already sparse network. In general Deltacon is only appropriate when detecting anomalies that cause a change in the connectivity or centralities of the network.

Figure 3 shows the same trials applied to data with distribution anomalies. Note that as these detectors are all delta values on network slices from their prior, they tend to create a detection both entering and exiting an anomaly. In this case the distribution anomalies are accurately found by all of the detectors.

Figure 4 shows trials with triangle anomalies. Here Netsimile performs almost as well as our own detector, only missing an anomaly in the sparsest portion of the network. This is largely due to the fact that the underlying edge count trend has little effect on the clustering coefficient of the networks, giving Netsimile a set of anomalies with a consistent signal strength. Deltacon, on the other hand, struggles to detect these anomalies. In the sparse network slices the noise overwhelms the anomaly signal, and this seems to affect the sensitivity in the denser portions of the network.

Although these synthetic datasets are useful for qualitatively examining the types of anomalies each method will detect, the synthetic anomalies generated are generally easy to detect and do not fully test the capabilities of the detectors. To this end, we applied the methods to a second synthetic data set where we introduced anomalies of varying "difficulty," where the amount of variation from the normal case determines the ease of detection. The data was generated in



(a) (b) (c)

(d) (e) (f)

Figure 5: (a) F1 Performance on data with anomalies of mixed type, with the difficulty of detection for anomalies varying from easy to hard. (b) Enron network at time 142. (c) Enron network at time 143, example of a dramatic anomaly. (d) Underlying message volume trend in Enron data and trend line from segmentation. (e) Enron network at time 82. (f) Enron network at time 83, example of less dramatic anomaly.

the same way as before, except that the class matrices have been altered slightly to provide for the varying difficulties. In addition, the baseline case has become a near-wedge: this allows for the easiest generation of anomalous triangle time steps without altering the number of edges in the time step. The matrices are as follows:

$$\text{Normal} \quad \begin{bmatrix} 0.001 & 0.11 & 0.11 \\ 0.11 & 0.001 & 0.02 \\ 0.11 & 0.02 & 0.001 \end{bmatrix}$$

$$\text{Easy Distribution} \quad \begin{bmatrix} 0.001 & 0.20 & 0.02 \\ 0.20 & 0.001 & 0.02 \\ 0.02 & 0.02 & 0.001 \end{bmatrix}$$

$$\text{Medium Distribution} \quad \begin{bmatrix} 0.001 & 0.19 & 0.03 \\ 0.19 & 0.001 & 0.02 \\ 0.03 & 0.02 & 0.001 \end{bmatrix}$$

$$\text{Hard Distribution} \quad \begin{bmatrix} 0.001 & 0.17 & 0.05 \\ 0.17 & 0.001 & 0.02 \\ 0.05 & 0.02 & 0.001 \end{bmatrix}$$

$$\text{Easy Triangles} \quad \begin{bmatrix} 0.001 & 0.08 & 0.08 \\ 0.08 & 0.001 & 0.08 \\ 0.08 & 0.08 & 0.001 \end{bmatrix}$$

$$\text{Medium Triangles} \quad \begin{bmatrix} 0.001 & 0.09 & 0.09 \\ 0.09 & 0.001 & 0.06 \\ 0.09 & 0.06 & 0.001 \end{bmatrix}$$

$$\text{Hard Triangles} \quad \begin{bmatrix} 0.001 & 0.10 & 0.10 \\ 0.10 & 0.001 & 0.04 \\ 0.10 & 0.04 & 0.001 \end{bmatrix}$$

Edge anomalies have the same class matrix as normal time steps but an additional 10 (easy), 5 (medium), or 1 (hard) resample of the model is performed. Figure 5 shows the results of these experiments in terms of F1 score on detected anomalies. dTrend clearly outperforms both Netsimile and Deltacon by a significant margin. Netsimile largely struggles to deal with the change in network density and so will report the anomalies in the sparser networks but not in the denser ones. Deltacon has a different problem in that it has difficulty detecting certain types of anomalies like triangle anomalies. dTrend, with its multiple test and data detrending, has the most success finding multiple anomaly types in changing networks.

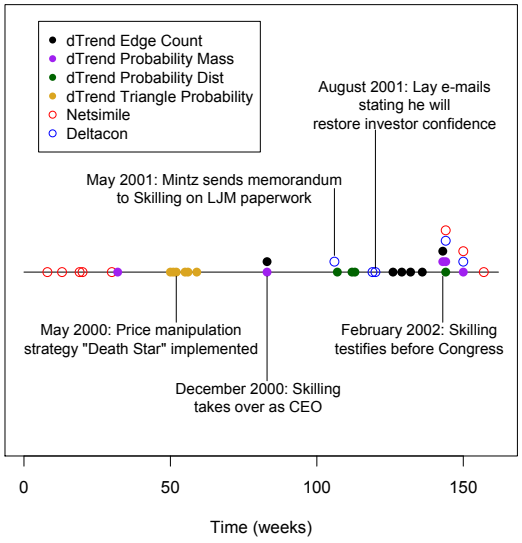


Figure 6: Detection of Anomalies in the Enron e-mail dataset. Filled circles are detections from dTrend components, open circles are other methods.

8. ENRON E-MAIL NETWORK

We will now demonstrate the utility of our detector in finding anomalies in a real-world scenario: the Enron e-mail network. The Enron e-mail network is the set of communications of the key players in the Enron scandal and downfall through the years of 1999-2002. The events of the scandal have a resulting effect on the structure of the network, so any anomalies found in the network stream should correspond to a real-world event. Due to a scarcity of messages on certain days we will be generating a network stream by aggregating the messages of each week. This aggregation causes a slight discrepancy in reported events compared to the analysis of Enron data in the original Deltacon paper as some events have been combined into a single detection. Figure 5 (d) shows the volume of messages in each time step through the company's lifespan. As you can see there is a slow upward trend followed by a dramatic rise and fall as the investigation of the company began. The red lines show the trend found through linear segmentation of the edge count stream.

Figure 6 shows the results of our detectors when applied to the set of e-mail data from the Enron corporation. Time step 143 represents the most significant event in the stream, Jeffrey Skilling's testimony before congress on February 6 2002, and is easily detected by any of our algorithms. The detected triangle anomalies at time steps 50-60 seem to correspond with Enron's price manipulation strategy known as "Death Star" which was put into action in May 2000. Step 83 coincides with the change in CEO from Kenneth Lay to Skilling. Other events not listed on the plot include Jordan Mintz sending a memorandum to Skilling for his sign-off on LJM paperwork (May 22, 2001, time step 106), Lay telling employees that Enron stock is an incredible bargain (September 26, 2001, time step 120), and Skilling approaching Lay about resigning (July 13, 2001, time step 113). Time step 150 seems to correspond to the virtual end of communication at the company.

Netsimile seems to have difficulty detecting important events in the Enron timeline. Although it can accurately find the time of the Congressional hearings, the other points flagged particularly early on do not correspond to any notable events and are probably false positives due to the artificial sensitivity of the algorithm in very sparse network slices.

Deltacon detects a greater range of events than Netsimile but still fails to detect several important events such as the price manipulation and first CEO transition. In particular the first CEO transition at time 83 is a fairly obvious anomaly in Figure 5 (d). Deltacon likely fails to detect events in these cases because it is sensitive only to certain types of anomalies, namely ones that cause a change in the network random walk distances. Anomalies that do not change the overall connectivity of the network will not be detected by Deltacon.

Figure 5 (b), (c), (e), and (f) show the network topography of the Enron data at key events. Subfigure (c) represents the most easily detected anomaly in the network stream, and it is clear why: this time step encounters a super-star structure which clearly does not exist in the previous time step. This not only adds a significant number of edges to the network but changes the distribution of edges in every measurable way. Subfigure (f) shows a very similar type of anomaly, with a large star structure appearing in the network. However, the main difference between this time step and 143 is the magnitude of the event: the star structure in 143 is vastly larger than the one in 83. A traditional anomaly detector would probably not flag 83 as an anomaly because the signal of anomalies like 143 are so much greater on unnormalized statistics like degree distribution. By detrending the data and using a normalized set of statistics we are able to generate accurate detections regardless of changes in the density of the network.

9. CONCLUSIONS AND FUTURE WORK

In this paper we have described the novel dynamic network anomaly detector dTrend and demonstrated its efficacy in both synthetic and real world data sets. dTrend adapts traditional network stream intuition about changing data trends to the realm of complex time-varying networks. In addition, dTrend is extremely flexible in application, as any desired network stream statistics can easily be incorporated into the algorithm. The joint dTrend detector is capable of accurately detecting and discriminating network stream anomalies of several different types even when the underlying properties of the network are shifting over time. We showed using synthetic data examples that dTrend outperforms other scaleable dynamic network anomaly detectors. When applied to the Enron e-mail network, dTrend is also able to flag anomalies at important events that the other algorithms do not detect. We have also posited a new underlying model for dynamic networks and suggested several modified statistics based on that model which are less sensitive to the degree of networks.

In the future we plan to expand the dTrend detector by providing more explanatory information about the reported anomalies. The current version of the detector can identify what kinds of anomalies have occurred in a time step, but do

not distinguish which nodes or communities are responsible for the signal or whether the anomaly is truly global. Details about the graph components affected by the anomaly may provide insight into the nature of the event responsible. In addition, pinpointing the exact times when the anomalous edges occur creates a more detailed narrative of the event and might be useful for diagnosing the anomaly. Both these additions will help not only identify the anomalies, but provide better explanations of their causes as well.

10. ACKNOWLEDGEMENTS

This research is supported by NSF under contract numbers IIS-1149789, CCF-0939370, and IIS-1219015. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of NSF or the U.S. Government.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

11. REFERENCES

[1] S. Moreno and J. Neville. Network Hypothesis Testing Using Mixed Kronecker Product Graph Models. In Proceedings of the 13th IEEE International Conference on Data Mining, 2013.

[2] R. Hanneman and M. Riddle. "Introduction to Social Network Methods." University of California, Riverside (2005)

[3] P. Yates. "An Inferential Framework for Biological Network Hypothesis Tests." Virginia Commonwealth University (2010).

[4] K. Faust and J. Skvoretz. "Comparing Networks Across Space and Time, Size and Species." Sociological Methodology, 2002.

[5] B. Desmarais, S. Cranmer. "Micro-level interpretation of exponential random graph models with application to estuary networks." Policy Studies Journal, Volume 40, Issue 3, pages 402-434, August 2012

[6] S. Hanneke, W. Fu, E.P. Xing. "Discrete Temporal Models of Social Networks." EJS Vol. 4 (2010) 585-605

[7] R. Madhavan, B.R. Koka and J.E. Prescott. "Networks in Transition: How Industry Events (Re)Shape Interfirm Relationships." Strategic Management Journal Vol. 19, 439-459 (1998).

[8] Snijders, Tom AB. "Models for longitudinal network data." Models and methods in social network analysis 1 (2005): 215-247.

[9] D. Banks, K. Carley. "Models for network evolution." Journal of Mathematical Sociology, 21:1-2, 173-196 (1996).

[10] I. McCulloh. "Detecting Changes in a Dynamic Social Network." (2009) CMU-ISR-09-104.

[11] T. Snijders, S. Borgatti. "Non-Parametric Standard Errors and Tests for Network Statistics." CONNECTIONS 22(2): 161-70 l'1999 INSNA

[12] T. Snijders, G. van de Bunt, C. Steglich. "Introduction to stochastic actor-based models for network dynamics." Social Networks 32 (2010) 44-60.

[13] Tartakovsky, Alexander, Hongjoong Kim, Boris Rozovskii, and Rudolf Blažek. "A novel approach to detection of 'denial-of-service' attacks via adaptive sequential and batch-sequential change-point detection methods." IEEE Transactions on Signal Processing, VOL. 54, NO. 9, September 2006.

[14] Siris, Vasilios A., and Fotini Papagalou. "Application of anomaly detection algorithms for detecting SYN flooding attacks." Computer Communications 29 (2006) 1433-1442.

[15] McCulloh, Ian A., and Kathleen M. Carley. "Social network change detection." No. CMU-ISR-08-116. CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2008.

[16] Basseville, Michéle, and Igor Nikiforov. "Detection of Abrupt Changes: Theory and Application." Inform. System Sci. Ser., Prentice-Hall, Englewood Cliffgs, NJ (1993).

[17] Koutra, Danai, Joshua Vogelstein, Christos Faloutsos. "DELTACON: A Principled Massive-Graph Similarity Function." SDM 2013, Austin, Texas, May 2013.

[18] Berlingerio, M., D. Koutra, T., and C. Faloutsos. "Network Similarity via Multiple Social Theories." arXiv preprint arXiv:1209.2684 (2012). In Proceedings of the 5th IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, pp. 1439-1440. ACM, 2013.

[19] Kawahara, Yoshinobu, and Masashi Sugiyama. "Change-Point Detection in Time-Series Data by Direct Density-Ratio Estimation." SDM. Vol. 9. 2009.

[20] Chen, Jie, and A. K. Gupta. "On Change Point Detection and Estimation." Commun. Statist. -Simula., 30(3), 665-697 (2001).

[21] Xu, Limei, et al. "Quantifying signals with power-law correlations: A comparative study of detrended fluctuation analysis and detrended moving average techniques." Physical Review E 71.5 (2005): 051101.

[22] Keough, Eamonn, et al. "An online algorithm for segmenting time series." IEEE Internation Conference on Data Mining 2001, pp. 289-296.

[23] Nowicki, K. and T. A. B. Snijders. "Estimation and prediction for stochastic block structures." Journal of the American Statistical Association, 96:1077-1087, 2001.

[24] Snijders, Tom AB, et al. "New specifications for exponential random graph models." Sociological methodology 36.1 (2006): 99-153.

Change-point detection in temporal networks using hierarchical random graphs

Leto Peel
Department of Computer Science
University of Colorado
Boulder, Colorado
leto.peel@colorado.edu

Aaron Clauset
Department of Computer Science
University of Colorado
Boulder, Colorado
aaron.clauset@colorado.edu

ABSTRACT

Networks are an important tool for describing and quantifying data on interactions among objects or people, e.g., online social networks, offline friendship networks, and object-user interaction networks, among others. When interactions are dynamic, their evolving pattern can be represented as a sequence of networks each giving the interactions among a common set of vertices at consecutive points in time. An important task in analyzing such evolving networks, and for predicting their future evolution, is *change-point detection*, in which we identify moments in time across which the large-scale pattern of interactions changes fundamentally. Here, we formalize the network change point detection problem within a probabilistic framework and introduce a method that can reliably solve it in data. This method combines a generalized hierarchical random graph model with a generalized likelihood ratio test to quantitatively determine if, when, and precisely how a change point has occurred. Using synthetic data with known structure, we characterize the difficulty of detecting change points of different types, e.g., groups merging, splitting, forming or fragmenting, and show that this method is more accurate than several alternatives. Applied to a high-resolution evolving social proximity network, this method identifies a sequence of change points that align with known external shocks to these network.

Categories and Subject Descriptors

G.3 [**Probability and Statistics**]: Time Series Analysis; G.2.2 [**Graph Theory**]: Network problems; E.1 [**Data Structures**]: Graphs and networks

General Terms

Algorithms, Theory

Keywords

Dynamic networks, change-point detection, generative models, model comparison, anomaly detection

ODD^2*'14,* August 24th, 2014, New York, NY, USA.

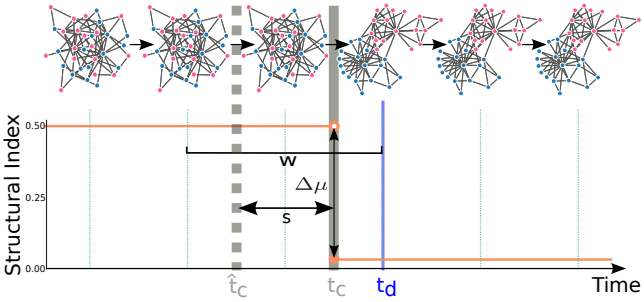


Figure 1: Schematic of a network change point. A sequence of networks in which vertices divide into two groups at time t_c, represented by a change in an abstract structural index $\Delta\mu$. To detect this change point, we estimate the time of change $\widehat{t}_c$ within a sliding window of the last w networks, and call t_d the time of detection in which $\widehat{t}_c$ is found to be statistically significant. The detection delay, s, is the difference between the change point t_c and the estimated change point $\widehat{t}_c$.

1. INTRODUCTION

Relational variables among objects or people are a common form of data, and networks provide a general framework through which to quantify and analyze their patterns. For example, online social interactions, offline friendships, and object-user interactions may all be represented as networks. In many cases, these relations are dynamic, and their evolution over time may be represented as a sequence of networks, each giving the interactions among a common set of vertices at consecutive points in time. Knowledge discovery in such networks thus depends on understanding both the large-scale structure of a network and how that structure changes over time.

A key task in this effort is *change-point detection*, in which we both identify moments in time across which the large-scale pattern of interactions changes fundamentally and quantify what kind and how large a change occurred (Fig. 1). Identifying the timing and shape of such change points divides a network's evolution into contiguous periods of relative structural stability, allowing us to subsequently analyze each period independently, while also providing hints about the underlying processes shaping the data.

For instance, in social networks, change points may be the result of normal periodic behavior, as in the weekly transi-

tion from weekdays to weekends. In other cases, change points may result from the collective anticipation of or response to external events or system "shocks". Detecting such changes in social networks could provide a better understanding of patterns of social life and could potentially be used for early detection of illegal or malicious activities or as indicators of social stress caused by, e.g, natural or man-made disasters.

Traditionally in network change-point detection, the data is a univariate or multivariate continuous valued time series [5]. In this case change-point detection involves estimating "norms", for instance using a univariate or multivariate Gaussian distribution, and then detecting when a significant deviation from this norm has occurred. To develop rigorous change-point detection methods for networks we require a network-based notion of "normal" so that we may detect when these interactions have changed significantly from that norm. To characterize what kind and how large a change occurred, we prefer interpretable models of network structure, so that changes in parameter values have direct meaning with respect to the network's large-scale structure.

Much like change-point detection methods for scalar or vector-valued time series [5], our approach for change-point detection in networks has three components:

1. select a parametric family of probability distributions appropriate for the data, and a sliding window size w;
2. infer two versions of the model, one representing a change of parameters at a particular point in time within the window, and the one representing the null hypothesis of no change over the entire window; and,
3. conduct a statistical hypothesis test to choose which model, change or no-change, is the better fit.

Here we introduce a change-point detection technique based on generative models of networks, which define parametric probability distributions over graphs. Our particular choice of model is the generalized hierarchical random graph (GHRG), which compactly models nested community structure at all scales in a network. The framework, however, is entirely general, and the GHRG could, in principle, be replaced with another generative network model, e.g., the stochastic block model [15, 23], hierarchical random graph [8], or Kronecker product graph model [18]. To compare the change versus no-change models, we use a generalized likelihood ratio test, with a user-defined parameter specifying a target false-positive rate.

We then show that this approach quantitatively and accurately determines *if* the network has changed, *when* precisely the change occurred, and *how* the network has changed. Specifically, we present a taxonomy of different types and sizes of network change points and a quantitative characterization of the difficulty of detecting them, in synthetic network data with known change points. We then test the method on two real, high-resolution evolving social networks of physical and digital interactions, showing that it accurately recovers the timing of known significant external events.

2. RELATED WORK

Change-point detection in networks is a form of anomaly detection, and within this area, there are two main thrusts: anomalous subgraphs, which focuses on detecting subgraphs that are structured differently from the rest of the network, and temporal anomalies, which focuses on detecting changes in the network's structure over time. Most efforts have focused on detecting anomalous subgraphs, which differ from the temporal anomalies we consider in that they focus on changes across a sequence of networks rather than variations within a single network.

2.1 Detecting anomalous subgraphs

Early work in detecting anomalies in graphs examined networks containing vertices of different types or labels, and sought to identify unusual patterns of connectivity between the vertex types [22, 10]. Later work used principle eigenvectors of a graph's modularity matrix to detect subgraphs that may have been generated by a different process to the rest of the network [20].

Similar to traditional outlier detection, some efforts have focused on detecting individual anomalous vertices [12] by comparing vertex attributes to the distribution of attributes within a network community. Finally, rather than detecting anomalous vertices, another approach aims to detect anomalous connections by assessing the observed interactions via link prediction models [16].

2.2 Detecting temporal network anomalies

Detecting temporal anomalies and detecting change points are closely related tasks. On the one hand, an anomaly is an outlier relative to the distribution from which the rest of the data are drawn, while a change point is a shift from one moment to the next in the overall distribution of the data. Thus, a succession of change points away from and then back to some particular distribution can be viewed as as a temporal anomaly.

Along these lines, a method called GraphScope [27], based on minimum description length, was used to detect when and how communities change in evolving networks. At each time step, this method performs a local search for alternative models that are close to the current community decomposition. As a result, non-smoothness in the underlying community structure score function [13] can prevent the detection of real changes. Other approaches use spectral techniques [14, 4], e.g., using the eigenvectors of the correlation matrix of vertex in-degrees to detect global and local anomalies in the network or by constructing "eigen-behaviors" from the eigenvalues of the correlation matrix of local network measures for each vertex in the network. Although quantitative in nature, these approaches do not provide statistically rigorous determinations of whether a change has occurred and therefore the detection itself is a qualitative decision. We bridge this gap by utilising statistical hypothesis tests to provide a quantitative decision about whether or not a change has occured.

Statistical hypothesis tests provide a reliable mechanism for estimating the probability of seeing a change as large or larger than the one we observe, relative to a fully-defined model of "normal" behavior. When combined with a specified error rate, this approach provides principled estimates for detecting changes. Ref. [21] recently constructed a statistical hypothesis test for estimating the likelihood that a particular network was generated by a Kronecker product graph model fitted to other data, with good results. However, hypothesis tests based on local [25] and global [19] network measures are more common. The former approach

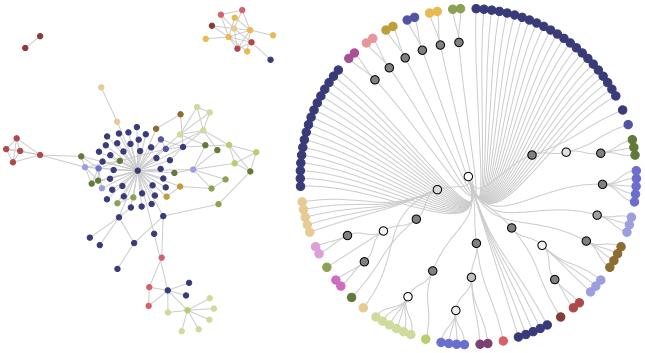


Figure 2: A snapshot of a social network and its corresponding GHRG dendrogram. In the dendrogram, leaves are vertices in the social network and the tree gives their nested group structure.

identified local subregions of excessive activity in networks. In practice, this method appears to have low sensitivity, primarily identifying as anomalies events in which vertices suddenly began or ceased all activity. The latter approach employs classic statistical change-point detection methods (specifically cumulative sum [24] and exponentially weighted moving average [26]) to detect overall structural changes. By compressing an entire network into a small number of scalars, however, these methods necessarily discard information that may be crucial to detecting changes. Furthermore, the use of these summaries can make it hard to interpret how the large-scale structure of the network has changed. In contrast, our approach captures more of the network structural information and is less sensitive to small scale changes (e.g., addition or removal of a single edge). Additionally the interpretability of our model enables us to analyse how the network has changed from before to after the change point.

3. DEFINING A PROBABILITY DISTRIBUTION OVER NETWORKS

Under a probabilistic approach to change-point detection, we must choose a parametric distribution over networks. Here, we introduce the generalized hierarchical random graph (GHRG) model, which has several features that make it attractive for change-point detection and generalizes the popular hierarchical random graph (HRG) model [8]. First, this model naturally captures both assortative and disassortative community structure patterns, models community structure at all scales in the network, and provides accurate and interpretable fits to social, biological and ecological networks. Second, our generalization relaxes the requirement that the dendrogram is a full binary tree, thereby eliminating the HRG's non-identifiability and improving the model's interpretability by quantifying how a network's structure varies across a change point. Third, we use a Bayesian model of connection probabilities that quantifies our uncertainty about the network's underlying generative model.

While we make the particular choice to use the GHRG, our framework is entirely general, and the GHRG could, in principle, be replaced with another generative network model, e.g., the stochastic block model [15, 23], or Kronecker product graph model [18]. These models, however, require the number of groups to be specified as a parameter to the model. As a result, this introduces the additional problems of deciding whether or not the number of groups changes from snapshot to snapshot and how to map groups and at one time step to groups at the next. The GHRG naturally addresses these issues, as it adjusts the levels in the hierarchy from the observed structure in the network.

The GHRG models a network $G = \{V, E\}$ composed of vertices V and edges $E \subseteq \{V \times V\}$. The model decomposes the N vertices into a series of nested groups, whose relationships are represented by a dendrogram T. Vertices in G are leaves of T, and the probability that two vertices u and v connect in G is given by a parameter p_r located at their lowest common ancestor in T. In the classic HRG model [8], each tree node in T has exactly two subtrees, and p_r gives the density of connections between the vertices in the left and right subtrees. As a result, distinct combinations of dendrograms and probabilities produce identical distributions over networks, producing a non-identifiable model. In the GHRG, we eliminate this possibility by allowing tree nodes to have any number of children. Figure 2 illustrates the GHRG applied to a network of email communications.

Given tree T and set of connection probabilities $\{p_r\}$, the GHRG defines a distribution over networks and a likelihood function

$$p(G \,|\, T, \{p_r\}) = \prod_r p_r^{E_r} (1 - p_r)^{N_r - E_r} \quad , \tag{1}$$

where E_r is the number of edges between vertices with common ancestor r and N_r is the total number of possible edges between vertices with common ancestor r:

$$N_r = \sum_{c_i < c_j \in C_r} |c_i||c_j| \quad . \tag{2}$$

By relaxing the binary-tree requirement, the GHRG produces a spectrum of hierarchical structure (Fig. 3). On one end of this spectrum, T contains only a single internal tree node—the root—and every pair of vertices connects with the same probability p_r associated with it, equivalent to the popular Erdős-Rényi random graph model. As more tree nodes and their parameters are added to T, the number of levels of hierarchy increases, allowing the model to capture more varied large-scale patterns. In the limit, T is a full binary tree, and we recover the classic HRG model [8].

The structure of the inferred GHRG readily lends itself to interpretation, since the more internal splits the model recovers, the greater the level of substructure in the network. If the tree has no splits, then this indicates that edges appear to be iid random variables with equal probability, i.e., there is no internal structure in this network. This enables us to not only use this model to detect when changes occur, but also to understand how the network structure at changes at each change point.

4. LEARNING THE MODEL

Fitting the GHRG model to a network requires a search over all trees on N leaves and the corresponding link probability sets $\{p_r\}$, which we accomplish using Bayesian posterior inference and techniques from phylogenetic tree reconstruction.

Tree structures are not amenable to classic convex optimization techniques, and instead must be searched explicitly. But, searching the space of all non-binary trees

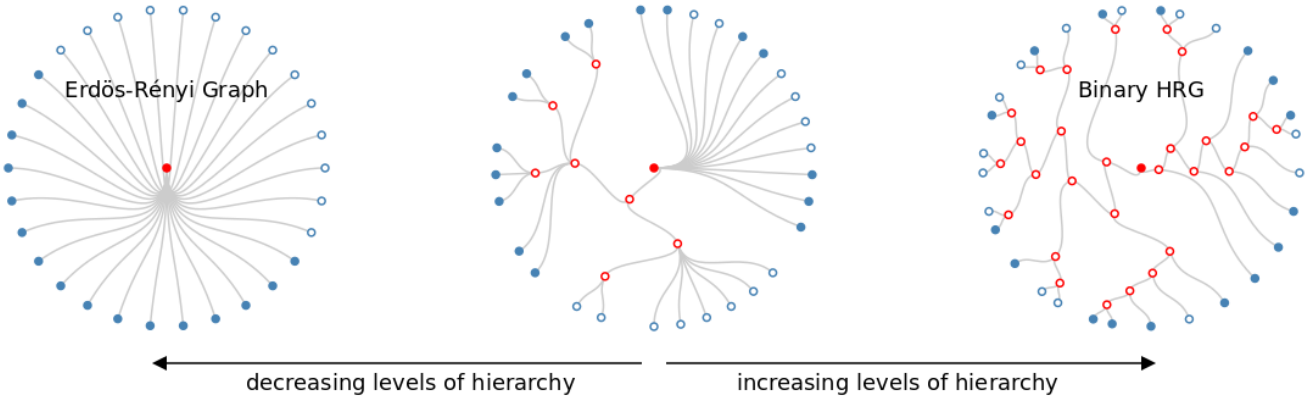


Figure 3: A spectrum of large-scale structure, corresponding to different amounts of hierarchy in the GHRG, ranging from a simple random graph to a complete hierarchical organization.

is costly. Phylogenetic tree reconstruction faces a similar problem, which is commonly solved by taking a majority "consensus" of a set of sampled binary trees [6]. This consensus procedure selects the set of bipartitions on the leaves that occur in a majority of the sampled binary trees, and each such set denotes a unique non-binary tree containing exactly those divisions. For instance, if every sampled tree is identical, then each is identical to the consensus tree, while if every sampled tree is a distinct set of bipartitions, the consensus tree has a single internal node to which all of the leaf nodes are connected.. Thus, we estimate T in the GHRG by using a Markov Chain Monte Carlo (MCMC) procedure to first sample binary HRG tree structures with probability proportional to their likelihood of generating the observed network. From this set of sampled binary dendrograms, we derive their non-binary majority consensus tree (an approach previously outlined in Ref. [8], but not used to produce a probabilistic model) and assign link probabilities $\{p_r\}$ to the remaining tree nodes.

One approach would be to choose their values via maximum likelihood, setting each

$$\hat{p}_r = E_r / N_r \quad . \tag{3}$$

However, this choice provides little room for uncertainty and is likely to increase our error rate in change-point detection. For instance, consider the case where exactly zero connections $E_r = 0$, or equivalently all connections, $E_r = N_r$, are observed for a particular branch r. Under maximum likelihood, we would set $p_r = 0$ or 1. If a subsequent network has, or lacks, even a single edge whose common ancestor is r, then $E_r > 0$ or $E_r < 1$, and the likelihood given by Eq. (1) drops to 0, an unhelpful outcome.

We mitigate this behavior by assuming Bayesian priors on the p_r values [1]. Now, instead of setting p_r to a point value, we model each p_r as a distribution, which quantifies our uncertainty in its value and prevents its expected value from becoming 0 or 1. For convenience, we employ a Beta distribution with hyperparameters α and β, which act like pseudocounts for the presence or absence of connections respectively. We set these to $\alpha = \beta = 1$, which corresponds to a uniform distribution over the parameters p_r. Because the Beta distribution is conjugate with the Binomial distribution, we may integrate out each of the p_r parameters analytically

$$\begin{aligned} p(G \,|\, T, \alpha, \beta) &= \int p(G \,|\, T, \{p_r\}) p(\{p_r\} \,|\, \alpha, \beta) \, \mathrm{d}\{p_r\} \qquad (4) \\ &= \prod_r \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(E_r+\alpha)\Gamma(N_r - E_r + \beta)}{\Gamma(N_r+\alpha+\beta)} \quad . \end{aligned}$$

Given this formulation, we may update the posterior distribution over the parameter p_r given a sequence of observed networks $\{G_t\}$ by updating the hyperparameters as

$$\tilde{\alpha}_r = \alpha + \sum_{\{G_t\}} E_r^{G_t} \tag{5}$$

$$\tilde{\beta}_r = \beta + \sum_{\{G_t\}} N_r - E_r^{G_t} \quad . \tag{6}$$

Thus, we obtain the posterior hyperparameters from the sum of the prior pseudocounts of edges and the empirically observed edge counts (number of present and absent connections). This Bayesian approach produces an implicit regularization. As the number of observations N_r increases, the posterior distribution becomes increasingly peaked, reflecting a decrease in parameter uncertainty. In the GHRG model, parameters closer to the root of T represent larger-scale structures in G and govern the likelihood of more edges. These parameters are thus estimated with greater certainty, while the distribution over parameters far from the root, which represent small-scale structures, have greater variance, which prevents over-fitting to small-scale structural variations.

5. DETECTING CHANGE POINTS IN NETWORKS

The final piece of change-point detection in evolving networks requires a method to determine whether and when the parameters of our current model of "normal" connectivity have changed. Here, we develop a generalized likelihood ratio test over a sliding window of fixed length w to detect if any changes have occurred with respect to a fitted GHRG

model. A change is detected if the log-likelihood ratio versus a null or "no change" model exceeds a threshold determined by a desired false positive rate.

5.1 Generalized likelihood ratio test

Using a likelihood ratio test, we compare the observed data's likelihood under two different models: a null hypothesis model H_0, in which no change occurs, and an alternative hypothesis model H_1, in which a change occurs at some particular time t_c. Since we use network snapshots at regular intervals, we assume t_c occurs between some pair of snapshots, which we indicate using a 0.5 offset (Fig. 1).

We restrict our consideration of change points to those within a sliding window of w networks, the last of which is at the "current" time τ. For the no-change model, we say that all networks within the window were drawn from a model with parameters $\psi^{(\emptyset)}$. For the change model, we let $\psi^{(0)}$ denote the model parameters for networks up to t_c within our window and $\psi^{(1)}$ the parameters for networks after t_c, but still within the window. Rewriting the change and no-change hypotheses in terms of such a shift in a parametric distribution over graphs at t_c, we have

$$\begin{aligned} H_0 &: \psi^{(\emptyset)} = \psi^{(\emptyset)} \qquad \text{(no change)} \\ H_1 &: \psi^{(0)} \neq \psi^{(1)} \qquad \text{(change)} \ . \end{aligned}$$

These change and no-change hypotheses are composite, meaning that the parameters ψ can take on any permissible value, so long as they meet the definitions of H_0 and H_1 given above. To test such hypotheses, we must use a generalized likelihood ratio (GLR) test.

The GLR's test statistic is the log-likelihood ratio, which compares the likelihood of the w networks under a model with a change point at some time t_c versus the no-change model, which we denote Λ_{t_c},

$$\Lambda_{t_c} = \log \frac{\sup_{\widehat{\psi}^{(0)},\widehat{\psi}^{(1)} \in \Omega} L\left(\widehat{\psi}^{(0)}, \widehat{\psi}^{(1)}\right)}{\sup_{\widehat{\psi}^{(\emptyset)} \in \Omega} L\left(\widehat{\psi}^{(\emptyset)}\right)} \ , \tag{7}$$

where Ω is the set of permissible parameter values. In our Bayesian formulation, rather than set the parameters that maximize the likelihood, we set the prior parameters to their posterior values in accordance with Eqs. (5) and (6) and integrate over the parameters [2].[1]

Finally, the time at which the change occurs t_c is itself an unknown value, and must be estimated. We make the conservative choice, choosing $\widehat{t}_c$ as the time point between a pair of consecutive networks that maximizes our test statistic Λ across the window. Letting g_τ for a given window of w networks ending at τ be that largest value

$$g_\tau = \max_{\tau - w + 1 < \widehat{t}_c < \tau} \Lambda_{\widehat{t}_c} \ . \tag{8}$$

Using $\widetilde{\psi} = \{\tilde{\alpha}_r, \tilde{\beta}_r\}$ to denote the set of posterior hyperparameters, the GHRG's generalized likelihood ratio is

$$\Lambda_{\widehat{t}_c} = \log \frac{\prod_{t=\tau-w+1}^{\widehat{t}_c - 0.5} p(G_t \,|\, T_\tau, \widetilde{\psi}^{(0)}_{\widehat{t}_c}) \prod_{t=\widehat{t}_c+0.5}^{\tau} p(G_t \,|\, T_\tau, \widetilde{\psi}^{(1)}_{\widehat{t}_c})}{\prod_{t=\tau-w+1}^{\tau} p(G_t \,|\, T_\tau, \widetilde{\psi}^{(\emptyset)})} \ ,$$

[1]This ratio of marginal likelihoods is called a *posterior Bayes factor* and may be interpreted as a likelihood ratio for two simple hypotheses [3].

Algorithm 1: Parametric bootstrap sampling g_τ from the null model distribution

Input: $\mathcal{G} = \{G\}_{\tau-w+1}^{\tau}$
$\{g_\tau\}_{\text{null}} = \emptyset$.
$\text{GHRG}(T_\tau, \widetilde{\psi}_\tau) = \text{fitGHRG}(\mathcal{G})$.
for $i = 1$ *to* 1000 **do**
  sample w graphs, $\{G_t\}_{t=1}^{w} \sim \text{GHRG}(T_\tau, \widetilde{\psi}_\tau)$.
  for $\tilde{t}_c = 1$ *to* w **do**
    calculate $\widetilde{\psi}^{\emptyset}, \widetilde{\psi}^{0}, \widetilde{\psi}^{1}$ according to Eq. (5) and Eq. (6).
    $\Lambda_{\tilde{t}_c} = \sum_{t=1}^{\tilde{t}_c - 1} \log p\left(G_t | T_\tau, \widetilde{\psi}^{(0)}_{\tilde{t}_c}\right)$
      $+ \sum_{t=\tilde{t}_c}^{w} \log p\left(G_t | T_\tau, \widetilde{\psi}^{(1)}_{\tilde{t}_c}\right)$
      $- \sum_{t=1}^{w} \log p\left(G_t | T_\tau, \widetilde{\psi}^{\emptyset}\right)$
  end
  $g_\tau^{(i)} = \max_{\tilde{t}_c} \Lambda_{\tilde{t}_c}$.
  $\{g_\tau\}_{\text{null}} = \{g_\tau\}_{\text{null}} + g_\tau^{(i)}$.
end

where $\widetilde{\psi}^{(\emptyset)}$ is the set of posterior hyperparameters pertaining to the no-change hypothesis (no change point anywhere in the window of w networks), while $\widetilde{\psi}^{(0)}_{\widehat{t}_c}$ and $\widetilde{\psi}^{(1)}_{\widehat{t}_c}$ are the hyperparameters for the networks up to and then following the point $\widehat{t}_c$, respectively. These hyperparameters are estimated from the data using Eqs. (5) and (6).

5.2 Parametric Bootstrapping

The choice of threshold h, which g_τ must exceed for a detection to occur, sets the method's resulting false positive rate and the distribution of g_τ under the null model. Recent results on model comparison for statistical models of networks, and specifically the stochastic block model, of which the GHRG is a particularly useful special case, suggest that for technical reasons the null distribution can deviate substantially from the χ^2 distribution [28]. To avoid a misspecified test, we estimate the null distribution numerically, via Monte Carlo samples from a parametric bootstrap distribution [11] defined by the GHRG for the no-change model. In this way, we estimate the null distribution exactly, rather than via a possibly misspecified approximation.

For each network we sample from the no-change GHRG model, we calculate g_τ from Eq. (8) to obtain its distribution under the hypothesis of no change (see Algorithm 1). Using the sampled distribution, the threshold h may then be chosen so that $p(g_\tau > h) = p_{\text{fp}}$ is the desired false positive rate. In practice we do this by calculating a p-value for the test case by counting the proportion of likelihood ratios in our null distribution that are higher than our test statistic, g_τ:

$$p\text{-value} = \frac{|\{g_\tau\}_{\text{null}} > g_\tau|}{|\{g_\tau\}_{\text{null}}|} \tag{9}$$

Thus, if we find a p-value below the chosen threshold, we say a change is detected, and when the no-change model is correct, we are incorrect no more than p_{fp} of the time.

6. RESULTS

We now demonstrate our change-point detection method on synthetic and real networks. Each of the synthetic net-

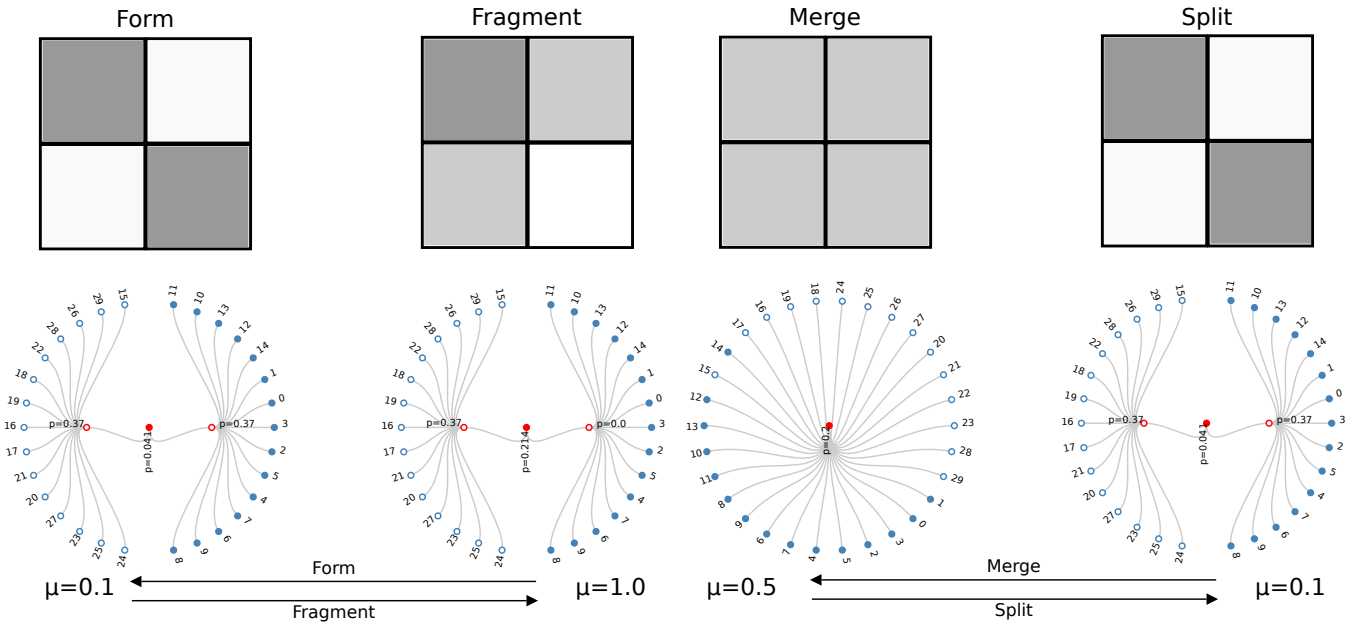


Figure 4: Taxonomy of change points: formation versus fragmentation and merging versus splitting. In each case, we show both the block structure of the adjacency matrix for two groups and the corresponding GHRG model. For our experiments, we switch from one structure to the other by changing the structural index μ. See text for more detail.

works contain a known change point of varying magnitude, which can be used to quantitatively evaluate the false positive and false negative error rates. The real network provides a case study on which we perform a more qualitative comparison against known external events.

6.1 Change points in synthetic networks

Before applying our method to empirical data with unknown structure and unknown change points, we first systematically characterize the detectability of different types of network change points under controlled circumstances on synthetic data, generated using our GHRG model, with known structure and changes.

The following change-point types constitute difficult but realistic tests that cover a broad variety of empirically observed large-scale changes to network structure. Specifically, we choose a modest-sized network of $N = 30$ and a sparse and constant expected number of connections (marginal link probability of 0.2). Furthermore, we define four general types of change points: *splitting*, when one large community divides in two; *merging*, when two communities combine (the time-reversal of splitting); *formation*, when one of two groups of vertices add edges to make a community; and *fragmentation*, when one of two groups loses all its edges (the time-reversal of formation). Figure 4 illustrates these types of change points.

To provide a single parameter that controls the switching between these distinct states, we define a single structural index μ as

$$\mu = \frac{p_{\text{out}}}{p_{\text{in}} + p_{\text{out}}} \quad . \tag{10}$$

For the merge/split change points, we choose the merged state to be $\mu = 0.5$, which produces a single community in which every edge occurs with the same probability $p_{\text{in}} = p_{\text{out}}$. In the split state, the network is comprised of two distinct communities. If $\mu < 0.5$ then the communities are assortative ($p_{\text{in}} > p_{\text{out}}$, and vertices prefer to connect within the community) and disassortative if $\mu > 0.5$ ($p_{\text{in}} < p_{\text{out}}$, and vertices prefer to connect across communities).

For formation/fragmentation change points, we use the same two-community model, but now fix the link probability within one community and use μ to describe relationship between the p_{in} and p_{out} of the second community.

We now summarize these change points with respect to μ:

merge	$\mu \neq 0.5 \rightarrow \mu = 0.5$
split	$\mu = 0.5 \rightarrow \mu \neq 0.5$
fragment	$\mu < 1 \quad \rightarrow \mu = 1$
form	$\mu = 1 \quad \rightarrow \mu < 1$.

In all of the tests a window size of $w = 4$ and a false-positive rate of 0.05 was used. For comparison we use three simple methods that use the same probabilistic framework as our method but instead parameterize the distribution over graphs using a univariate Gaussian to model a network summary statistic (mean degree, mean geodesic distance, mean local clustering coefficient).

In Figure 5 we compare the false positive and false negative error rates among all four methods. On false positives, all methods are close to 0.05, which matches the desired false alarm rate. However, the false negative rates differ widely, with the simple methods performing terribly in nearly every case, even when the size of the change is large. In contrast, our method performs well across all four tests, except when the size of the change is very small, e.g., when $\Delta\mu \approx 0$,

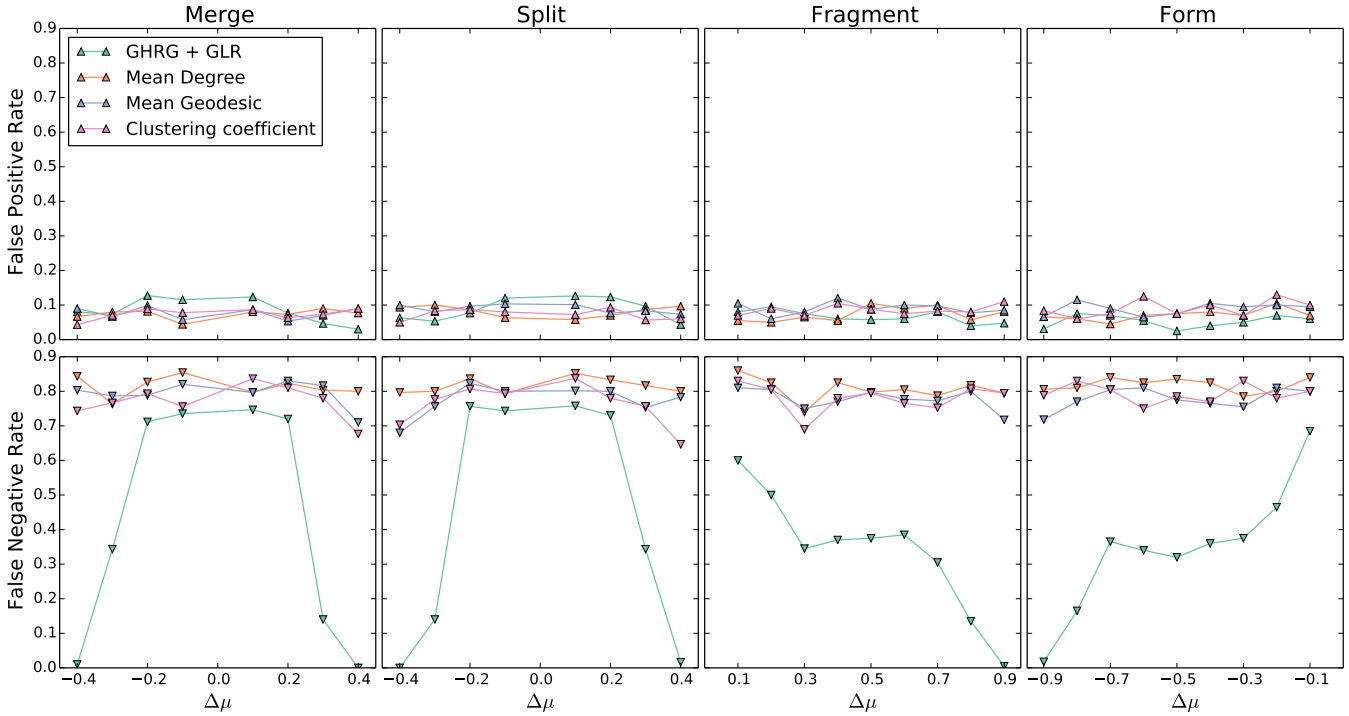


Figure 5: False positive (*top*) and false negative (*bottom*) error rates for our method and the simple methods on the four different change types for different magnitudes of change($\Delta\mu$).

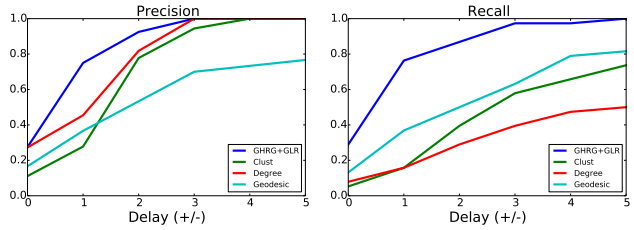


Figure 6: Precision and recall of our method and the three baseline methods at identifying known external events as a function of delay (s).

which represents the hardest cases, where small-sample fluctuations obscure much of the actual change.

6.2 Change points in real networks

We now apply these approaches[2] to detect changes in a high-resolution evolving network, the Hypertext 2009 proximity network[3] [17]. We provide a quantitative and qualitative evaluation of the approach by comparing the detected change points with known external events, i.e., the planned conference schedule.

The dataset is an evolving network of human social interactions collected over 2.5 days at the ACM Hypertext 2009 conference. Vertices represent the 113 conference attendees who volunteered to wear RFID badges to monitor face-to-face proximity. We constructed a sequence of networks in which an edge denotes face-to-face physical proximity to one of the 113 subjects at some point during a 15 minute time span.

For each detection method, the GHRG and the three simple methods, we used a window size $w = 4$, the same as in the synthetic experiments. The results for each method are shown in Figure 7 (top), which includes the identified change points for each method, the conference program and the time series of network measures for the simple methods. In the lower half of Figure 7 the inferred GHRG dendrograms for the first day of the conference are shown. We quantify the performance in terms of precision and recall as a function of the detection delay, s, between estimated change points, $\{\widehat{t}_c\}$, and scheduled events, $\{t_c\}$, i.e.,

$$\text{Precision(s)} = \frac{\sum_i \delta\left(\inf_j |\widehat{t}_c^{(i)} - t_c^{(j)}| \leq s\right)}{n_c}, \qquad (11)$$

$$\text{Recall(s)} = \frac{\sum_j \delta\left(\inf_i |\widehat{t}_c^{(i)} - t_c^{(j)}| \leq s\right)}{n_a}, \qquad (12)$$

where $\delta(\cdot)$ is a delta function which is 1 if $(\cdot)$ is true and 0 otherwise, and n_c and n_a are the number of estimated change points and actual events respectively. The precision is then the proportion of estimated change points that occur within a given delay, s, of a known event. Similarly, recall is the proportion of known events that occur within a delay s of an estimated change point. Figure 6 shows the precision and recall for our method and the three simple methods. We

[2]Code for our change-point detection method is available at `http://gdriv.es/letopeel/code.html`

[3]`www.sociopatterns.org/datasets/hypertext-2009-dynamic-contact-network/`

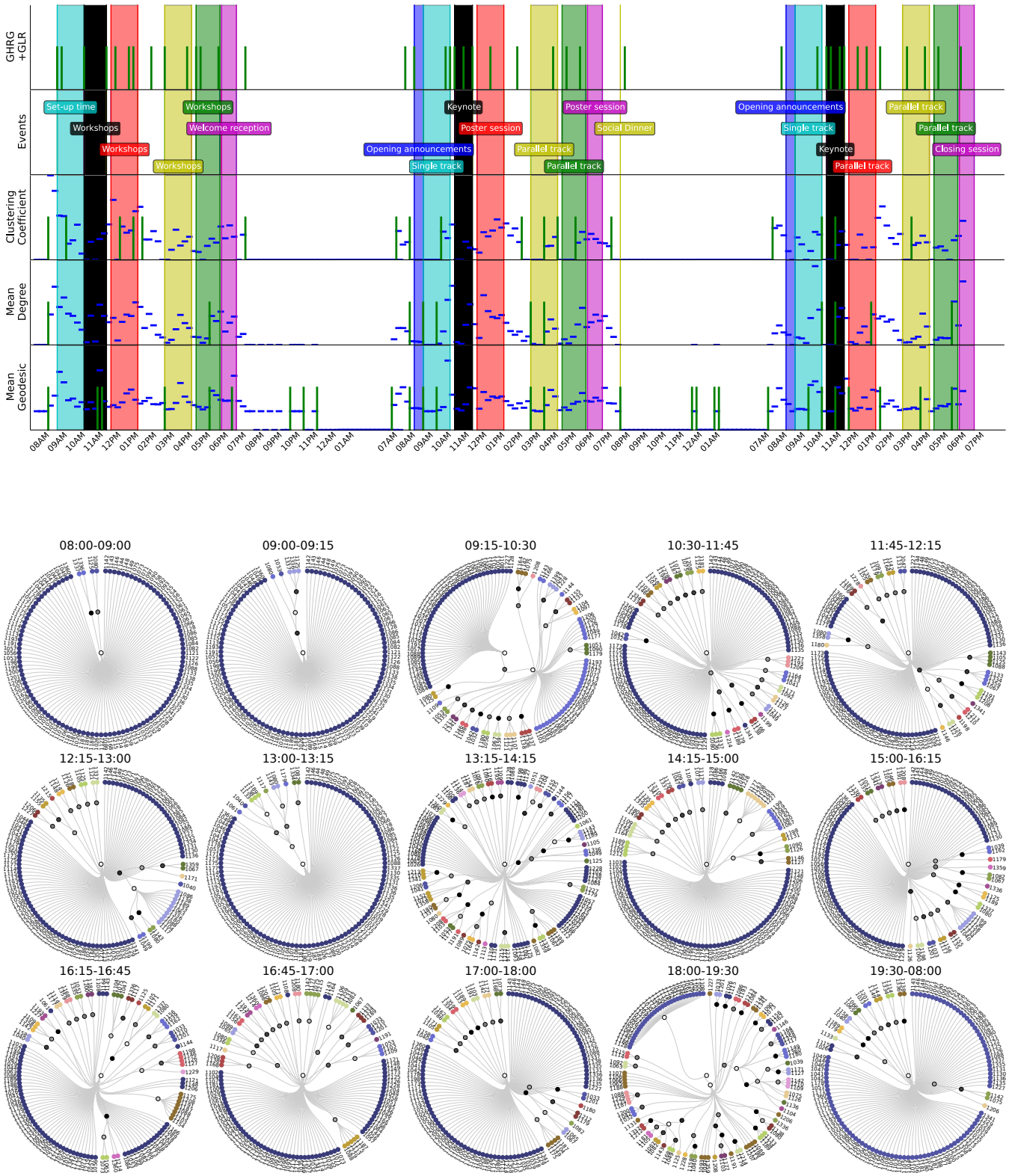


Figure 7: Change points detected in the Hypertext 2009 conference proximity network (*top*). Each row shows, from top to bottom, the change points detected (green vertical bars) using our method, the associated known events, and the change points detected with the three simple methods along with the values of the relevant summary statistic (blue horizontal bars). The inferred GHRG dendrograms for the first day (*bottom*) show how the network structure evolves during the day.

see that our approach outperforms all three simple methods for both metrics.

We can also see this qualitatively in Figure 7. Examining the simple methods' results, we observe that they exhibit low sensitivity relative to the known conference schedule[4] . It is common at academic conferences for the schedule timings to deviate somewhat from the planned program and this could account for discrepencies regarding exactly when change points occur. However, even accounting for this, the simple methods do pick out several real change points, including the start of the workshops on day one and the breaks and sessions between 10am and 2pm on day two. Additionally there are inconsistencies between the approaches such as the beginning of lunch on day one is not detected by mean degree and mean geodesic and the begining of lunch on day 3 is not detected by the clustering coefficient.

On the other hand, our GHRG method identifies nearly all of the conference session transitions along with some additional changes. For instance, during the workshop and parallel sessions that could be indications of conference participants switching sessions betweeen individual talks.

Examining the dendrogram structure for the first day of the conference (Figure 7 (bottom)), we can see *how* the network evolves during the course of the day. We see that before 9:15am there is very little structure in the hierarchy accounting for the fact that very few people are interacting before the start of the conference. There is a dramatic increase in the number of levels of the hierarchy indicating the formation of many communities as attendees mingle during the setup period before the start of the first talk. During the workshop sessions the majority of conference attendees would be facing the same direction (towards the speaker) and therefore not facing each other resulting in a sparser network with less structure (10:30-13:15).

The scheduled lunch break was between 13:30 and 15:00, but with our approach we detect two phases 13:15-14:15 and 14:15-15:00. We first of all note that we detect the lunch break early, which could be either because the sessions finished early or because we estimate the change point early by one network. Looking at the dendrogram structure we see a lot of levels in the hierarchy indicating tight communities of attendees that took lunch together. In the second detected lunch phase we see that a few stable communities remain, but at least half of the attendees do not participate in communities. This change point is suggests that this is the time that a significant proportion of attendees have finished lunch.

7. DISCUSSION

When analyzing a sequence of time evolving networks, a central goal is to understand how the network's structure has changed over time, and how it might change in the future. Change-point detection provides a principled approach to this problem, by decomposing the networks sequence into subsequences of distinct but relatively stable structural patterns (Fig. 1). Formalizing this problem within a probabilistic framework, we developed a statistically principled method, based on generative models and hypothesis tests, that can detect if, when and how such change points occur in the large-scale patterns of interactions. Under our framework, change points occur when the shape of an estimated probability distribution over networks changes significantly.

Not all such change points are equally easy to detect, however. Using synthetic data with known structure and known change points, we found that we could only reliably detect changes that were of a large enough magnitude. Given that our aim is to detect abrupt changes in the overall structure, then this behavior is actually desireable. While it is theoretically possible to tune our model to be more sensitive to smaller changes, this would be traded off against a greater false positive rate due to a greater sensitivity to changes in individual links in the network.

That being said, change-point methods based on network measures like the mean degree, clustering coefficient, or mean geodesic path length performed poorly, yielding high false negative rates even for large structural changes (Fig. 5). This poor performance is likely the result of network measures discarding much of the specific information that generative models utilize. Applied to a high-resolution evolving social networks, our method provided very good results. Our precision and recall at recovering the timing of many known external "shock" events out-performed the network-measure methods (Fig. 7). We were also able to qualitatively assess how the network was evolving over time by examining the dendrogram structure of the GHRG.

A notable assumption of our method is that each observed network is an iid sample from some underlying distribution. Real evolving networks likely violate this assumption, as a result of underlying non-stationary or periodic dynamics. When each network spans more time than the natural time scale of network dynamics [7], the iid assumption may be reasonable, and our empirical results support this notion. An interesting direction for future work would relax the iid assumption perhaps using Markov models.

Although the generalized hierarchical random graph model yielded good results, in principle, any generative model could be used in its place, e.g., the stochastic block model [15, 23, 1] or the Kronecker product graph model [18]. Similarly, the recent work in graph hypothesis testing [21] could potentially be adapted to the change-point detection problem. Two key features of the generalized hierarchical random graph model (GHRG) for change-point detection, however, are its interpretability and the way it naturally adapts its dendrogram structure to fit the network, adding or removing levels in the hierarchy, as the network evolves.

Our synthetic experiments allowed us to explore the range of change points that we are able to detect. An interesting direction for future work would be to investigate how the detectability of change points relates to the detectability theshold in community detection [9].

Our strong results on synthetic data give us confidence in our ability to detect change points in real-world data. By combining the correlation of estimated change points with known external events with the interpretability of the GHRG, this approach to change-point detection promises to have broad application in testing hypotheses about the underlying dynamics driving network evolution.

8. ACKNOWLEDGEMENTS

We thank Dan Larremore for helpful conversations, and acknowledge support from Grant #FA9550-12-1-0432 from the U.S. Air Force Office of Scientific Research (AFOSR) and the Defense Advanced Research Projects Agency (DARPA).

[4] www.ht2009.org/program.php

9. REFERENCES

[1] C. Aicher, A. Z. Jacobs, and A. Clauset. Adapting the stochastic block model to edge-weighted networks. In *ICML Workshop on Structured Learning*, 2013. arxiv:1305.5782.

[2] M. Aitkin. Posterior bayes factors (with discussion). *Journal of the Royal Statistical Society*, 53:111–142, 1991.

[3] M. Aitkin. The calibration of p-values, posterior Bayes factors and the AIC from the posterior distribution of the likelihood. *Statistics and Computing*, 7(4):253–261, 1997.

[4] L. Akoglu and C. Faloutsos. Event detection in time series of mobile communication graphs. In *27th Army Science Conference*, Orlando, FL, USA, 2010.

[5] M. Basseville and I. V. Nikiforov. *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[6] D. Bryant. A classification of consensus methods for phylogenetics. In *Bioconsensus*, volume 61 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 163–183. Amer. Math. Soc., Providence, RI, 2003.

[7] A. Clauset and N. Eagle. Persistence and periodicity in a dynamic proximity network. In *DIMACS Workshop on Computational Methods for Dynamic Interaction Networks*, 2007. arxiv:1211.7343.

[8] A. Clauset, C. Moore, and M. E. J. Newman. Structural inference of hierarchies in networks. In E. Airoldi, D. M. Blei, S. E. Fienberg, A. Goldenberg, E. P. Xing, and A. X. Zheng, editors, *Statistical Network Analysis: Models, Issues, and New Directions*, volume 4503 of *Lecture Notes in Computer Science*, pages 1–13. Springer Berlin Heidelberg, 2007.

[9] A. Decelle, F. Krzakala, C. Moore, and L. Zdeborová. Inference and phase transitions in the detection of modules in sparse networks. *Phys. Rev. Lett.*, 107:065701, Aug 2011.

[10] W. Eberle and L. Holder. Discovering structural anomalies in graph-based data. In *Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, ICDMW '07, pages 393–398, Washington, DC, USA, 2007. IEEE Computer Society.

[11] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*. Chapman and Hall, 1993.

[12] J. Gao, F. Liang, W. Fan, C. Wang, Y. Sun, and J. Han. On community outliers and their efficient detection in information networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 813–822, New York, NY, USA, 2010. ACM.

[13] B. H. Good, Y.-A. de Montjoye, and A. Clauset. Performance of modularity maximization in practical contexts. *Phys. Rev. E*, 81:046106, 2010.

[14] S. Hirose, K. Yamanishi, T. Nakata, and R. Fujimaki. Network anomaly detection based on eigen equation compression. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 1185–1194, New York, NY, USA, 2009. ACM.

[15] P. W. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983.

[16] Z. Huang and D. Zeng. A link prediction approach to anomalous email detection. In *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on*, volume 2, pages 1131–1136, Oct 2006.

[17] L. Isella, J. Stehlé, A. Barrat, C. Cattuto, J. Pinton, and W. Van den Broeck. What's in a crowd? analysis of face-to-face behavioral networks. *Journal of Theoretical Biology*, 271(1):166–180, 2011.

[18] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos. Realistic, mathematically tractable graph generation and evolution, using Kronecker multiplication. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD'05, pages 133–145, Berlin, Heidelberg, 2005. Springer-Verlag.

[19] I. McCulloh and K. M. Carley. Detecting change in longitudinal social networks. *Journal of Social Structure*, 12, 2011.

[20] B. Miller, N. Bliss, and P. Wolfe. Subgraph detection using eigenvector l1 norms. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1633–1641. 2010.

[21] S. Moreno and J. Neville. Network hypothesis testing using mixed Kronecker product graph models. In *13th IEEE International Conference on Data Mining*, 2013.

[22] C. C. Noble and D. J. Cook. Graph-based anomaly detection. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 631–636, New York, NY, USA, 2003. ACM.

[23] K. Nowicki and T. A. B. Snijders. Estimation and Prediction for Stochastic Blockstructures. *Journal of the American Statistical Association*, 96(455), 2001.

[24] E. S. Page. Cumulative sum charts. *Technometrics*, 3(1):1–9, 1961.

[25] C. E. Priebe, J. M. Conroy, D. J. Marchette, and Y. Park. Scan statistics on Enron graphs. *Comput. Math. Organ. Theory*, 11(3):229–247, Oct. 2005.

[26] S. W. Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 1(3):239–250, 1959.

[27] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. Graphscope: Parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 687–696, New York, NY, USA, 2007. ACM.

[28] X. Yan, J. E. Jensen, F. Krzakala, C. Moore, C. R. Shalizi, L. Zdeborová, P. Zhang, and Y. Zhu. Model selection for degree-corrected block models. *arXiv*, 1207.3994, 2012.

Real time contextual collective anomaly detection over multiple data streams

Yexi Jiang, Chunqiu Zeng
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
{yjian004, czeng001}@cs.fiu.edu

Jian Xu
School of Computer Science Technology and Engineering
Nanjing University of Science and Technology
Nanjing, China
dolphin.xu@njust.edu.cn

Tao Li
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
taoli@cs.fiu.edu

ABSTRACT

Anomaly detection has always been a critical and challenging problem in many application areas such as industry, healthcare, environment and finance. This problem becomes more difficult in the Big Data era as the data scale increases dramatically and the type of anomalies gets more complicated. In time sensitive applications like real time monitoring, data are often fed in streams and anomalies are required to be identified online across multiple streams with a short time delay. The new data characteristics and analysis requirements make existing solutions no longer suitable.

In this paper, we propose a framework to discover a new type of anomaly called contextual collective anomaly over a collection of data streams in real time. A primary advantage of this solution is that it can be seamlessly integrated with real time monitoring systems to timely and accurately identify the anomalies. Also, the proposed framework is designed in a way with a low computational intensity, and is able to handle large scale data streams. To demonstrate the effectiveness and efficiency of our framework, we empirically validate it on two real world applications.

1. INTRODUCTION

Anomaly detection is one of the most important tasks in data-intensive applications such as the healthcare monitoring [22], stock analysis [26], disaster management [32], system anomaly detection [24], and manufacture RFID management [4], etc. In the Big Data era, the aforementioned applications often require real-time processing. However, existing data processing infrastructures are designed based on inherent non-stream programing paradigm such as MapReduce [11], Bulk Synchronous Parallel (BSP) [30], and their variations. To reduce the processing delay, these applications have gradually migrated to stream processing engines [27, 10, 1, 9]. As the infrastructures have been changed, anomalies in these applications are required to be identified online across multiple data streams. The new data characteristics and analysis requirements make existing anomaly detection solutions no longer suitable.

ODD'14, August 24 - 27 2014, New York, NY, USA.

http://dx.doi.org/10.1145/2656269.2656271.

1.1 A Motivating Example

EXAMPLE 1. *Figure 1 illustrates the scenario of monitoring a 6-node computer cluster, where the x-axis denotes the time and the y-axis denotes the CPU utilization. The cluster has been monitored during time* $[0, t_6]$. *At time* t_2, *a computing task has been submitted to the cluster and the cluster finishes this task at time* t_4. *As shown, two nodes (marked in dashed line) behave differently from the majority during some specific time periods. Node* ① *has a high CPU utilization during* $[t_1, t_2]$ *and a low CPU utilization during* $[t_3, t_4]$ *while node* ② *has a medium CPU utilization all the time. These two nodes with their associated abnormal periods are regarded as anomalies. Besides these two obvious anomalies, there are a slight delay on node* ③ *due to the network delay and a transient fluctuation on node* ④ *due to some random factors. However, they are normal phenomena in distributed systems and are not regarded as anomalies.*

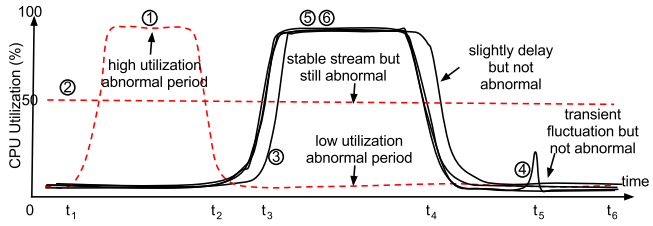


Figure 1: CPU utilization of a computing cluster

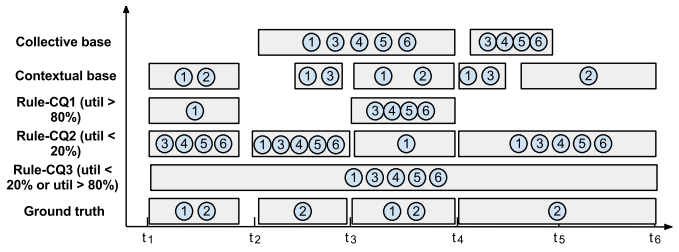


Figure 2: Identified anomalies in Example 1 (The box lists the IDs of abnormal streams during specified time period)

A quick solution for stream based anomaly detection is to leverage the techniques of *complex event processing (CEP)* [23, 2] by expressing the anomalies detection rules with corresponding continuous query statements. This rule-based detection method can be applied to the scenarios where the anomaly can be clearly defined. Besides using CEP, several stream based anomaly detection algorithms have also been proposed. They either focus on identifying contextual anomaly over a collection of stable streams [7] or collective anomaly from one stream [3, 25]. These existing methods are useful in many applications but they still cannot identify certain types of anomalies. A simple example of such scenario is illustrated in Example 1.

Figure 2 plots the ground truth as well as all the anomalies identified by existing methods including the CEP query with three different rules (Rule-CQ1, 2, and 3), the collective based anomaly detection [6], and contextual based anomaly detection [8].

To detect the anomalies via CEP query, the idea is to capture the events when the CPU utilizations of nodes are too high or too low. An example query following the syntax of [2] can be written as follows:

```
PATTERN SEQ(Observation o[])
WHERE avg(o[].cpu) oper threshold
  (AND|OR avg(o[].cpu) oper threshold)*
WITHIN {length of sliding window}
```

where the selection condition in `WHERE` clause is the conjunction of one or more boolean expressions, *oper* is one of {>, <, <>, ==}, and `threshold` can be replaced by any valid expression. However, CEP queries are unable to correctly identify the anomalies in Figure 1 no matter how the selection conditions are specified. For instance, setting the condition as `avg(o[].cpu) > {threshold}` would miss the anomalies during $[t_3, t_4]$ (Rule-CQ1); setting the condition as `avg(o[].cpu) < {threshold}` would miss the anomalies during $[t_1, t_2]$ (Rule-CQ2); and combining the two above expressions with OR still does not work (Rule-CQ3). Besides deciding the selection condition, how to rule out the situations of slight delays and transient fluctuations, and how to set the length of the sliding windows are all difficult problems when writing the continuous queries. The main reason is that the continuous query statement is not suitable to capture the contextual information where the "normal" behaviors are also dynamic (the utilizations of normal nodes also change over time in Figure 1).

Compared with CEP based methods, contextual anomaly detection methods (such as [14, 17]) achieve a better accuracy as they utilize the contextual information of all the streams. However, one limitation of contextual based methods is that they do not leverage the temporal information of streams and are not suitable for anomaly detection in dynamic environments. Therefore, these methods would wrongly identify the slightly delayed and fluctuated nodes as anomalies.

For the given example, collective anomaly detection methods do not work well neither. This is because these methods would identify the anomaly of each stream based on its normal behaviors. Once the current behavior of a stream is different from its normal behaviors (identified based on historical data), it is considered as abnormal. In the example, when the cluster works on the task during $[t_3, t_4]$, all the working nodes would be identified as abnormal due to the sudden burst.

1.2 Contributions

In this paper, we propose an efficient solution to identify this special type of anomaly in Example 1, named *contextual collective anomaly*. Contextual collective anomalies bear the characteristics of both contextual anomalies and collective anomalies. This type of anomaly is common in many applications such as system monitoring, environmental monitoring, and healthcare monitoring, where data come from distributed but homogeneous data sources. We will formally define this type of anomaly in Section 2.

Besides proposing an algorithm to discover the contextual collective anomalies over a collection of data streams, we also consider the scale-out ability of our solution and develop a distributed streaming processing framework for contextual collective anomaly detection. More concretely, our contributions can be described as follows:

- We provide the definition of contextual collective anomaly and propose an incremental algorithm to discover the contextual collective anomalies in real time. The proposed algorithm combines the contextual as well as the historical information to effectively identify the anomalies.
- We propose a flexible three-stage framework to discover such anomalies from multiple data streams. This framework is designed to be distributed and can be used to handle large scale data by scaling out the computing resources. Moreover, each component in the framework is pluggable and can be replaced if a better solution is proposed in the future.
- We empirically demonstrate the effectiveness and efficiency of our solution through the real world scenario experiments.

The rest of the paper is organized as follows. Section 2 gives a definition of contextual collective anomaly and then presents the problem statement. Section 3 provides an overview of our proposed anomaly detection framework. We introduce the three-stage anomaly detection algorithm in detail in Section 4. Section 5 presents the result of experimental evaluation. The related works are discussed in Section 6. Finally, we conclude in Section 7.

2. PROBLEM STATEMENT

In this section, we first give the notations and definitions that are relevant to the anomaly detection problem. Then, we formally define the problem based on the given notations and definitions.

DEFINITION 1. DATA STREAM. *A data stream S_i is an ordered infinite sequence of data instances $\{s_{i1}, s_{i2}, s_{i3}, ...\}$. Each data instance s_{it} is the observation of data stream S_i at timestamp t.*

The data instances s_{it} in S_i can have any number of dimensions, depending on the concrete applications. For the remaining of this paper, the terms "data instance" and "observation" would be used interchangeably. *To make the notation uncluttered, we use s_i in places where the absence of timestamp does not cause the ambiguity.*

Definition 2. Stream Collection. *A stream collection $\mathcal{S} = \{S_1, S_2, ..., S_n\}$ is a collection of data streams. The number of streams $|\mathcal{S}| = n$.*

The input of our anomaly detection framework is a *stream collection*. For instance, in example 1, the input stream collection is $\mathcal{S} = \{①, ②, ③, ④, ⑤, ⑥\}$

Definition 3. Snapshot. *A snapshot is a set of key-value pairs $S^{(t)} = \{S_i : s_{it} | S_i \in \mathcal{S}\}$, denoting the set of the observations $\{s_{1t}, s_{2t}, \cdots, s_{|S|t}\}$ of the data streams in stream collection $\mathcal{S}$ at time t.*

A *snapshot* captures the configuration of each data stream in the stream collection for a certain moment. Taking Figure 1 for example, the snapshot at time t_5 is $S^{(t_5)} = \{① : 0\%, ② : 50\%, ③ : 0\%, ④ : 20\%, ⑤ : 0\%, ⑥ : 0\%\}$. For simplicity, we use $S(i)$ to denote the ith dimension of the observations in a certain snapshot. Note that all the observations in a snapshot have the same dimension.

Definition 4. Contextual Collective Anomaly. *A contextual collective stream anomaly is denoted as a tuple $< S_i, [t_b, t_e], N >$, where S_i denote a data stream from the collection of data streams $\mathcal{S}$, $[t_b, t_e]$ is the associated time period when S_i is observed to constantly deviate from the majority streams in S, and N indicates the severity of the anomaly.*

In Example 1, 3 contextual collective anomalies can be found in total. During time period $[t_1, t_2]$, node ① behaves constantly different from the other nodes, so there is an anomaly $<①, [t_1, t_2], N_1>$. The other two contextual collective anomalies, $<①, [t_3, t_4], N_2>$ and $<②, [0, t_6], N_3>$, can also be found with the same reason.

In Definition 4, the severity of deviation is measured in a given metric space $\mathcal{M}$ with a distance function $f : s_{it} \times s_{ku} \to \mathbb{R}$. For simplicity, we use Euclidean distance as an example throughout this paper.

Problem Definition. The anomaly detection problem in our paper can be described below: Given a stream collection $\mathcal{S} = \{S_1, S_2, ..., S_m\}$, identify the source of the contextual collective anomalies S_i, the associated time period $[t_s, t_e]$, as well as a quantification about the confidence of the detection p. Moreover, the detection has to be conducted on data streams that look-back is not allowed and the anomalies are able to be identified in real time.

3. FRAMEWORK OVERVIEW

In this section, we briefly describe how the aforementioned problem is addressed and then introduce the proposed distributed real time anomaly detection framework from a high level perspective.

As previously mentioned, *this paper focuses on discovering contextual collective anomalies over a collection of data streams obtained from a homogeneous distributed environment*. An example of the homogeneous distributed environment is the system with load balance, which is widely used at the backend by the popular web sites like Google, Facebook, and Amazon, etc.

It is known that, in such a kind of environment, the components should behave similar to each other. Therefore, the snapshots (the current observations) of these streams should be close to each other at any time. Naturally, we need to identify the anomalies by investigating both contextual information (the information of the current snapshot) and collective information (the historical information).

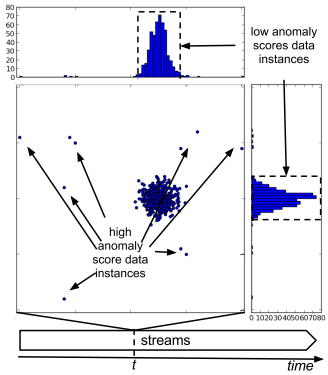


Figure 3: The snapshot at a certain timestamp

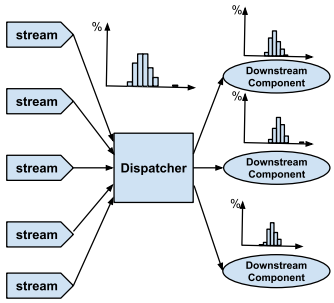


Figure 4: Random partition of data instance

The anomaly detection is conducted in three stages: the *dispatching stage*, the *scoring stage*, and the *alert stage*. The functionality of the three stages are briefly described as follows:

- *Dispatching stage:* This stage uses *dispatchers* to receive the observations from external data sources and then shuffle the observations to different downstream processing components.
- *Scoring stage:* This stage quantifies the candidate anomalies using *snapshot scorer* and then *stream scorer*.

  The *snapshot scorer* leverages contextual information to quantify the confidence of anomaly for each data instance at a given *snapshot*. Taking Figure 3 for example, it shows the data distribution by taking the snapshot of the 2-dimensional data instances of 500 streams at timestamp t. As shown, most of the data instances are close to each other and located in a dense area. These data instances are not likely to be identified as anomalies as their instance anomaly scores are small. On the contrary, a small portion of the data instances (those points that are far away from the dense region) have larger instance anomaly scores and are more likely to be abnormal.

  A data instance with a high anomaly score does not indisputably indicate its corresponding stream to be a real anomaly. This is because the transient fluctuation and phase shift are common in real world distributed environment. To mitigate such effects, the *stream scorer* is designed to handle the problem. In particular, the *stream scorer* combines the information obtained from the *instance scorer* and the historical information of each stream to quantify the anomaly confidence of each stream.
- *Alert stage:* The alert stage contains the *alter trigger*. The alert triggers leverage the unsupervised learning methods to identify and report the outliers.

The advantage of our framework is reflected by the ease of integration, the flexibility, and the algorithm independence.

Firstly, any external data sources can be easily fed to the framework for anomaly detection. Moreover, the components in every stage can be scaled-out to increase the processing capability if necessary. The number of components in each stage can be easily customized according to the data scale of concrete applications. Furthermore, the algorithms in each stage can be replaced and upgraded with better alternatives and the replacement would not interfere with other stages.

4. METHODOLOGY

4.1 Data Receiving and Dispatching

The dispatching stage is an auxiliary stage in our framework. When the data scale (i.e., the number of streams) is too large for a single computing component to process, the dispatcher would shuffle the received observations to downstream computing components in the scoring stage (as shown in Figure 4). By leveraging random shuffling algorithm like Fisher-Yates shuffle [12], dispatching can be conducted in constant time per observation. After dispatching, each downstream component would conduct scoring independently on a sampled stream observations with identical distribution.

Ideally, the observations coming from homogeneous data sources have very similar measurable value (e.g. workload of each server in a load balanced system), but random factors can easily cause the variations of the actual observations. Therefore in fact, an observation $\boldsymbol{s_i} \in \mathbb{R}^d$ is viewed as the ideal case value $\boldsymbol{s_{ideal}}$ with additive Gaussian noise so that $\boldsymbol{s_i} = \boldsymbol{s_{ideal}} + \boldsymbol{\epsilon}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. For those data sources in abnormal conditions, their observations are generated according to a different but unknown distributed. It is not difficult to know that, given enough observations, the mean and covariance can be easily estimated locally via maximum likelihood, i.e. $\hat{\boldsymbol{\mu}}_{ML} = \frac{\sum_i \boldsymbol{s_i}}{n}$ and $\widehat{cov}(S(x), S(y))_{ML} = \frac{1}{n-1}\sum_{i=1}^{n}(s_i(x) - \bar{S}(x))(s_i(y) - \bar{S}(y))$, where $\bar{S}(x)$ and $\bar{S}(y)$ respectively denote the sample mean of dimension x and y.

4.2 Snapshot Anomaly Quantification

Quantifying the anomaly in a snapshot is the first task in the *scoring stage* and we leverage *snapshot scorer* in this step. This score measures the amount of deviation of the specified observation s_{it} to the center of all the observations in a snapshot $S^{(t)}$ at timestamp t.

To quantify the seriousness of snapshot anomaly, we propose a simple yet efficient method. The basic idea is that the anomaly score of an observation is quantified as the amount of uncertainty it brings to the snapshot $S^{(t)}$. As the observations in a snapshot follows the normal distribution, it is suitable to use the increase of entropy to measure the anomaly of an observation. To quantify the anomaly score, two types of variance are needed: the *variance* and the *leave-one-out variance*, where the leave-one-out variance is the variance of the distribution when one specific data instance is not counted.

A naive algorithm to quantify the anomaly scores requires quadratic time ($O(dn + dn^2)$). By reusing the intermediate results, we propose an improved algorithm with time complexity linear to the number of streams. The pseudo code of the proposed algorithm is shown in Algorithm 1. As illustrated, matrix M is used to store the distances between each

Algorithm 1 Snapshot Anomaly Quantification

1. **INPUT**: Snapshot $S^{(t)} = (s_1, \cdots, s_n)$, where $s_i \in \mathbb{R}^d$.
2. **OUTPUT**: Snapshot anomaly scores $N \in \mathbb{R}^n$.
3. Create a $d \times n$ matrix $M = (s_1, \cdots, s_n)$
4. Conduct 0-1 normalization on rows of M.
5. $\boldsymbol{x} \leftarrow \mathbf{0}^d$ and $N = \mathbf{0}^d$
6. $\boldsymbol{m} \leftarrow (\mathbb{E}(S(1)), \cdots, \mathbb{E}(S(d))^T$
7. $M \leftarrow (s_{1t} - \boldsymbol{m}, s_{2t} - \boldsymbol{m}, \cdots, s_{dt} - \boldsymbol{m})$
8. **for** $j \leftarrow 0$ to d **do**
9. $\quad \boldsymbol{x}_j = ||j$th column of $M||_2^2$
10. **end for**
11. **for all** $s_i \in S^{(t)}$ **do**
12. $\quad$ Calculate N_i according to Equation (1).
13. **end for**
14. Conduct 0-1 normalization on N.
15. **return** N

dimension of the observations to the corresponding mean. Making use of M, the *leave-one-out variance* can be quickly calculated as $\sigma_{ik}^2 = \frac{n\sigma_k^2 - M_{i,k}^2}{n-1}$, where σ_k^2 denotes the variance of dimension k and σ_{ik}^2 denotes the leave-one-out variance of dimension k by excluding s_i. As the entropy of normal distribution is $H = \frac{1}{2}\ln(2\pi e\sigma^2)$, the increase of entropy for observation s_i at dimension k can be calculated as

$$d_k = H'_k - H_k = \ln\frac{\sigma_{ik}^2}{\sigma_k^2} = \ln\frac{(\boldsymbol{x}_k - M_{ik}^2)/(n-1)}{\boldsymbol{x}_j/n}, \quad (1)$$

where H'_k and H_k respectively denote the entropy of the snapshot distribution if s_i is not counted or counted.

Summing up all dimensions, the snapshot anomaly score of s_{it} is $N_{it} = \sum_k d_k$. Note that the computation implicitly ignores the correlation between dimensions. The reason is that if an observation is an outlier, the correlation effect would only deviate it further from other observations.

4.3 Stream Anomaly Quantification

As a stream is continuously evolving and its observations only reflect the transient behavior, snapshot anomaly score alone would result in a lot of false-positives due to the transient fluctuation and slight phase shift. To mitigate such situations, it is critical to quantify the stream anomaly by incorporating the historical information of the stream.

An intuitive way to solve this problem is to calculate the stream anomaly score from the recent historical instances stored in a sliding window. However, this solution has two obvious limitations: (1) It is hard to decide the window length. A long sliding window would miss the real anomaly while a short sliding window cannot rule out the false-positives. (2) It ignores the impact of observations that are not in the sliding window. The observation that is just popped out from the sliding window would immediately and totally lose its impact to the stream.

To well balance the history and the current observation, we use *stream anomaly score* N_i to quantify how significant a stream S_i behaves differently from the majority of the streams. To quantify N_i, we exploit the exponential decay function to control the influence depreciation. Supposing Δt is the time gap between two adjacent observations, *the influence of an observation* s_{it} *at timestamp* $t_{x+k} = t_x + k\Delta t$ can be expressed as $N_{it_x}(t_{x+k}) = N_{it_x}(t_x + k\Delta t) =$

$N_{it_x}e^{-\lambda kt}, (\lambda > 0)$, where λ is a parameter to control the decay speed. In the experiment evaluation, we will discuss how this parameter affects the anomaly detection results.

To make the notation uncluttered, we use t_{-i} to denote the timestamp that is $i\Delta t$ ahead of current timestamp t, i.e. $t_{-i} = t - i\Delta t$. Summing up the influences of all the historical observations, the overall historical influence I_{it} for current timestamp t can be expressed as Equation (2).

$$\begin{aligned} I_{it} &= N_{it_{-1}}(t) + N_{it_{-2}}(t) + N_{it_{-3}}(t) + ... \\ &= N_{it_{-1}}e^{-\lambda} + N_{it_{-2}}e^{-2\lambda} + N_{it_{-3}}e^{-3\lambda} + ... \\ &= e^{-\lambda}(N_{it_{-1}} + e^{-\lambda}(N_{it_{-2}} + e^{-\lambda}(N_{it_{-3}} + ... \\ &= e^{-\lambda}(N_{it_{-1}} + I_{it_{-1}}). \end{aligned} \tag{2}$$

The stream anomaly score of stream S_i is the summation of the data instance anomaly score of current observation N_{it} and the overall historical influence, i.e.,

$$N_i = N_{it} + I_{it}. \tag{3}$$

As shown in Equation (2), the overall historical influence can be incrementally updated with cost $O(1)$ for both time and space complexity. Therefore, *stream anomaly scorer* can be efficiently computed.

4.3.1 Properties of Stream Anomaly Score

The properties of stream anomaly score make our framework insensitive to the transient fluctuation and effective to capture the real anomaly.

Comparing to the transient fluctuation, the real anomaly is more durable. Figure 5 shows the situations of a transient fluctuation (in the left subfigure) and a real anomaly(in the right subfigure). In both situations, the stream behaves normally before timestamp t_x. For the left situation, a transient fluctuation occurs at timestamp t_{x+1}, and then the stream returns to normal at timestamp t_{x+2}. For the right situation, an anomaly begins at timestamp t_{x+1}, lasts for a while till timestamp t_{x+k}, and then the stream returns to normal afterwards. Based on Figure 5, we show two properties about the stream anomaly score.

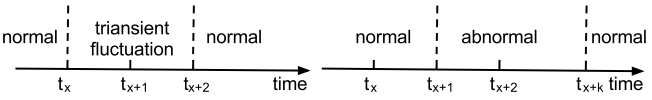


Figure 5: Transient Fluctuation and Anomaly

PROPERTY 1. *The increase of stream anomaly score caused by transient disturbance would decrease over time.*

PROPERTY 2. *The increase of stream anomaly score caused by anomaly would be accumulated over time.*

Similar properties can also be shown for the situation of slight shifts. A slight shift can be treated as two transient fluctuations occur at the beginning and the end of the shift. In the next section, we will leverage these two properties to effectively identify the anomalies in the ALERT STAGE.

4.4 Alert Triggering

Most of the stream anomaly detection solutions [13] identify the anomalies by picking the streams with top-k anomaly scores or the ones whose scores exceed a predefined threshold. However, these two approaches are not practical in real world applications for the following reasons: (1) *Threshold is hard to set.* It requires the users to understand the underlying mechanism of the application to correctly set the parameter. (2) *The number of anomalies are changing all the time.* It is possible that more than k anomaly streams exist at one time, then the top-k approach would miss these real anomalies.

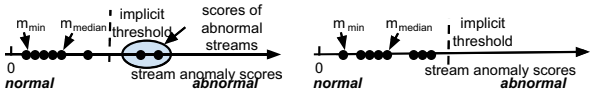


Figure 6: Abnormal Streams Identification

To eliminate the parameters, we propose an unsupervised method to identify and quantify the anomalies by leveraging the distribution of the anomaly scores. The first step is to find the median of the stream anomaly scores (N_{median}). If the distance between a stream anomaly score and the median score is larger than the distance between the median score and the minimal score (N_{min}), the corresponding stream is regarded as abnormal. As shown in Figure 6, this method implicitly defines a dynamic threshold (shown as the dashed line) based on the hypothesis that there is no anomaly. If there is no anomaly, the skewness of the anomaly score distribution should be small and the median score should be close to the mean score. If the hypothesis is true, $N_{median} - N_{min}$ should be close to half of the distance between the minimum score and the maximum score. On the contrary, if a score N_i is larger than $2 \times (N_{median} - N_{min})$, the hypothesis is violated and all the streams with scores at least N_i are abnormal.

Besides the general case, we also need to handle one special case: a transient fluctuation occurs at the current timestamp. According to Property (1) in Section 4.3.1, the effect of transient fluctuation is at most $d_{upper} = N_{it_{x+1}} - N_{jt_{x+1}}$ and it will monotonically decrease. Therefore, even a stream whose anomaly score is larger than $2 \times (N_{median} - N_{min})$, it can still be a normal stream if the difference between its anomaly score and N_{min} is smaller than d_{upper}. To prune the false-positive situations caused by transient fluctuation, the stream is instead identified as abnormal if

$$N_i > max(2(N_{median} - N_{min}), N_{min} + d_{upper}). \tag{4}$$

Another thing needs to be noted is that the stream anomaly scores have a upper bound $\frac{d_{upper}}{1-e^{-\lambda}}$. According to the property of convergent sequence, the stream anomaly scores of all streams would converge to this upper bound. When the values of stream anomaly scores are close to the upper bound, they tend to be close to each other and hard to be distinguished. To handle this problem, we reset all the stream anomaly scores to 0 whenever one of them close to the upper bound.

In terms of the time complexity, the abnormal streams can be found in $O(n)$ time. Algorithm 2 illustrates the algorithm of stream anomalies identification. The median of the scores can be found in $O(n)$ in the worst case using the *BFPRT algorithm [5]*. Besides finding the median, this algorithm also partially sorts the list by moving smaller scores before the median and larger scores after the median, making it trivial to identify the abnormal streams by only checking the streams appearing after the median.

5. EXPERIMENTAL EVALUATION

Algorithm 2 Stream Anomaly Identification

1. **INPUT**: λ, and unordered stream profile list $\mathcal{S} = \{S_1, ..., S_n\}$.
2. $mIdx \leftarrow \lceil \frac{|\mathcal{S}|}{2} \rceil$
3. $N_{median} \leftarrow \text{BFPRT}(\mathcal{S}, mIdx)$
4. $N_{min} \leftarrow \min(S_i.score | 0 \leq i \leq mIdx)$
5. $N_{max} \leftarrow N_{median}$
6. **for** $i \leftarrow mIdx$ to $|\mathcal{S}|$ **do**
7. **if** Condition (4) is satisfied **then**
8. Trigger alert for S_i with score N_i at current time.
9. **if** $N_i > N_{max}$ **then**
10. $N_{max} \leftarrow N_i$
11. **end if**
12. **end if**
13. **end for**
14. **if** N_{max} is close to the upper bound **then**
15. Reset all stream anomaly scores.
16. **end if**

To investigate the effectiveness and efficiency of our framework, we design several sets of experiments with one real world data applications: *anomaly detection over a computing cluster*. Taking these two applications as case studies, we show that our proposed framework can effectively identify the abnormal behavior of streams. It should be pointed out that our proposed framework can also be applied to many other application areas such as PM2.5 environment monitoring, healthcare monitoring, and stock market monitoring.

System anomaly detection is one of the critical tasks in system management. In this set of experiments, we show that our proposed framework can effectively and efficiently discover the abnormal behaviors of the computer nodes with high precision and low latency.

5.1 Experiment Settings

For the experiments, we leverage a distributed system monitoring tool [31] into a 16-node computing cluster. Then we deploy the proposed anomaly detection program on an external computer to analyze the collected trace data in real time. To well evaluate our proposed framework, we terminate all the irrelevant processes running on these nodes. On this node, we intentionally inject various types of anomalies and monitor their running status for 1000 seconds. The source code of injection program is available at `https://github.com/yxjiang/system-noiser`. The details of the injections are listed in Table 1.

Table 1: List of Injections

No.	Time Period	Node	Description
1	[100, 150]	2	Keep CPU utilization above 95.
2	[300, 400]	3	Keep memory usage at 70%.
3	[350, 400]	3	Keep CPU utilization above 95%.
4	[600, 650]	4	Keep memory usage at 70%.
5	[900, 950]	2,5	Keep CPU utilization above 95%.
6	[800, 850]	1-5,7-16	Keep CPU utilization above 95%.

Through these injections, we can answer the following questions about our framework: (1) Whether our framework can identify the anomalies with different types of root causes. (2) Whether our framework can identify multiple anomalies occurring simultaneously.

5.2 Results Analysis

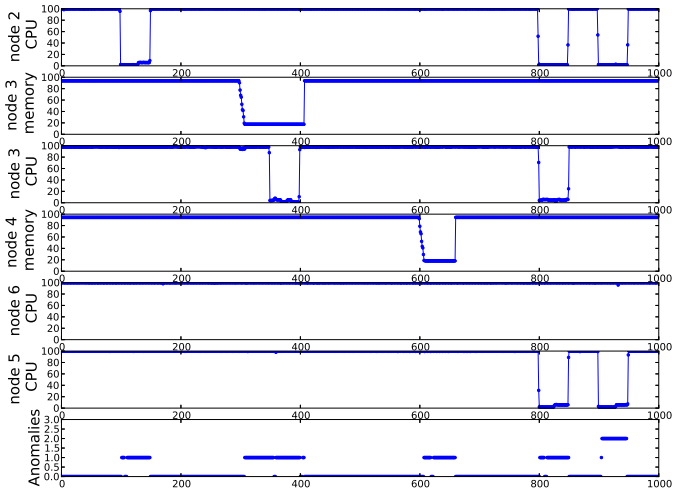


Figure 7: Injections and the captured alerts

Figure 7 illustrates the results of this experiment by plotting the actual injections (top 6 sub-figures) as well as the captured alerts (the bottom subplots), where the x-axis represents the time and y-axis represents the idled CPU utilization, idle memory usage or the number of anomalies in each timestamp. We evaluate the framework from 5 aspects through carefully-designed injections.

1. *Single dimension (e.g. idle CPU utilization or idle memory usage) of a single stream behaves abnormally.* This is the simplest type of anomalies. It is generated by injections No.1 and No.4 in Table 1. As shown in Figure 7, our framework effectively identifies these anomalies with the correct time periods.

2. *Multiple dimensions (e.g. CPU utilization and memory usage) of a single stream behaves abnormally at the same time.* This type of anomalies is generated by injections No.2 and No.3 in Table 1, and our framework correctly captures such anomalies during the time period [300, 400]. One thing should be noted is that the stream anomaly score of node 3 increases faster during the time period [350, 400] than the time period [300, 350]. This is because two types of anomalies (CPU utilization and memory usage) appear simultaneously during the time period [350, 400].

3. *Multiple streams behave abnormally simultaneously.* This type of anomalies is generated by injection No.5. During the injection time period, our framework correctly identifies both anomalies (on node 2 and node 5).

4. *Stable but abnormal streams.* This kind of anomaly is indirectly generated by injection No.6 in Table 1. This injection emulates the scenario that all the nodes but one (i.e., node 6) in a cluster received the command of executing a task. As is shown, although the CPU utilization of node 6 behaves stable all the time, it is still considered to be abnormal during the time period [800, 850]. This is because it remains idle when all the other nodes are busy.

5. *Transient fluctuation and slight delay would not cause false-positive.* As this experiment is conducted in a distributed environment, delays exist and vary for different nodes when executing the injections. Despite

this intervention, our framework still does not report transient fluctuations and slight delays as anomalies.

Based on the evaluation results, we find that our solution is able to correctly identify all the anomalies in all these 5 different cases.

5.3 Results Comparison

To demonstrate the superiority of our framework, we also conduct experiments to identify the anomalies with the same injection settings using the alternative methods including *contextual anomaly detection (CAD)* and *rule-based continuous query (Rule-CQ)*. The contextual anomaly detection is equivalent to the snapshot scoring in our framework. For the rule-based continuous query, we define three rules to capture three types of anomalies, including high CPU utilization (rule 1), low CPU utilization (rule 2), and high memory usage anomalies (rule 3), respectively. Different combinations of the three rules are used in the experiments.

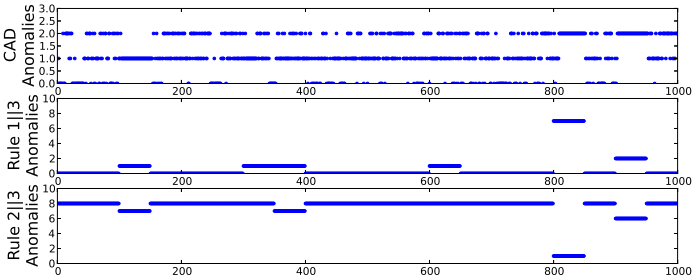


Figure 8: Generated alerts by CAD and Rule-CQ

The generated alerts of these methods are shown in Figure 8, where the x-axis denotes the time and y-axis denotes the number of anomalies. As illustrated, the contextual anomaly detection method generates a lot of false alerts. This is because this method is sensitive to the transient fluctuation. Once an observation deviates from the others at a timestamp, an alert would be triggered. For Rule-CQ method, we experiment all the combinations and report the results of the two best combinations: C1 (rule 1 or rule 2) and C2 (rule 2 or rule 3). Similarly, the Rule-CQ method also generates many false alerts since it is difficult to use rules to cover all the anomaly situations. Table 2 quantitatively shows the precision, recall, and F-measure of the three methods as well as the results of our method. The low-precision and high-recall results of CAD and Rule-CQ indicate that all these method are too sensitive to fluctuations.

Table 2: Measures of different methods performed on the trace data in distributed environment

Method / Measure	precision	recall	F-measure
CAD	0.4207	1.0000	0.5922
C1: Rule 1||3	0.5381	1.0000	0.6997
C2: Rule 2||3	0.0469	1.0000	0.0897
Our method (worst case)	0.9832	0.8400	0.9060

5.4 A real system problem detected

We have identified a real system problem when deployed our framework on two computing clusters in our department. In one of the clusters, we continuously receive alerts. Logging into the cluster, we find the CPU utilization is high even no tasks are running. We further identify that the high CPU utilization is caused by several processes named *hfsd*. We reported the anomaly to IT support staffs and they confirmed that there exist some problems in this cluster. The high CPU utilization is caused by continuous attempts to connect to a failure node in the network file system. After fixing this problem, these out-of-expectation but real alerts disappear.

6. RELATED WORKS

Although anomaly detection has been studied for years [15, 8]. To the best of our knowledge, our method is the first one that focused on mining contextual collective anomalies among multiple streams in real time. In this section, we briefly review two closely related areas: mining outliers from data streams and mining outliers from trajectories.

With the emerging requirements of mining data streams, several techniques have been proposed to handle the data incrementally [33, 20, 19, 28, 18]. Pokrajac et al. [25] modified the static Local Outlier Factor (LOF) [6] method as an incremental algorithm, and then applied it to find data instance anomalies from the data stream. Takeuchi and Yamanishi [16] trained a probabilistic model with an online discounting learning algorithm, and then use the training model to identify the data instance anomalies. Angiulli and Fassetti [3] proposed a distance-based outlier detection algorithm to find the data instance anomalies over the data stream. However, all the aforementioned works focused on the anomaly detection of a single stream, while our work is designed to discover the contextual collective anomalies over multiple data streams.

A lot of works have been conducted on trajectory outlier detection. One of the representative work on trajectory outlier detection is conducted by Lee et al. [21]. They proposed a partition-and-detection framework to identify the anomaly sub-trajectory via the distance-based measurement. Liang et al. [29] improved the efficiency of Lee's work by only computing the distances among the sub-trajectories in the same grid. As the aforementioned two algorithms require to access the entire dataset, they cannot be adapted to trajectory streams. To address the limitation, Bu et al. [7] proposed a novel framework to detect anomalies over continuous trajectory streams. They built local clusters for trajectories and leveraged efficient pruning strategies as well as indexing to reduce the computational cost. However, their approach identified anomalies based on the local-continuity property of the trajectory, while our method does not make such an assumption. Our approach is close to the work of Ge et al. [13], where they proposed an incremental approach to maintain the top-K evolving trajectories for traffic monitoring. However, their approach mainly focused on the geospatial data instances and ignored the temporal correlations, while our approach explicitly considers the temporal information of the data instances.

7. CONCLUSION

In this paper, we propose a real time anomaly detection framework to identify the contextual collective anomalies from a collection of streams. Our proposed method firstly quantifies the snapshot level anomaly of each stream based on the contextual information. Then the contextual information and the historical information are used in combina-

tion to quantify the anomaly severity of each stream. Based on the distribution of the stream anomaly scores, an implicit threshold is dynamically calculated and the alerts are triggered accordingly. To demonstrate the usefulness of the proposed framework, several sets of experiments are conducted to demonstrate its effectiveness and efficiency.

8. ACKNOWLEDGEMENT

The work was supported in part by the National Science Foundation under grants DBI-0850203, HRD-0833093, CNS-1126619, and IIS-1213026, the U.S. Department of Homeland Security under grant Award Number 2010-ST-06200039, Army Research Office under grant number W911NF-10-1-0366 and W911NF-12-1-0431, National Natural Science Foundation of China under grant number 61300053, and an FIU Dissertation Year Fellowship.

9. REFERENCES

[1] A.Arasu, B.Babcock, S.Babu, M.Datar, K.Ito, I.Nizhizawa, J.Rosenstein, and J.Widom. Stream: The stanford stream data manager. In *SIGMOD*, 2003.

[2] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *SIGMOD*, 2008.

[3] F. Anguilli and F. Fassetti. Detecting distance-based outliers in streams of data. In *CIKM*, 2007.

[4] Y. Bai, F. Wang, P. Liu, C. Zaniolo, and S. Liu. Rfid data processing with a data stream query language. In *ICDE*, 2007.

[5] M. Blum, R. Floyd, V.Pratt, R.Rivest, and R. Tarjan. Time bounds for selection. *Journal of Computer System Science*, 1973.

[6] M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *SIGMOD*, 2000.

[7] Y. Bu, L. Chen, A. W.-C. Fu, and D. Liu. Efficient anomaly monitoring over moving object trajectory streams. In *KDD*, 2009.

[8] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 2009.

[9] S. Chandrasekaran, O.Cooper, A.Deshpande, M.J.Franklin, J.M.Hellerstein, W.Hong, S.Krishnamurthy, S.R.Madden, V.Raman, F.Reiss, and M.A.Shah. Telegraphcq: Continous dataflow processing for an uncertain world. In *CIDR*, 2003.

[10] D.Carney, U.Cetintemel, M.Cherniack, C. Convey, S.Lee, G.Seidman, M.Stonebraker, N.Tatbul, and S.Zdonik. Monitoring streams: A new class of data management applications. In *VLDB*, 2002.

[11] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[12] R. A. Fisher, F. Yates, et al. Statistical tables for biological, agricultural and medical research. *Statistical tables for biological, agricultural and medical research.*, (Ed. 3.), 1949.

[13] Y. Ge, H. Xiong, Z.-H. Zhou, H. Ozdemir, J. Yu, and K. C. Lee. Top-eye: Top-k evolving trajectory outlier detection. In *CIKM*, 2010.

[14] M. Gupta, A. B. Sharma, H. Chen, and G. Jiang. Context-aware time series anomaly detection for complex systems. In *WORKSHOP NOTES*, page 14, 2013.

[15] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 2004.

[16] J. ichi Takeuchi and K. Yamanishi. A unifying framework for detecting outliers and change points from time series. *IEEE Transactions on Knowledge and Data Engineering*, 2006.

[17] G. Jiang, H. Chen, and K. Yoshihira. Modeling and tracking of transaction flow dynamics for fault detection in complex systems. *Dependable and Secure Computing, IEEE Transactions on*, 3(4):312–326, 2006.

[18] Y. Jiang, C.-S. Perng, T. Li, and R. Chang. Asap: A self-adaptive prediction system for instant cloud resource demand provisioning. In *ICDM*, 2011.

[19] Y. Jiang, C.-S. Perng, T. Li, and R. Chang. Intelligent cloud capacity management. In *NOMS*, 2012.

[20] Y. Jiang, C.-S. Perng, T. Li, and R. Chang. Self-adaptive cloud capacity planning. In *SCC*, 2012.

[21] J.-G. Lee, J. Han, and X. Li. Trajectory outlier detection: A partition-and-detect framework. In *ICDE*, 2008.

[22] B. Lo, S. Thiemjarus, R. king, and G. Yang. Body sensor network-a wireless sensor platform for pervasive healthcare monitoring. In *PERVASIVE*, 2005.

[23] B. Mozafari, K. Zeng, and C. Zaniolo. High-performance complex event processing over xml streams. In *SIGMOD*, 2012.

[24] A. Oliner, A. Ganapathi, and W. Xu. Advances and challenges in log analysis. *Comm. of ACM*, 2012.

[25] D. Pokrajac, A. Lazarevic, and L. J. Latecki. Incremental local outlier detection for data streams. In *CIDM*, 2007.

[26] Y. Song and L. Cao. Graph-based coupled behavior analysis: a case study on detecing collaborative manipulations in stock markets. In *IEEE world congress on computational intelligence*, 2012.

[27] storm. http://storm-project.net.

[28] L. Tang, C. Tang, L. Duan, Y. Jiang, C. Zeng, and J. Zhu. Movstream: An efficient algorithm for monitoring clusters evolving in data streams. In *Grc*, 2008.

[29] L. Tang, C. Tang, Y. Jiang, C. Li, L. Duan, C. Zeng, and K. Xu. Troadgrid: An efficient trajectory outlier detection algorithm with grid-based space division. In *NDBC*, 2008.

[30] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[31] C. Zeng, Y. Jiang, L. Zheng, J. Li, L. Li, H. Li, C. Shen, W. Zhou, T. Li, B. Duan, et al. Fiu-miner: a fast, integrated, and user-friendly system for data mining in distributed environment. In *KDD*, pages 1506–1509. ACM, 2013.

[32] L. Zheng, C. Shen, L. Tang, T. Li, S. Luis, S.-C. Chen, and V. Hristidis. Using data mining techniques to address critical information exchange needs in disaster affected public-private networks. In *KDD*, 2010.

[33] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, 2002.

STOUT : Spatio-Temporal Outlier detection Using Tensors

Yanan Sun
Information Systems Department,
University of Maryland, Baltimore County,
MD, USA
sun8@umbc.edu

Vandana P. Janeja
Information Systems Department,
University of Maryland, Baltimore County,
MD, USA
vjaneja@umbc.edu

ABSTRACT

Spatio-Temporal data is inherently large since each spatial node has spatial attributes and may also be associated with large amounts of measurement data captured over time. In such large and multi-dimensional data identifying anomalies can be a challenge due to the massive data size and relationships among spatial objects. Discovering anomalies in spatio-temporal data is relevant in several domains, such as detecting rare disease outbreaks, detecting oil spills, discovering regions with highway traffic congestion. Most existing techniques for discovering anomalies in spatio-temporal data may find the spatial outliers first and then identify the spatio-temporal anomalies from the data of that specific spatial location. Alternatively, some approaches may discover anomalous time periods and then discover the unusual spatial location in them. This may lead to identifying incorrect spatio-temporal outliers or missing important spatio-temporal phenomena due to the elimination of information after each step. Thus, there is a need to address to capture both space and time simultaneously. Tensor is a multi-dimensional array and can facilitate integrating complex relationships in spatio-temporal data. Their mathematical framework can help detect spatio-temporal outliers in an effective and efficient manner. In this paper, we present our novel approach addressing the key limitation of existing spatio-temporal outlier detection methods by using an efficient tensor based model that supports complex relationships in spatio-temporal data to detect outliers by looking at space and time simultaneously as well as handling the scalability issue when it comes to manipulating large datasets. We present a class of algorithms to discover different types of spatio-temporal outliers namely point based and window based outliers. We discuss detailed experimental results for each of the algorithms proposed and also present comparative results.

1. INTRODUCTION

The focus of this paper is spatio-temporal outlier detection in large datasets utilizing tensors in such a way so that we can accommodate space and time dimensions simultaneously. This essentially means that for outlier detection we do not create spatial neighborhoods first and then do temporal outlier detection or create time slices and then do spatial outlier detection but capture both the spatial and temporal proximities simultaneously leading to a discovery of outliers in these spatio-temporal proximities or neighborhoods. Barnett[10] defines an outlier as an observation (or subset of observations) which appears to be inconsistent with respect to the remainder of the dataset. We extend this definition to spatio-temporal datasets. We define spatio-temporal outliers as the spatio-temporal readings which are significantly different from their spatio-temporal neighborhood. However, we go beyond this point based outlier detection to addressing window based outliers as well. A spatial window is a set of contiguous points in a region. A window based outlier is a set of contiguous points in space that are unusual with respect to the rest of the data.

Most existing techniques for discovering anomalies [33, 15, 13] in spatio-temporal data may find the spatial outliers first and then identify the spatio-temporal anomalies from the data of that specific spatial location. Alternatively, some approaches may discover anomalous time periods and then discover the unusual spatial location. This may lead to identifying incorrect spatio-temporal outliers or missing important spatio-temporal phenomena due to the elimination of information after each step. Thus, there is a need to address capturing both space and time simultaneously.

In general, spatio-temporal outlier detection can be challenging due to the complex relationships among spatial objects. The lack of efficient spatio-temporal data representation models makes it even harder for formulating spatio-temporal anomaly detection methods to deal with space and time at the same time.

Spatio-temporal data is high-resolution data, where the size of the data grows rapidly over time. For example, oceanographic data: to study the El Nino phenomena, the TAO / TRITON array of sensors [6] collected the oceanographic and meteorological data at approximately 70 sensor locations in Tropical Pacific Ocean every minute. Spatio-temporal datasets consist of spatial attributes, temporal attributes and non-spatial attributes. To handle such large spatio-temporal datasets there is a need to develop effective algorithms.

We utilize tensors to manipulate such large spatio-temporal datasets. A tensor is a multi-dimensional array and can be utilized to manipulate multi-dimensional and multi-variate

ODD^2'14, August 24th, 2014, New York, NY, USA.

data. It has a concise mathematical framework for formulating and solving complex data problems efficiently. Tensors can handle complex relationship in spatio-temporal data and the mathematical framework can help detect spatio-temporal outliers in an effective and efficient manner. Tensor multiplication can integrate spatial and temporal aspects in spatio-temporal data at the same time. Tensor decomposition methods can help us to keep the computational cost low as well as maintain the insight and underlying structure of the large spatio-temporal data. In this paper, we detect the spatio-temporal outliers in large datasets utilizing tensors in an effective and efficient manner.

1.1 Motivation

Spatio-temporal data analysis has received a great deal of attention due to the explosive growth of spatio-temporal data. Anomaly detection in such datasets is highly relevant in many domains: (a) environment: detect boundaries of gas leak/oil spill locations, detect water pollution zones; (b) medicine: detect tumor cells, detect rare disease outbreaks ; and (c) transportation: discover highway congestion areas.

Next, we take one example to elaborate our research goal. Let us take a highway traffic data as a sample dataset where our goal is to find a traffic congestion section on the highway, the congestion duration and special situation(accident, road work, detour, etc). The answer to this problem can have a direct impact to the traffic control and management. Accurately identifying the congested locations and congestion duration can also provide support for better resource planning.

We start with the data of the vehicle count at each spatial location ((a), (b), (c), (d) and (e) in Figure 1) at every hour. Indeed such datasets are routinely collected by different state highway administrations [1, 5, 3, 2, 4]. Here spatial neighborhood is the cluster formed based on using vehicle count and spatial location as circled in Figure 1). Temporal neighborhood is the cluster formed based on the vehicle count and temporal interval. Spatio-temporal neighborhood is the cluster formed based on the vehicle count, spatial location and temporal intervals. In table 1, location (a), (b) and (c) consistently have higher vehicle count than other locations (d) and (e) from hour 15 to hour 17 as compared to other locations. Location (a), (b) and (c) are in the same spatial neighborhood. Thus, we consider this section from location (a) to (c) as the congestion section. From hour 15 through hour 17, all five locations (a), (b), (c), (d) and (e) have consistently higher vehicle count than the other time period hour 18. Thus, hour 15 to hour 17 is the congestion duration hours. However, the location (d) at hour 18 has higher vehicle count than its neighboring locations (e), when hour 18 does not have traffic congestion. This might indicate there is some special circumstance at location (d) at hour 18. If we just look at vehicle count at location (d) in all four hours (hour 15, 16, 17 and 18), then the location(d) and the vehicle count at location(d) at hour 18 are not considered as unusual. But if we consider the congestion duration hours and locations, then the vehicle count at location(d) at hour 18 becomes an anomaly.

In summary, if we simply look at spatial dimension, then location (a),(b) and (c) are potential outliers. If we simply look at temporal dimension, then hour 15, 16 and 17 are outliers. Thus if we look at space dimension and then time or time dimension and then space dimension we will still miss

Figure 1: I-270

Location	Hour 15	Hour 16	Hour 17	Hour 18
(a)	360	350	340	120
(b)	350	370	360	140
(c)	340	360	350	130
(d)	200	210	210	250
(e)	210	190	200	100

Table 1: Highway traffic data example

the spatio-temporal outlier at location (d) at hour 18. Unless we look at both space and time dimensions simultaneously, we cannot identify such an outlier.

Traffic congestion is considered as outlier, because it has unusually increased volume of traffic at a certain time. If we look at the spatial aspect, we consider the contiguous spatial locations where the traffic congestion occurs as 'spatial outliers'. If we look at the temporal aspect, we consider the contiguous temporal intervals when the traffic congestion occurs as 'temporal outliers'. If we look at both spatial and temporal aspects, we consider the spatio-temporal readings where the special situation occurs as 'spatio-temporal outliers'. This can translate to discovering a location at a specific point in time which has severe congestion.

In other words, the spatial outlier is the spatial location which exhibits significantly different behavior as compared to the rest of its spatial neighborhood; the temporal outlier is the temporal interval that is significantly different from the rest of the time period; the spatio-temporal outlier is the spatio-temporal reading (or a set of readings) significantly different from the spatio-temporal neighborhood. Accordingly, spatio-temporal outlier detection requires us to look at both spatial and temporal aspect.

The existing spatio-temporal outlier detection methods can be grouped into three categories, namely satscan-based ([23, 25, 18]) distance-based (Jin et al., 2006 [20], Sun et al., 2005 [31], Adam et al., 2004 [8], and Subramaniam et al., 2006 [29]), density-based (Derya Birant, 2006 [15] and Subramaniam et al., 2006 [29]), and others (Wang et al., 2008 [33], Davy et al., 2006 [14], and Cheng and Li, 2004, Cheng and Li, 2006 [11, 12]). They have several limitations when it comes to identifying the spatio-temporal outliers: *1)* They do not address the space and time dimensions simultaneously. Most existing techniques applied to identifying outliers in spatio-temporal datasets only look at one dimension at a time. For example, they may find the spatial neighbor-

hoods first and then identify the spatio-temporal anomalies from the data of that spatial neighborhood. Thus, we need to consider the space and time dimensions simultaneously to produce accurate results. *2*) They need to predefine the search shape or the data distributions in order to identify the interesting events. *3*) Most approaches are not designed for large datasets, which means it may have high computational cost when the dataset is large or it will sacrifice the accuracy to keep up with the computational cost.

In this paper, our goal is to detect spatio-temporal anomalies by looking at space and time simultaneously to produce high accuracy as well as to handle the scalability issue when it comes to manipulating large datasets.

1.2 Contributions

We present our novel approach for *S*patio-*T*emporal *O*utlier Detection *U*sing *T*ensors (STOUT) that supports complex relationships in spatio-temporal data to detect outliers by looking at space and time simultaneously as well as handling the scalability issue when it comes to manipulating large datasets. By taking advantage of tensor and tensor decomposition, we present a class of algorithms to discover different types of spatio-temporal outliers namely point based and window based outliers. We discuss detailed experimental results for each of the algorithms proposed. Specifically, in this paper we make the following contributions:

1. We construct a third-order tensor based spatio-temporal data representation model with modes of $time \times location \times measurement$.
2. We propose a unified spatio-temporal outlier detection framework. By utilizing tensor multiplication and tensor decomposition, we handle space and time simultaneously. Our tensor-based unified spatio-temporal outlier detection framework produces accurate results and it can be easily extended to higher-order tensor with high dimensional datasets.
3. We provide extensive experimental results for identifying point based and window based spatio-temporal outliers and provide comparative results. We utilize synthetic datasets which are derived from existing spatio-temporal datasets.

The rest of the paper is organized as follows. Section 2 introduces the necessary background on tensor and tensor mathematical framework. Section 3 explains our approach in detail. Section 4 presents the experimental results. Finally, section 5 presents the conclusion and future extensions of our approach.

2. PRELIMINARIES

In the last ten years, interest in tensor decompositions has expanded to many fields ([22, 21]). Examples include signal processing, numerical linear algebra, computer vision, numerical analysis, data mining, graph analysis, neuroscience, and more ([27, 19, 16, 28, 17, 26, 24, 7, 34]).

Tensors present an intuitive way to model a complex data structure problem such as spatio-temporal data mining which deals with spatial, temporal and multivariate attribute data. In this section, we discuss the basics of tensor and its mathematical framework. The notations used throughout the paper are outlined in Table 2.

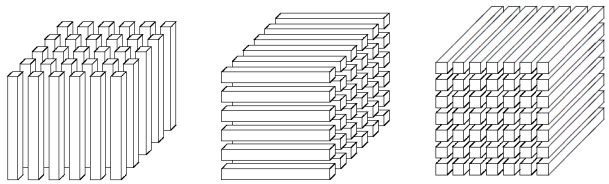
2.1 Tensor and Tensor Elements

A tensor can be described as a multi-dimensional array, n-way or n^{th}-order representation of the data. The number of dimensions of a tensor is referred to as order. A first-order and a second-order tensor are vector and matrix respectively. Each individual dimension of a tensor is referred to as *mode*.

In this paper, a tensor is denoted by a Calligraphy letter, eg. $\mathcal{X}$ and the element (i, j, k) of a third-order tensor $\mathcal{X}$ is denoted by x_{ijk}.

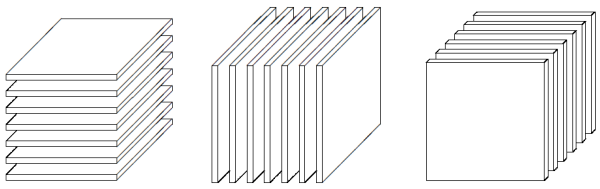
Fibers are the higher-order analogue of matrix row and columns. A fiber is defined by fixing every index but one. A third-order tensor has column, row, and tube fibers, denoted by $x_{:jk}$, $x_{i:k}$ and $x_{ij:}$, respectively (as shown in Figure 2).

Slices are two-dimensional section of a tensor, defined by fixing all but two indices. A third-order tensor has horizontal, lateral, and frontal slices, denoted by $X_{i::}$, $X_{:j:}$ and $X_{::k}$, respectively (as shown in Figure 3).



(a) Mode-1 fiber $x_{:jk}$ (b) Mode-2 fiber $x_{i:k}$ (c) Mode-3 fiber $x_{ij:}$

Figure 2: Fibers of a third-order tensor.



(a) Horizontal slices: $X_{i::}$ (b) Lateral slices: $X_{:j:}$ (c) Frontal slices: $X_{::k}$

Figure 3: Slices of a third-order tensor.

$\mathcal{X}$	Tensor(order three or higher)
X	Matrix
x_{ijk}	Tensor $\mathcal{X}$ element
$x_{:jk}$	Mode-1 fibers
$x_{i:k}$	Mode-2 fibers
$x_{ij:}$	Mode-3 fibers
$X_{i::}$	Horizontal slice of a third-order tensor
$X_{:j:}$	Lateral slice of a third-order tensor $\mathcal{X}$
$X_{::k}$	Frontal slice of a third-order tensor $\mathcal{X}$
X_k	kth frontal slice of tensor $\mathcal{X}$
$\mathcal{X}_{(i)}$	Mode-i unfolding of tensor $\mathcal{X}$
$\mathcal{X} \times_i U$	i-mode matrix product of a tensor

Table 2: Summary of notations

2.2 Tensor Unfolding and Multiplication

For understanding tensor multiplication, first we explain how to unfold a tensor into a matrix. Tensor matricization, also known as unfolding or flattening, is a process of arranging the elements of a tensor into a matrix [22, 21]. The

mode-i matricization of a tensor $\mathcal{X}$ is denoted by $X_{(i)}$ and arranges the mode-n fibers to the columns of the resulting matrix.

We next define n-mode multiplication of a third-order tensor by a matrix. Let $\mathcal{X}$ be a tensor of size $I_1 \times I_2 \times I_3 \times ... \times I_N$ and let U be a matrix of size of $J_n \times I_n$.

$$(\mathcal{X} \times_n U)_{i_1 \ldots i_{n-1} j_{n+1} .. i_N} = \sum_{i_n=1}^{I_N} x_{i_1 i_2 .. i_N} u_{j i_n}.$$

Each mode-n fiber is multiplied by matrix U. The idea can also be expressed in terms of unfolded tensors as follow:

$$\mathcal{Y} = \mathcal{X} \times_n U \Leftrightarrow Y_{(n)} = U \times X_{(n)}$$

2.3 Tensor Decomposition

High-order Singular Value Decomposition (HOSVD) and CANDECOMP/PARAFAC (CP) tensor decomposition are two popular tensor decomposition methods.

High-order Singular Value Decomposition (HOSVD) is known as Tucker1 decomposition method. It was introduced by Tucker in 1966 and is one of the three methods for computing a decomposition. The basic idea of Tucker decomposition is to decompose a tensor into a core tensor by multiplication of matrices along each mode to best capture the variation in any mode independent of others [32]. In a third-order tensor, HOSVD sets two of the factor matrices to be the identity matrix [32]. This is represented as:

$\mathcal{E} \leftarrow \mathcal{X} \times_1 U^{(1)T} \times_2 U^{(2)T} \times_3 U^{(3)T}$

where $\mathcal{X}$ is the core tensor and $U^{(N)}$ is the factor matrix. and each element of a third-order tensor $\mathcal{X}$ can be written as

$\mathcal{E}_{ijk} \approx \sum_{p=1}^{I} \sum_{q=1}^{J} \sum_{s=1}^{K} u_{ip}^{(1)} u_{jq}^{(2)} u_{ks}^{(3)} x_{pqs}$

The CANDECOMP/PARAFAC (CP) tensor decomposition can decompose the tensor into a sum of multiple rank-one tensors, usually based on the rank(R) of the tensor. In a third-order tensor, this is represented as:

$\mathcal{E}_{ijk} \approx \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}$ for $i = 1, \ldots I, j = 1, \ldots, J, k = 1, \ldots, K$.

3. APPROACH

In the following, we propose a novel approach for Spatio-Temporal Outlier detection using Tensors (STOUT) by modeling the spatio-temporal data as a tensor and proposing a class of outlier detection methods. We utilize the tensor to propose multiple algorithms to address efficiency and the spatio-temporal simultaneity aspect as follows: (1) For addressing the simultaneous nature of space and time we multiply spatial contiguity matrix and temporal contiguity matrix on space and time mode respectively. This way, the new tensor is already enriched with spatial and temporal contiguity relationships. (2) For efficiency we model the data with small bins of the tensor along time. Each small tensor is of the same size by including all spatial sensor locations and their respective measurements in that time interval.

We create a class of algorithms combining the two aspects above such that we can compare the variants and provide the optimal method for both efficiency and simultaneous representation of space and time. We create algorithms for outlier detection in tensors with binning with two variations of using multiplication and the other without using the multiplication. We also discuss variations of these algorithms by changing the order of multiplication and binning by performing multiplication first followed by binning. Lastly, we provide variations for both point based and window based outlier detection methods. We next discuss out approach and algorithms in detail.

3.1 STOUT

Our approach consists of the following four distinct steps.

(1). The first step is to build tensor based on fixed-size temporal binning. We define the spatio-temporal tensor as follows:

DEFINITION 1. ***Spatio-temporal Tensor*** *Tensor $\mathcal{X} \in \Re^{N_1 \times N_2 \times N_3}$, is a third-order tensor with modes of (Time $\times$ Location $\times$ Measurement) and the (i, j, k) element of the tensor is denoted by x_{ijk}. The horizontal, lateral, and the frontal slices of the tensor $\mathcal{X}$ are denoted by $X_{t::}$, $X_{:l:}$ and $X_{::m}$ respectively. The first (time), second (location) and third (measurement) frontal slices of tensor $\mathcal{X}$, may be denoted more compactly as $X_1(X_t)$, $X_2(X_l)$ and $X_3(X_m)$.*

We next bin the tensor using equal width binning. When it comes to spatio-temporal data, the data grows rapidly over time and by nature it should maintain certain pattern along each bin. It's reasonable to craft the data along time mode. For example, in highway traffic data, we should see certain traffic volume pattern for example rush hour every day. The window size can be defined by user based on the nature of data. Here, we use fixed-size binning method, which means each bin tensor is of the same size in every mode, $X_1, X_2, \ldots, X_i, \ldots, X_n$.

Here, although we are using binning, we still accommodate the space and time simultaneously in each bin tensor. We can imagine one bin tensor as a month worth of data, alternatively it can be hours or minutes of data. Binning is allowing us to deal with large amounts of data more efficiently. However, it does not affect how we analyze the spatio-temporal data. Intuitively, it is better to have somewhat smaller chunks since creating large bins may bring in points which are not very relevant to each other as they move further and further in time. We also provide experimental results to show that the bin size does not affect the accuracy of outlier detection results substantially.

We also identify a base tensor from the bins to create a benchmark with which we compare the bin tensors to identify outliers. We define a base tensor as follows:

DEFINITION 2. ***Base Tensor*** *Base tensor is a spatio-temporal tensor $\mathcal{X}_{base} \in \Re^{N_1 \times N_2 \times twin}$ with size of $N_1 \times N_2 \times$ twin where twin is the bin size. We generate the base tensor by taking average value of few random bins.*

To address the simultaneous accommodation of space and time we derive a new enriched tensor $\mathcal{N}$ by multiplying the spatio-temporal tensor $\mathcal{O}$ with the spatial contiguity matrix and temporal contiguity matrix on the space and time mode respectively. The spatial contiguity matrix represents the pairwise spatial distance relationship between the set of spatial nodes in the region. If we have N spatial nodes then a spatial contiguity matrix is an $N \times N$ matrix. Similarly the temporal contiguity matrix represents the pairwise temporal distance relationship between the set of time periods under

observation. If we have M time periods then the temporal contiguity matrix is $M \times M$ matrix. We define the contiguity matrices as follows:

DEFINITION 3. ***Spatial Contiguity Matrix*** *represents the spatial neighborhood relationship between spatial nodes, which is determined based on the pairwise physical distance between two spatial locations s_i and s_j, denoted as SC. If the pairwise distance between s_i and s_j is within a threshold δ_s, then SC_{ij} is equal to 1, otherwise SC_{ij} is equal to 0.*

DEFINITION 4. ***Temporal Contiguity Matrix*** *represents the temporal neighborhood relationships between time periods and is determined based on equal frequency binning method, denoted as TC. If the time interval between two time periods t_i and t_j is within the threshold δ_t, then TC_{ij} is equal to 1, otherwise TC_{ij} is equal to 0.*

(2). We next decompose the base tensor $\mathcal{X}_{base}$ and current bin tensor $\mathcal{X}_{current}$. Given a tensor $\mathcal{X}_{current}$, we apply CP decomposition introduced earlier to decompose the tensor in order to identify the differences between the base tensor and the current bin tensor in the next step. After the CP decomposition we get the $\lambda_{base}, \lambda_{current}$, factor matrices $S_{base}, T_{base}, S_{current}$ and $T_{current}$ for space and time mode.

(3). Now instead of looking at all the cells in the current bin tensor for outliers we identify spatio-temporal regions which may potentially have outliers. For this we calculate the difference between the base and the current bin tensor and find out the spatial nodes and time periods where the absolute measurement values are greater than a threshold.

By applying CP decomposition, we calculate the differences on both space and time mode simultaneously. This narrows down the outlier search space substantially.

(4). Now we perform outlier detection on the narrowed search space which is really a subspace of the tensor that may potentially have outliers. For each cell in this subspace we compute the Neighborhood Dynamicity β_{ij} of the cell $\mathcal{X}_{imj}$ with the potential combination of space and time to identify the spatio-temporal outliers. Since we already have the spatial and temporal contiguity matrices we identify the spatio-temporal neighborhoods for each cell as the adjacent cells which are in direct spatial and temporal proximity.

We next formally define the Neighborhood dynamicity.

DEFINITION 5. ***Neighborhood Dynamicity*** *β β is the measurement distance within the spatio-temporal neighborhood divided by measurement distance outside of the neighborhood.*

We use this ratio to determine which cell in a spatio-temporal neighborhood is the dynamic cell and can be labeled as an outlier. If the value of β is over certain threshold, that cell is considered as outlier.

$$\beta = \frac{D_{inside\ spatio-temporalneighborhood}}{D_{outside\ spatio-temporalneighborhood}}$$

This approach applies to both point based and window based outliers. Firstly, the multiplication of the spatial and temporal contiguity matrices captures the entire proximity information about a cell in the tensor. Thus, it really represents the neighborhood in proximity. This helps capture the unusual windows since each cell is really representing a window (including the cell and its immediate neighbors). Secondly, when we compute the difference between the current bin tensor and the base tensor, if its fluctuation is a lot more than the base tensor over time, we can capture which time period actually changes the most and then calculate the Neighborhood Dynamicity β to identify the spatio-temporal outliers.

In this paper, we propose three algorithm with variations to detect spatio-temporal outliers. The algorithm 1 (we refer to this as STOUT -NoM) follow the four steps we show above, however, we do not take advantage of spatial contiguity matrix and temporal contiguity matrix. That is we do not perform the matrix multiplication here. Thus, in algorithm 1 we do not enrich the tensor with the spatial and temporal contiguity matrices. This will allow us to see the impact of the simultaneity (accommodating space and time together) in the other algorithms. The algorithm 2 (referred to as STOUT-AM) and algorithm 3 (referred to as STOUT-BM) utilize the tensor multiplication where we multiply our spatio-temporal tensor with spatial contiguity matrix and temporal matrix on the space mode and time mode respectively. The difference between these two methods is that algorithm 2 (STOUT-AM) applies the multiplication after binning where as algorithm 3 (STOUT-BM) applies the multiplication before the binning. Here we want to see the distinction in using contiguity matrices before vs. after binning. The logic is that as the data comes in the contiguity matrix may change so if we perform binning after multiplication our contiguity matrices are static throughout the process. However, if we apply multiplication after binning then we are renewing the contiguity matrices. The intuition is that STOUT-AM should give us better results, which is indeed the case as demonstrated later in the section 4.

Algorithm 1 STOUT-NoM: Spatio-temporal outlier detection without multiplication

Require: Spatio-temporal matrix X_{smt}, size $N_1 \times N_2 \times N_3$, Temporal window size *twin*, Space difference threshold $thred_s$, Time difference threshold $thred_t$, Beta threshold $thred_{beta}$, Number of bins *iter*

1: **for** $i = 1 \rightarrow iter$ **do**
2: *Create data bins along the time mode and build spatio-temporal bin tensor*
3: $\mathcal{X}_i$=Tensor($X_{(s,m,(i \times twin+1:(i+1) \times twin))}$)
4: *Tensor decomposition*
5: $(\lambda_{base}, S_{base}, M_{base}, T_{base})$=CP($\mathcal{X}_{base}$)
6: $(\lambda_i, S_i, M_i, T_i)$=CP($\mathcal{X}_i$)
7: *Calculate the differences between base tensor and current tensor*
8: $diff_s = \lambda_{base} \times S_{base} - \lambda_i \times S_i$
9: $diff_t = \lambda_{base} \times T_{base} - \lambda_i \times T_i$
10: *Calculate the Neighborhood Dynamicity β_{ij}*
11: **if** $diff_{s_i} \geq thred_s$ **then**
12: **if** $diff_{t_j} \geq thred_t$ **then**
13: Calculate $beta_{imj}$
14: **if** abs($beta_{imj}$) $\geq thred_{beta}$ **then**
15: X_{imj} is the spatio-temporal outlier
16: **end if**
17: **end if**
18: **end if**
19: **end for**

In algorithm 1 STOUT-NoM, line 2-3 generates bins along the time mode and builds spatio-temporal bin tensors. Line 4-6 applies CP tensor decomposition on the base tensor $\mathcal{X}_{base}$ and current bin tensor $\mathcal{X}_i$. We get $\lambda_{base}, \lambda_{current}$, factor matrices $S_{base}, T_{base}, S_{current}$ and $T_{current}$. Line 7-9

calculates the differences between base tensor and current bin tensor in order to narrow down the search space. Line 10-19 calculates the neighborhood dynamicity β_{ij} to identify the spatio-temporal outliers. Notice that we do not perform the multiplication of the contiguity matrices here.

Algorithm 2 STOUT-AM: Spatio-temporal outlier detection multiplication with binning

Require: Spatio-temporal matrix X_{smt}, size $N_1 \times N_2 \times N_3$, Spatial contiguity matrix SC, Temporal contiguity matrix TC, Temporal window size $twin$, Space difference threshold $thred_s$, Time difference threshold $thred_t$, Beta threshold $thred_{beta}$, Number of bins $iter$

1: **for** $i = 1 \rightarrow iter$ **do**
2: *Create data bins along the time mode and build spatio-temporal bin tensor*
3: $\mathcal{X}_i$=Tensor($X_{(s,m,(i \times twin+1:(i+1) \times twin))}$)
4: *Multiply spatial contiguity matrix and temporal contiguity matrix on the space and time mode respectively*
5: $\mathcal{X}_{newi}=\mathcal{X}_i \times_1 SC \times_3 TC$
6: *Tensor decomposition*
7: $(\lambda_{base}, S_{base}, M_{base}, T_{base})$=CP($\mathcal{X}_{base}$)
8: $(\lambda_i, S_i, M_i, T_i)$=CP($\mathcal{X}_i$)
9: *Calculate the difference between base tensor and current tensor*
10: $diff_s = \lambda_{base} \times S_{base} - \lambda_i \times S_i$
11: $diff_t = \lambda_{base} \times T_{base} - \lambda_i \times T_i$
12: *Calculate the Neighborhood Dynamicity* β_{ij}
13: **if** $diff_{s_i} \geq thred_s$ **then**
14: **if** $diff_{t_j} \geq thred_t$ **then**
15: Calculate $beta_{imj}$
16: **if** abs($beta_{imj}$) $\geq thred_{beta}$ **then**
17: X_{imj} is the spatio-temporal outlier
18: **end if**
19: **end if**
20: **end if**
21: **end for**

In algorithm 2 STOUT-AM, line 2-4 generate bins along the time mode and build spatio-temporal bin tensors. Line 4-5 multiplies the spatio-temporal bin tensor with spatial contiguity matrix and temporal contiguity matrix on the space and time mode respectively. Line 6-8 applies CP tensor decomposition on the base tensor $\mathcal{X}_{base}$ and current bin tensor $\mathcal{X}_i$. We get $\lambda_{base}, \lambda_{current}$, factor matrices S_{base}, T_{base}, $S_{current}$ and $T_{current}$. Line 9-11 calculates the differences between base tensor and current tensor in order to narrow down the search space. Line 12-21 calculates the neighborhood dynamicity β_{ij} to identify the spatio-temporal outliers. Notice that we perform the multiplication of the contiguity matrices here after the binning process.

In the algorithm 3 STOUT-BM, line 1-2 builds spatio-temporal tensors. Line 3-4 multiplies the spatio-temporal tensor with spatial contiguity matrix and temporal contiguity matrix on the space and time mode respectively. Line 6-7 generates the bin tensors along the time mode. Line 8-10 applies CP tensor decomposition on the base tensor $\mathcal{X}_{base}$ and current bin tensor $\mathcal{X}_i$. We get $\lambda_{base}, \lambda_{current}$, factor matrices $S_{base}, T_{base}, S_{current}$ and $T_{current}$. Line 11-13 calculates the differences between base tensor and current tensor in order to narrow down the search space. Line 14-23 calculates the neighborhood dynamicity β_{ij} to identify the spatio-temporal outliers. Notice that we perform the multiplication of the contiguity matrices here before the binning process.

4. EXPERIMENTS AND RESULTS

Algorithm 3 STOUT-BM: Spatio-temporal outlier detection multiplication before binning

Require: Spatio-temporal matrix X_{smt}, size $N_1 \times N_2 \times N_3$, Temporal window size $twin$, Space difference threshold $thred_s$, Time difference threshold $thred_t$, Beta threshold $thred_{beta}$, Number of bins $iter$

1: *Build spatio-temporal tensor*
2: $\mathcal{X}$=Tensor(X_{smt})
3: *Multiply spatial contiguity matrix and temporal contiguity matrix on the space and time mode respectively*
4: $\mathcal{X}_{new}=\mathcal{X} \times_1 SC \times_3 TC$
5: **for** $i = 1 \rightarrow iter$ **do**
6: *Create data bins along the time mode*
7: $\mathcal{X}_{newi}=\mathcal{X}_{new(s,m,(i \times twin+1:(i+1) \times twin))}$)
8: *Tensor decomposition*
9: $(\lambda_{base}, S_{base}, M_{base}, T_{base})$=CP($\mathcal{X}_{newbase}$)
10: $(\lambda_i, S_i, M_i, T_i)$=CP($\mathcal{X}_{newi}$)
11: *Calculate the differences between base tensor and current tensor*
12: $diff_s = \lambda_{base} \times S_{base} - \lambda_i \times S_i$
13: $diff_t = \lambda_{base} \times T_{base} - \lambda_i \times T_i$
14: *Calculate the Neighborhood Dynamicity* β_{ij}
15: **if** $diff_{s_i} \geq thred_s$ **then**
16: **if** $diff_{t_j} \geq thred_t$ **then**
17: Calculate $beta_{imj}$
18: **if** abs($beta_{imj}$) $\geq thred_{beta}$ **then**
19: X_{imj} is the spatio-temporal outlier
20: **end if**
21: **end if**
22: **end if**
23: **end for**

In this section, we present the experimental evaluation of our approach. The experiments were conducted on a machine with 3.4GHz Intel Core, 16GB 64-bit operating system, and running Windows 7 professional. All algorithms were implemented in Matlab using Matlab Tensor Toolbox [9] Version 2.4.

4.1 Dataset

We synthesized the dataset from existing spatial datasets. Data was retrieved from the TAO project data delivery website[6]. We downloaded high resolution data (10 minute average) for the entire year of 2009. This consisted of data from 53 sensors, 9 of which were missing an extensive number of time periods, and 43 had a full record for the year and had no missing data values. Therefore, we used the 43 sensors that had a full records for our experiments. Then we replicated the dataset to form 86 spatial locations with modified spatial contiguity matrix and each spatial location with a full year of readings. We introduced two types of outliers, namely point based outliers and window based outliers in the dataset. For the point based outliers, we randomly injected 5% of the spatio-temporal dataset as outliers with extreme values. For the window based outliers, we randomly selected the time period as a target to inject the entire window with extreme values for every spatial location.

4.2 Experiment Results

In the experiments, we validate the results from our three algorithms (STOUT-NoM, STOUT-AM and STOUT-BM) based on the synthetic dataset we generated in terms of accuracy, precision and recall. In addition we also present comparisons with the Dynamic Tensor Analysis (DTA) algorithm [30]. When comparing the algorithms for point based

outliers in Figure 4, 5, 6, the algorithm 1 (STOUT_NoM) consistently works well for point based outlier detection. However, we can also see that STOUT_AM also performs well and does better in terms of recall.

When we consider window based outliers the binning based variations work better than any of the algorithms. However, we notice in Figures 7, 8, 9, the algorithm 2 (STOUT-AM) works better for window based outlier detection consistently in terms of accuracy, precision and recall. This was our expectation since it is intuitive that as the data comes in the spatial and temporal contiguity changes. Therefore, performing the multiplication after the binning provides better results as this truly captures the changing neighborhoods and also encompasses space and time proximities. In all the comparisons we see that our proposed algorithms consistently perform better than DTA [30] in both window based and point based outlier detection.

Further, we test if the bin size plays an important role in the process by testing our algorithms with different bin sizes. Here bin size refers to the number of time periods per bin and the number of spatial locations do not change. In each iteration we utilize the same size of tensors in terms of spatial locations to identify the outliers. The bin size is the major factor determining the size of the tensor in our algorithms, which also is a factor when it comes to computational cost. We can see in Figure 10 the bin size does not impact accuracy of the outlier detection results substantially. However, we do see a slight drop in the accuracy with the increase in the bin size. We also notice some slight improvement in the recall but a drop in the precision as expected when the bin size increases. Thus, it is intuitive to select bins with somewhat smaller number of time periods. As we increase the number of time periods we loose the spatial and temporal neighborhoods which may lead to the drop in precision.

Finally we also compare the runtime of the various algorithms. We can see in Figure 11 that our algorithms do not cost more than DTA in terms of run time to achieve better results in terms of accuracy, precision and recall.

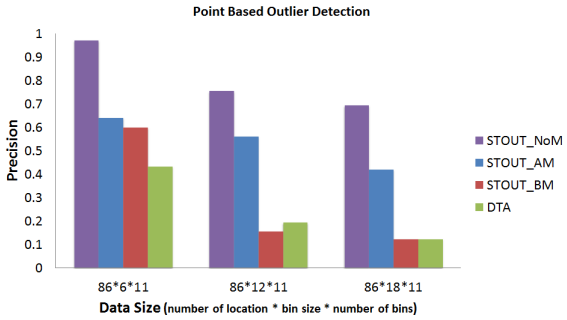


Figure 5: Point based outlier detection for all three algorithms - Precision

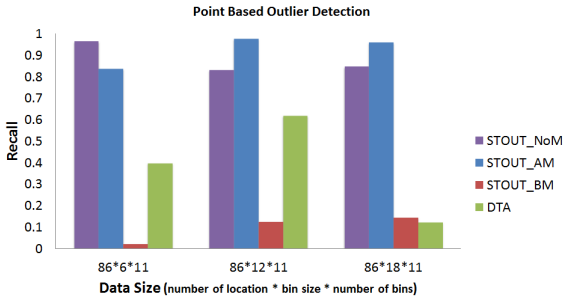


Figure 6: Point based outlier detection for all three algorithms - Recall

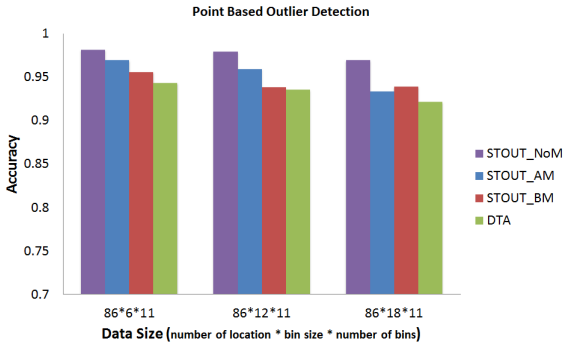


Figure 4: Point based outlier detection for all three algorithms - Accuracy

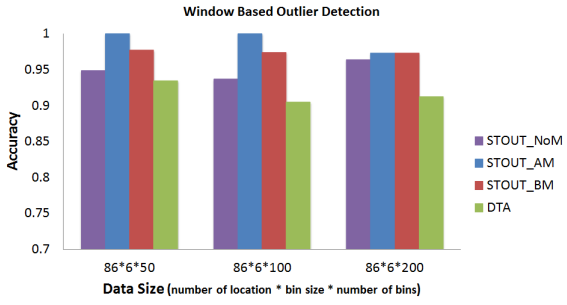


Figure 7: Window based outlier detection for all three algorithms - Accuracy

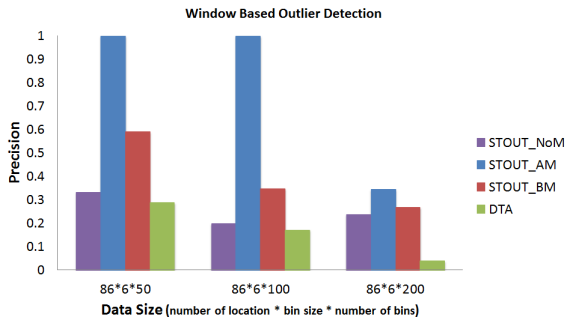


Figure 8: Window based outlier detection for all three algorithms - Precision

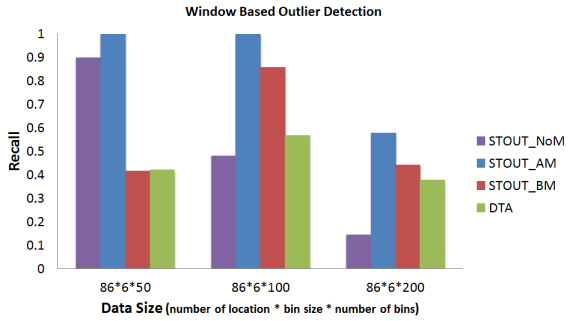


Figure 9: Window based outlier detection for all three algorithms - Recall

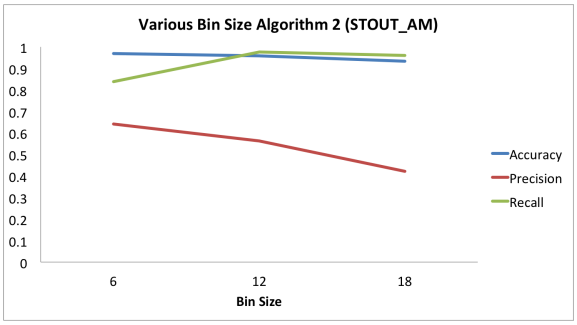


Figure 10: Various bin size for Algorithm 2 (STOUT-AM)

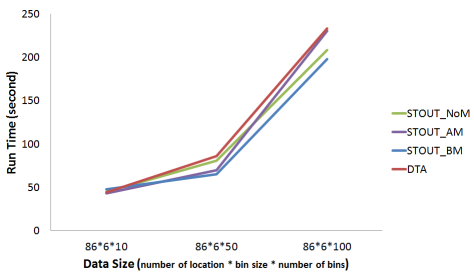


Figure 11: Run time

Data Size	memory
86*6*10	1,998568
86*6*50	2,001,796
86*6*100	2,002,052

Table 3: Memory cost

We also test our algorithms by bin on the spatial mode instead of temporal mode with point based outliers. The results in Figure 12, 13 and 14 show that our Algorithm 1 (STOUT-NoM) and Algorithm 2 (STOUT-AM) continually works better on point based outlier detection in terms of accuracy, precision and recall. We can see in Figure 15 that our algorithms do not cost more than DTA in terms of run time to achieve better results in terms of accuracy, precision and recall. It proves that our algorithms can be easily extended to the case where more number of spatial objects than the number of time readings.

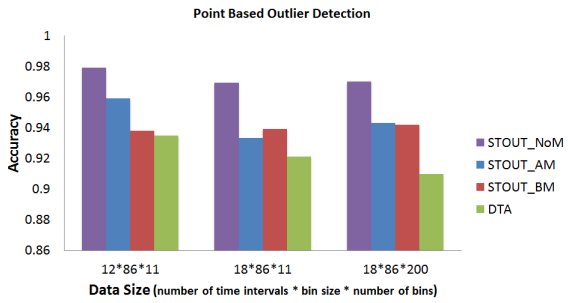


Figure 12: Point based outlier detection for all three algorithms (STOUT-NoM, STOUT-AM, STOUT-BM) by binning on the spatial mode - Accuracy

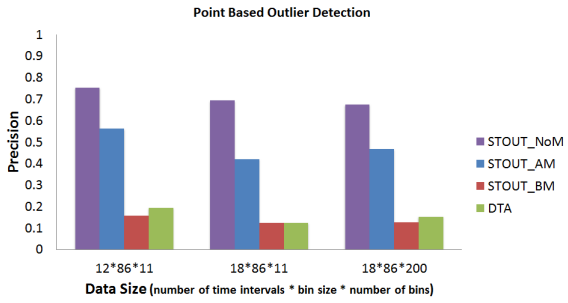


Figure 13: Point based outlier detection for all three algorithms (STOUT-NoM, STOUT-AM, STOUT-BM) by binning on the spatial mode - Precision

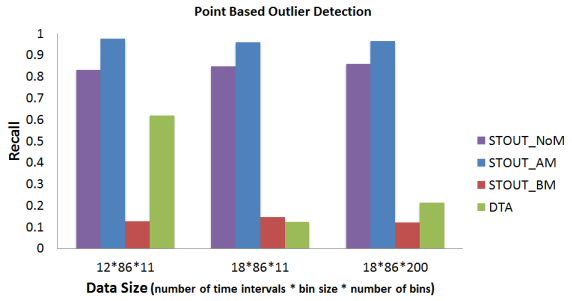


Figure 14: Point based outlier detection for all three algorithms (STOUT-NoM, STOUT-AM, STOUT-BM) by binning on the spatial mode - Recall

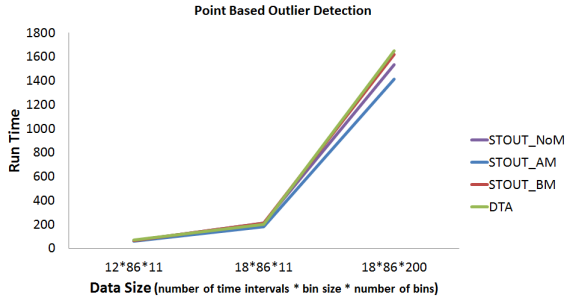


Figure 15: Point based outlier detection for all three algorithms (STOUT-NoM, STOUT-AM, STOUT-BM) by binning on the spatial mode - Run Time

5. CONCLUSION

In this paper we propose a class of algorithms for detecting Spatio-Temporal Outliers using Tensors. The major contribution of our approach is that we accommodate the spatial and temporal proximity information simultaneously for detecting different types of outliers namely point based and window based outliers. Existing techniques for discovering anomalies in spatio-temporal data may find the spatial outliers first and then identify the spatio-temporal anomalies from the data of that specific spatial location. Alternatively, some approaches may discover anomalous time periods and then discover the unusual spatial location. This may lead to identifying incorrect spatio-temporal outliers or missing important spatio-temporal phenomena due to the elimination of information after each step. In this paper, we presented our novel approach addressing the key limitation of existing spatio-temporal outlier detection methods by using an efficient tensor based model that supports complex relationships in spatio-temporal data to detect outliers by looking at space and time simultaneously as well as handling the scalability issue when it comes to manipulating large datasets. We presented a class of algorithms to discover different types of spatio-temporal outliers namely point based and window based outliers. We discussed detailed experimental results for each of the algorithms proposed and also presented comparative results with a state of the art approach.

In our future work we plan to extend our approach to realtime streaming data. We also propose to identify evolving patterns over space and time to discover interesting phenomena in large spatio-temporal datasets. Our current framework can be adapted to study how patterns evolve in dynamic data streams. This is very useful in studying moving phenomena such as outbreaks which evolve over time and follow specific trajectories, shifting of temperature zones over time, movement of chemical toxins in water or air to name a few.

6. REFERENCES

[1] Center for advanced transportation technology laboratory (catt lab) at university of maryland, regional integrated transportation information system, pi:michael l. pack. `https://www.ritis.org/index.php`.

[2] Colorado department of transportation online transportation information system. `http://dtdapps.coloradodot.info/otis`.

[3] Massachusetts department of transportation highway division. `http://www.massdot.state.ma.us/highway/Main.aspx`.

[4] State of new jersey department of transportation roadway information and traffic monitoring system program. `http://www.state.nj.us/transportation/refdata/roadway/traffic_counts/`.

[5] Traffic data branch at california department of transportation. `http://traffic-counts.dot.ca.gov/`.

[6] Noaa, tropical atmosphere ocean project, 2000.

[7] E. Acar, S. A. Çamtepe, M. S. Krishnamoorthy, and B. Yener. Modeling and multiway analysis of chatroom tensors. In *IEEE International Conference on Intelligence and Security Informatics,ISI*, pages 256–268, Atlanta, GA, 2005.

[8] N. R. Adam, V. P. Janeja, and V. Atluri. Neighborhood based detection of anomalies in high dimensional spatio-temporal sensor datasets. In *Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, pages 576–583, New York, NY, USA, 2004. ACM.

[9] B. W. Bader and T. G. Kolda. Matlab tensor toolbox version 2.4. `http://csmr.ca.sandia.gov/~tgkolda/TensorToolbox`, March 2010.

[10] V. Barnett and T. Lewis. Outliers in statistical data. 1994.

[11] T. Cheng and Z. Li. A hybrid approach to detect spatial-temporal outliers. In *Proc. 12th Int. Conf. on Geoinformatics Geospatial Information Research: Bridging the Pacific and Atlantic*, June 2004.

[12] T. Cheng and Z. Li. A multiscale approach for spatio-temporal outlier detection. *Transactions in GIS*, 10(2):253–263, 2006.

[13] M. Das. Spatio-temporal anomaly detection. 2009.

[14] M. Davy, F. Desobry, A. Gretton, and C. Doncarli. An online support vector machine for abnormal events detection. *Signal Process.*, 86(8):2009–2025, Aug. 2006.

[15] A. K. Derya Birant. Spatio-temporal outlier detection in large databases. *Journal of Computing and Information Technology*, 4:291–297, apr 2006.

[16] D. M. Dunlavy, T. G. Kolda, and W. P. Kegelmeyer. Multilinear algebra for analyzing data with multiple

linkages. Technical Report SAND2006-2079, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, April 2006.

[17] H. Huang, C. Ding, D. Luo, and T. Li. Simultaneous tensor subspace selection and clustering: the equivalence of high order svd and k-means clustering. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 327–335, New York, NY, USA, 2008. ACM.

[18] L. Huang, M. Kulldorff, and D. Gregorio. A spatial scan statistic for survival data. *Biometrics*, 63(1):109–118, 2007.

[19] R. Jäschke, L. Marinho, A. Hotho, S.-T. Lars, and S. Gerd. Tag recommendations in social bookmarking systems. *AI Commun.*, 21:231–247, December 2008.

[20] Y. Jin, J. Dai, and C.-T. Lu. Spatial-temporal data mining in traffic incident detection. In *SIAM Conference on Data Mining, Workshop on Spatial Data Mining*, april 2006.

[21] T. Kolda and B. Bader. Tensor decompositions and applications. *SIAM REVIEW*, 51(3):455–500, 2009.

[22] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. Technical Report SAND2007-6702, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, November 2007.

[23] M. Kulldorff. A spatial scan statistic. *Communications in Statistics-Theory and Methods*, 26(6):1481–1496, 1997.

[24] N. Liu, B. Zhang, J. Yan, Z. Chen, W. Liu, F. Bai, and L. Chien. Text representation: From vector to tensor. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, pages 725–728, Washington, DC, USA, 2005. IEEE Computer Society.

[25] D. B. Neill, A. W. Moore, M. Sabhnani, and K. Daniel. Detection of emerging space-time clusters. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 218–227, New York, NY, USA, 2005. ACM.

[26] Rendle, Steffen, and S.-T. Lars. Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 81–90, New York, NY, USA, 2010. ACM.

[27] S. Rendle, L. Balby Marinho, A. Nanopoulos, and S.-T. Lars. Learning optimal ranking with tensor factorization for tag recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 727–736, New York, NY, USA, 2009. ACM.

[28] T. M. Selee, T. G. Kolda, W. P. Kegelmeyer, and J. D. Griffin. Extracting clusters from large datasets with multiple similarity measures using IMSCAND. In M. L. Parks and S. S. Collis, editors, *CSRI Summer Proceedings 2007, Technical Report SAND2007-7977, Sandia National Laboratories, Albuquerque, NM and Livermore, CA*, pages 87–103, December 2007.

[29] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 187–198. VLDB Endowment, 2006.

[30] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: Dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 374–383, New York, NY, USA, 2006. ACM.

[31] Y. Sun, K. Xie, X. Ma, X. Jin, W. Pu, and X. Gao. Detecting spatio-temporal outliers in climate dataset: A method study. *INTERNATIONAL GEOSCIENCE AND REMOTE SENSING SYMPOSIUM*, 2:760–763, Nov 2005.

[32] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *PSYCHOMETRIKA*, 31(3):279–311, 1966.

[33] X. R. Wang, J. T. Lizier, O. Obst, M. Prokopenko, and P. Wang. Spatiotemporal anomaly detection in gas monitoring sensor networks. In *Proceedings of the 5th European conference on Wireless sensor networks*, EWSN'08, pages 90–105, Berlin, Heidelberg, 2008. Springer-Verlag.

[34] X. Yang and L. Latecki. Affinity learning on a tensor product graph with applications to shape and image retrieval. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2369 –2376, June 2011.

An Ensemble Approach for Event Detection and Characterization in Dynamic Graphs

Shebuti Rayana
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794-4400, USA
srayana@cs.stonybrook.edu

Leman Akoglu
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794-4400, USA
leman@cs.stonybrook.edu

ABSTRACT

Event detection in datasets represented by dynamic graphs is an important task for its applications in a variety of domains, such as cyber security, online and telecommunications, fault and fraud detection, etc. Despite recent advances in this area, there does not exist a single winning algorithm known to work well across different datasets. In fact, designing a single method that is effective on a wide range of datasets is a challenging task. In this work, we propose an *ensemble* approach for event detection and characterization of dynamic graphs. Our ensemble leverages three different base detection techniques, the results of which are systematically combined to get a final outcome. What is more, we characterize the events; by identifying the specific entities, i.e. nodes and edges, that are most responsible for the detected changes. Our ensemble employs a robust rank aggregation strategy to order both the time points as well as the entities by the magnitude of their anomalousness, which as a result yields a superior ranking compared to the base techniques, thanks to its voting mechanism. Experiments performed on both simulated (network traffic flow data with ground truth) and real data (New York Times news corpus) show that our proposed ensemble successfully identifies the important change points in which a given dynamic graph goes through notable state changes, and reveals the key entities that instantiate these changes.

1. INTRODUCTION

Anomaly detection in datasets represented by dynamic graphs has received much attention in recent years because of its wide range of applications in intrusion detection in cyber networks [28], fraud detection (insurance fraud, credit card fraud, auction fraud etc.) [2, 6], fault detection in medical claims, engineering systems [9], sensor networks and many more domains. In time series data, events which deviate from the normal behavior are anomalous. In dynamic graph analysis, the whole graph is observed in a sequence of discrete time ticks. Each of these time ticks represents a partial connectivity of the whole graph. To detect the anomalies one has to find the change points in the time series (known as event detection) and also the entities (nodes/edges) responsible for those changes (known as characterization).

Despite the availability of enormous amount of data from different domains, spotting the events of interest is challenging as they are quite rare. For example, the frequency of cyber attacks is very low compared to the whole network flow volume. The number of fraudulent transactions are rare compared to normal transactions in a financial organization. Although these events of interest occur infrequently their importance is very high compared to other events.

A lot of research has been done on event detection and characterization in different research communities. Both supervised and unsupervised learning techniques are used. The limitations of supervised methods are that they require labeled data which is not always available for real-world data, and cannot detect novel events that have never been observed previously. On the other hand, unsupervised methods do not require labeled data, they detect suspicious events (change points) or entities if the behavior deviates from the normal behavior in time [17]. The success of these algorithms depends on the metric upon which the change points are detected and also the characteristics of the data being used. For example, some approaches use a distance metric between consecutive graph pairs in a time series to find events [4, 29]. In event detection and characterization it is also important to find the anomalousness of rare events/entities and rank them accordingly. This ranking is often desirable over the traditional techniques which gives binary decisions (anomaly vs non-anomaly). There exist several outlier ranking approaches for graph data [14, 27, 15], however most are for static settings. Very few ranking approaches are available for dynamic graph based anomaly detection [13, 30]. Despite the fact that there exist many algorithms on event detection in dynamic graphs, there is no single winning algorithm known to work well across different datasets. In fact, designing a single approach robust to a wide range of datasets is a challenging task.

In this paper, we propose a unified framework for event detection and characterization in dynamic graphs by designing an *ensemble* approach that systematically combines results from different event detection algorithms. Our ensemble utilizes three detectors, and can accommodate other algorithms that provide detection and characterization. It is well known that ensemble approaches are effective in improving overall performance over the individual base techniques [1]. In particular, the motivation of using a set of algorithms is twofold.

ODD^2*'14,* August 24–27 2014, New York, NY, USA.

http://dx.doi.org/10.1145/2656269.2656274.

First, we aim to find consensus on the detected anomalies and second, we aim to identify those anomalies which are detected by one algorithm but not by the others. To the best of our knowledge, this is the first work in ensemble design for dynamic graph based event detection and characterization.

In our first algorithm we propose a new event detection technique which flags the change points in a time-varying graph at which many nodes deviate from their "normal" behavior. We use an "eigen-behavior" based technique which spots collective behavioral changes of nodes and edges to find suspicious events. At those events the nodes and edges which show the highest changes are also marked as anomalies. In detecting collective behavior changes, either many of the nodes/edges need to go through changes or a few nodes/edges need to change a lot. The second technique of the ensemble involves probabilistic time series anomaly detection. Here, we take a statistical approach to fit the time series of structural features of nodes and edges to several parametric distributions and select the best fit. The points are then flagged as anomalous based on their likelihoods. Finally, we employ a third algorithm called *SPIRIT* [23] which can find (hidden) trends in time series data, and dynamically detects change points by tracking the changes in the trends to spot potential anomalies. We remark that all three algorithms have two key properties: (1) event detection, and (2) characterization (finding the culprits).

In summary, our main contributions are as follows:

- **Ensemble Event Detection:** We propose an ensemble approach to detect change points in time series graph data. Our ensemble effectively merges results from different techniques to provide a robust outcome.
- **Consensus Rank Aggregation:** Our method uses both (i) rank-based and (ii) score-based aggregation approaches to build multiple consensus rankings. The results from different consensus approaches are then merged to obtain a final ranking that is better than most of the individual base algorithms.
- **Characterization:** Our approach could also attribute the changes to specific nodes and edges in the graph and hence characterize the detected changes.

Different from most earlier works on anomaly detection, we experiment with both simulated (network traffic flow data with ground truth) as well as real (New York Times news corpus data without ground truth) datasets to spot events and pinpoint anomalous agents. The network flow data has been carefully simulated in a realistic manner at NGAS R&T Space Park (`www.northropgrumman.com`) which mimics operations and anomalies that correspond to real-world events. We construct dynamic graphs based on the communications (edges) among different hosts (nodes) in the network. Our proposed approach is able to successfully unearth these events and the individual agents that initiated the events with high accuracy. Quantitative evaluation based on ground truth shows that our final ensemble yields better ranking of the events than most of the individual base algorithms. We also use published articles (Jan 2000 - July 2007) of New York Times (NYT) [26] where we construct dynamic graphs based on the named entities (nodes) being co-mentioned (edges) in the articles. Our experiments successfully reveal several big events during the time period of the NYT data, such as presidential elections of 2001, 9/11 terrorist attacks in World Trade Center, 2003 Columbia space shuttle disaster, etc., in addition to the key entities associated with these events.

The rest of the paper is organized as follows. Section 2 first discusses some important related works. Section 3 describes the work-flow of our ensemble and then its individual components. Section 4 gives our experimental setup and Section 5 presents the results. Finally, we provide a summary and discuss future directions for research.

2. RELATED WORK

Ensembles for unsupervised outlier detection is an emerging topic that has been neglected for a long time compared to ensembles for classification and clustering problems [7, 31, 11]. This is because in unsupervised settings no ground truth is available to evaluate the merit of the ensemble over its components. Moreover, unlike in clustering which falls under unsupervised learning, there exists no objective or fitness functions for outlier mining. Nevertheless, there have been several recent works on building outlier ensembles. In a recent position paper, Aggarwal [1] provided a categorization of the existing outlier ensemble approaches based on corresponding algorithmic strategies. According to Aggarwal, there have been traces of the very idea of combining results from different models in many earlier works but none of them are explicitly named as ensemble approaches.

In particular, ensemble approach is effectively used in high-dimensional outlier detection [1] where multiple subspaces of data are explored to detect outliers. The feature bagging approach [20] is the earliest work formalizing an outlier ensemble in high dimensional feature space, as outlier behavior of data points are often described by a subset of the dimensions. This approach uses the same base algorithm (LOF [3]) on different feature subsets and provides a rank based merging to create the final consensus. Feature bagging is better calibrated by [10, 18] which convert the outlier scores to probability estimates and use a score based merging. Our approach in contrast uses both rank based and score based aggregation to build the final ensemble.

In addition to using the variants of the same algorithm as ensemble components, it is possible to use different algorithms to build a heterogeneous ensemble. In [21] Nguyen et al. use both LOF [3] and LOCI [22] (density based outlier detectors) as components of their ensemble. While their approach is similar to ours, we build a heterogeneous ensemble for dynamic graphs in contrast to static clouds of points and capture the time evolving behavior of the data. Other existing outlier ensembles are also designed for clouds of multi-dimensional data points. To the best of our knowledge, ours is the first ensemble approach for event detection in time-evolving graph data.

3. PROPOSED METHOD

3.1 Ensemble Approach

We design an ensemble framework for event detection and characterization for dynamic graphs, as they appear in numerous scenarios including computer networks, trading networks, transaction networks, phone call and email communications. We propose to use three different event detection techniques to find change points in time series of graphs. Each of these algorithms operate in two phases (i) event de-

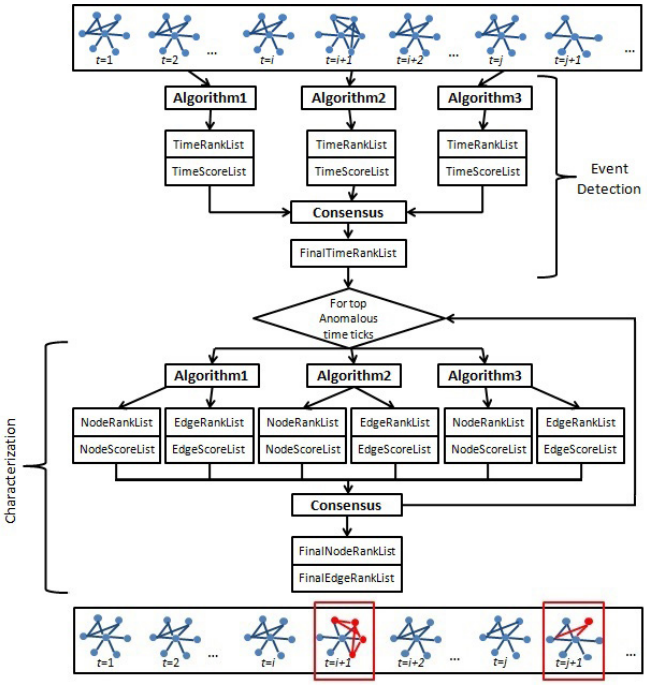


Figure 1: Proposed ensemble approach work-flow.

tection phase and (ii) characterization phase. In this section we give an overview of the work-flow of our ensemble approach, and defer the details of these individual techniques to Section 3.2. A flow chart of the framework is depicted in Figure 1, which is described next.

3.1.1 Event Detection

We use three event detection algorithms (abstracted as $Algorithm1, Algorithm2, Algorithm3$) to find anomalous time ticks (events) that show significant change points in the dynamic graph constructed from input time series data. Each algorithm has a specific measure to score the individual time ticks, depicting the amount of behavioral change of the graph. Different algorithms employ different measures. In general, the higher its score, the more anomalous a time tick is. Each algorithm provides a ranklist and a scorelist of time ticks in event detection phase. As such, we obtain three ranklists ranked from most to least anomalous and three scorelists for the time ticks as shown in Figure 1.

3.1.2 Characterization

For each time tick found to be anomalous for an algorithm we also identify the nodes and edges which are responsible for it. Again, each algorithm has a means to score individual nodes and edges which represents their amount of change in behavior. As such, for each anomalous time tick we create sorted ranked lists of nodes and similarly of edges as shown in Figure 1. Therefore, the characterization phase of each algorithm provides a ranklist and a scorelist of nodes/edges for each anomalous time tick. Note that some time ticks may be detected as anomalous by only a subset of the algorithms, in which case we create only as many lists.

3.1.3 Consensus Rank Aggregation

Our goal is to combine the results, i.e., ranked lists, of the ensemble algorithms to build a final rank ordering of (i) time ticks, and (ii) nodes/edges at top anomalous time ticks. In event detection phase, each of the base algorithm gives an anomalousness score to each time tick and then sort the list of those time ticks based on these scores to obtain the ranklist. Again in characterization phase, the anomalousness scores of nodes/edges are sorted to obtain the corresponding ranklists for top anomalous time ticks. To build the ensemble, we merge the results from different base algorithms to come up with the final result. We use both (i) rank based merging and (ii) score based merging to build the consensus.

Rank Based Merging: Different algorithms provide different scoring techniques of time ticks/nodes/edges according to their anomalousness which are not comparable. We use only the rankings of the time ticks to merge the ranklists obtained from the base algorithms in rank based merging. Here we use two techniques:

- **Inverse Rank Merging:** Each time tick/node/edge has a rank associated with it in a ranklist. We use inverse of these ranks $(1/R)$ (here, R is the rank of a time tick/node/edge in a ranklist) to calculate a score associated with individual time tick/node/edge. So, the top element in the ranklist has rank 1 and score 1, next element has rank 2 and score 0.5 and so on. We calculate the final score of individual time tick/node/edge by taking the average of these inverse rank scores from all the base algorithms. Finally, according to this final averaged rank scores we sort the time ticks/nodes/edges to obtain the final merged ranklist.

- **Kemeny Young:** *Kemeny Young* [16] method is a voting system which uses preferential ballot and pairwise comparison count to identify the most popular choice. We consider our algorithms as voters and the time ticks/nodes/edges as the candidates they vote for. The time tick/node/edge which comes at the top in a ranklist from a base algorithm is the most preferred candidate for that algorithm. The next one has less preference than the first one and so on. We calculate the scores in two steps, (i) creating a matrix that counts pair-wise voter preferences and (ii) calculating a score for each ranking position which is the sum of the pair-wise counts that apply to that ranking. We sort the time ticks/nodes/edges according to these scores to obtain the final merged ranklist.

Score based merging: As scores from different algorithms are not comparable we convert the scores to a well-calibrated probability estimate to make them comparable. In score based merging we use two techniques to convert output scores from the algorithms to probability estimates:

- **Unification:** The unification method [18] of anomalousness scores has three steps (i) Regularization, (ii) Normalization, and (iii) Gaussian Scaling:

  i) **Regularization:** We regularize the anomalousness scores from different base algorithms by transforming the interval of the scores from [base,∞) to [0,∞) in a way which does not change the rank order. If the range of score already is in the interval [0,∞) then we skip this step.

ii) **Normalization:** We use a linear transformation to transform interval of the scores to [0,1].

iii) **Scaling:** We use *Gaussian scaling* to convert the normalized scores to probability estimates which represent the probability of anomalousness of the time ticks/nodes/edges.

- **Mixture Modeling:** In the mixture modeling approach [10] we model the score distributions of the time ticks/nodes/edges as a mixture of *Exponential* and *Gaussian* probability distribution. The typical anomaly score distribution of the anomalous and normal classes show that the normal class follows an Exponential distribution and anomalous class follows a Gaussian distribution [10]. This suggests that a mixture model consisting of an Exponential and a Gaussian component may fit well to the anomaly score distributions. We use an *expectation maximization* (EM) algorithm to minimize the negative log likelihood function of the mixture model to estimate the parameters. We calculate the final posterior probability with *Bayes' rule* which represents the probability of anomalousness of the time ticks/nodes/edges.

After converting the scores from the base algorithms to probability estimates we use two techniques to merge them, (i) taking average of the probability scores and (i) taking maximum of the probability scores. Then we sort these scores to obtain the final merged ranklist.

These orderings from different consensus rank merging techniques essentially capture the agreement among the individual base algorithms and also includes the points where they disagree. Finally, we merge the ranklists from different consensus rank merging techniques using inverse rank merging approach to get the final ranklist of time ticks/nodes/edges. Here, we use inverse rank merging for its stable performance among all consensus approaches. The final ensemble scores $(1/R)$ associated with the final ranklist are fed to mixture modeling [10] to find a cut-off for anomaly detection. Section 5 contains our experimental results.

3.1.4 Feature Extraction

For event detection and specifically for characterization, we capture the behavior of the dynamic graph through the behavior of its entities, i.e. nodes and edges. As such, given the graph sequence $G_1, \ldots, G_t, \ldots$, we extract several graph-centric features for every node and edge for each G_i. This way, each node/edge is represented by a time series of its feature values. We experimented with several features. In particular, for nodes we extracted (1) *weighted degree*, (2) *degree*, and (3) number of *local triangles* associated with each node. For edges we extracted (1) *weight*, (2) number of *common neighbors*, and (3) number of *total neighbors* of the two end points of each edge.

Having outlined the general work-flow of our approach, we next describe the individual algorithms of the ensemble.

3.2 Ensemble Components

3.2.1 Eigen Behavior based Event Detection(EBED)

In this algorithm we use the time series features of nodes and edges of our input graph to detect the change points through eigen-behavior analysis.

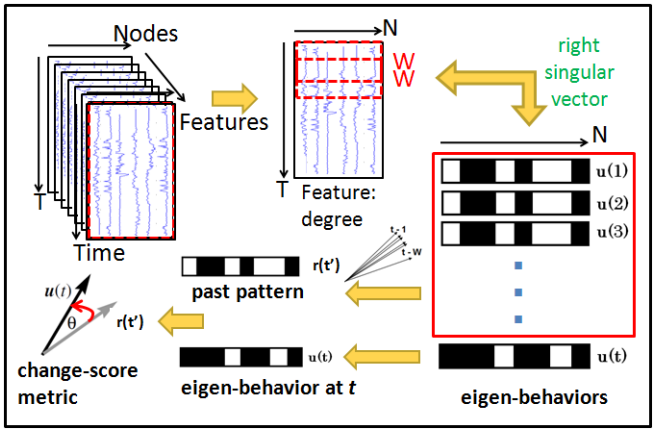


Figure 2: EBED change-point detection work-flow.

Figure 2 illustrates our proposed method, where T denotes the number of time ticks, N denotes the number of graph entities (nodes and edges), and F denotes the number of graph-centric features extracted. We take a $T \times N$ matrix for a selected feature F_i for nodes and edges (e.g., degree), define a window size W over the time series values of all entities, and compute the largest right singular vector of W. We treat this vector as the "eigen-behavior" $u(t)$ of the system during time window t, as the right singular vector of W is the same as the principal eigenvector of $W^T W$ which is essentially the time series similarity matrix of the graph entities. This vector is positive due to a famous theorem by Perron-Frobenius [24, 8], and lends itself to interpreting its entries as the "activity" or "behavior" of each graph entity. To capture the behavior of the graph over time, we slide the window down one time tick and recompute its eigen-behavior. We keep doing this until we reach the end of the time series.

Given new data i.e., a new graph at time t, we calculate the eigen-behavior of the new W. Using the eigen-behavior vectors $u(t')$ computed at previous time windows $t' < t$, we compute a typical eigen-behavior vector $r(t')$ by taking their arithmetic average. We compare the eigen-behavior $u(t)$ with the typical eigen-behavior $r(t')$ to quantify their similarity. As the anomalousness measure, we use what we call the Z *score* which is $Z = 1 - u^T r$. If $u(t)$ is the same as $r(t')$ then their dot product is 1, i.e., $Z = 0$ and if $u(t)$ is perpendicular to $r(t')$ then $Z = 1$. Thus, $Z \in [0, 1]$ and a significantly high value of Z indicates a change point.

For each detected anomalous time tick $\bar{t}$, we also pinpoint the specific nodes and edges which are responsible for the change. To do so, we compute the percentage of relative change $(u_i(\bar{t}) - r_i(\bar{t}'))$ for all nodes/edges. The higher the relative change, the more anomalous the node/edge.

3.2.2 Probabilistic Time Series Anomaly Detection (PTSAD)

To observe the lower-granularity behavior of nodes and edges we design a second algorithm which uses a probabilistic approach to detect anomalies in time series data. Here, we use several parametric distributions to fit individual time series of nodes and edges. We start with a well-known parametric distribution, *Poisson*, which is often used for fitting count data. While simple Poisson is not sufficient for sparse

series with many zeros, as often observed in the real-world. In fact most real-world count data is frequently characterized by overdispersion and excess number of zeros. Hence, our second choice is a *Zero-Inflated Poisson* (ZIP) [19] count model, as it provides a way of modeling excess zeros.

We further look for simpler models which fit the data with many zeros and employ the *hurdle models*. Rather than using a single complex distribution, hurdle models [25, 12] assume that the data is generated by two simple, separate processes; (i) the hurdle and (ii) the count components. The former process determines whether there exists activity or not at a certain time tick and in case of activity the count process determines the actual positive counts. One simple way to model the hurdle process is to assume an independent *Bernoulli*. In reality, there may be dependencies, where each activity influences the probability of subsequent activities. Thus, we also consider modeling the hurdle process with first order *Markov* models. For the count component, we use the *Zero-Truncated Poisson* (ZTP) [5] distribution.

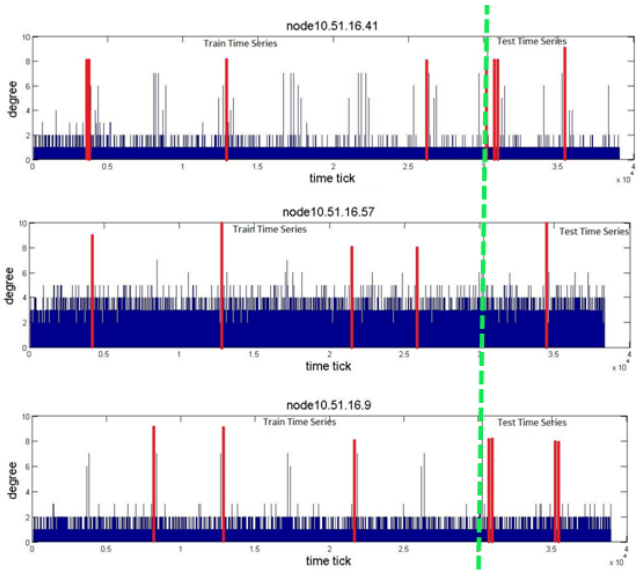


Figure 3: PTSAD approach for event detection in time series of three example nodes (rows) in network traffic flow data.

All in all, we fit four different (parametric) distributions to a training portion of each node/edge time series: Poisson, ZIP, Bernoulli+ZTP and Markov+ZTP. In order to select the best model for each time series, we employ a model selection procedure where we use several criteria including Akaike Information Criterion (AIC), log-likelihood of training data, Vuong's likelihood ratio test [32], and finally the log gain on test data. We further check the goodness of fit for those models with bootstrapping. Note that the best-fit model for each entity may be different. We also remark that our framework is flexible to incorporate additional distributions in the future, including non-parametric ones.

After fitting all the time series training data of individual nodes and edges we perform a single-sided test to compute a *p-value* (i.e., $P(X \geq x) = 1 - cdf_H(x) + pdf_H(x)$, where H is the model that fits best for a given entity). The lower the p-value, the more anomalous the time tick is for a given entity (to flag anomalies, we use the 0.05 threshold to capture significance at the 95 percentile). As it would be unrealistic to assume that the training data is anomaly-free, we first use the p-values to spot anomalous time ticks in the training data. We then remove those anomalous points and refit our models to build the representative ones. At this point, our system is ready to assess new coming data.

In Figure 3, we show example anomalous time ticks (red bars) as detected in the time series of three randomly selected nodes from our simulated network traffic flow data. The dashed bar separates test series from the training series.

PTSAD does not yield an aggregated result/score for the individual time ticks. It rather finds the anomalous time ticks for individual nodes/edges. Thus we aggregate the p-values of all nodes/edges along all the time ticks. We do so by taking the normalized sum of the p-values and invert them ($\in [0, 1]$) (so that higher is more anomalous) and then sort the time ticks accordingly. After detecting the most anomalous aggregated time ticks, we use the normalized logarithm of p-values ($\in [0, 1]$) of nodes/edges corresponding to those time ticks for characterization (again, the higher, the more anomalous).

3.2.3 SPIRIT

We use a third algorithm called *SPIRIT* for Streaming Pattern DIscoveRy in multIple Time-Series by Papadimitriou et al. [23]. This approach can incrementally capture correlations, discover trends, and dynamically detect change points in multiple regular time series data. Here, the intuition is to discover the underlying trend of a large number of numerical streams with a few hidden variables, where the hidden variables are the *projections* of the observed streams onto the principal direction vectors (eigenvectors). These discovered trends are exploited for detecting anomalies.

In a nutshell, the algorithm starts with a specific number of hidden variables which capture the general trends of the data. Whenever the main trends change, new hidden variables are introduced or several of existing ones are discarded to capture the change. This algorithm can further quantify the change in the behavior of graph entities for characterization through their *participation weights*, which are the entries of the principal direction vectors for the entities. For further details on the specific algorithm, we refer the reader to the original paper [23].

Finally, we remark that our ensemble is flexible to incorporate other approaches for event detection in multiple time series data.

4. EXPERIMENT SETUP

4.1 Dataset Description

4.1.1 Dataset 1: Challenge Network Traffic Flow

The simulated dataset that we use is a Cyber Challenge Network traffic flow data over 217 hours (≈9 days) and captures the interactions between hosts in the network. The dataset contains the to-from information of the interactions along with the time stamps. The time-aggregated graph representing the whole network has 125 nodes and 352 undirected edges connecting them. We choose the sample rate of 10 minutes for this dataset (with 1304 time ticks) as lower sample rates result in excess number of change points (potentially many false positives) due to large fluctuations in

the graph structure over small time periods, on the other hand, higher sample rates obscure the true positives (i.e., actual changes).

4.1.2 Dataset 2: NYT News Corpus

The real dataset that we use is the NYT hand-annotated (by human editors) news corpus [26] of seven and a half years of published articles(Jan 2000 - July 2007). We construct our dynamic graphs based on the named entities comentioned in an article. Therefore, the named entities are the nodes of the graph and if two entities are co-mentioned in an article there is an edge between them. Here the named entities are people, places, organizations, etc. This data has around 320000 entities to construct the time-aggregated graph. We analyze the data with weekly granularity.

4.2 Feature Selection

We extracted different features for nodes and edges with selected sample rates for both datasets. In particular, for nodes we extracted (1) degree, (2) weighted degree, and (3) number of local triangles. For undirected edges we extracted (1) weight, (2) number of common neighbors, and (3) number of total degree of the two end points. To avoid the skewing effect of large edge weights we selected the *degree* feature for the nodes to analyze the network for our final ensemble. We also selected the *total degree* feature for analyzing the time series of edges. For NYT data we used *weighted degree* feature for the nodes and *weights* feature for the edges to build the final ensemble. However the choice of these features can be application dependent.

4.3 Window Size Selection in EBED

For *EBED* we use a window size W of 4 for both datasets to calculate the principal eigen-vector. In social network or any human activity related network, a window size corresponding to 7 days generally represents the pattern of human behavior. As our experiment with network flow data was not portraying human activity, we experimented with different window sizes (4,5,6,7 etc.) to find the best size suitable for the network. Similarly, we experimented with different window sizes to come up with a suitable size for NYT data. In general, the larger the window gets, the more aggregated the results become.

4.4 Model Selection in PTSAD

For probabilistic time series analysis, we fit four different (parametric) distributions (Poisson, ZIP, Bernoulli+ZTP and Markov+ZTP) to a training portion of each node/edge time series. We used first 7 days of data from network flow dataset and first 100 weeks of data from NYT dataset for training and rest of the data from both datasets for testing. To pick the best fit out of the four for each time series, we employed several model selection procedures; including the straightforward log-likelihood of training data, the Akaike Information Criterion (AIC), Vuong's likelihood ratio test [32], and finally the log gain on test data. Each model selection criterion votes for a distribution (i.e., model) for the time series of each node/edge. The distribution which gets the highest vote is selected as a best fit for that particular node/edge. We briefly describe the four criteria as follows.

- *Log-likelihood* is higher for the better model. Model complexity (i.e. number of parameters) is not incorporated.
- *AIC* penalizes high model complexity. The lower the AIC, the better the model.
- *Vuong's test* finds the log-likelihood ratio of a given pair of models (say model 1 and model 2) and calculates a p-value for the sign of the ratio. If p-value is larger than 0.05, then the ratio does not give any decision (i.e., one model cannot be claimed better than the other). Otherwise, model 1 is claimed to be better than model 2 if the ratio is positive, and vice versa if the ratio is negative.
- *Log gain* is similar to Vuong's test, but instead assesses the log-likelihood ratio on the test data.

Table 1 shows the percentage pairwise agreements between the model selection criteria on the best model chosen for the nodes' time series of our network flow data. We also give the cardinality, which denotes the number of series for which both models can provide a fit (some fittings may fail due to model or data degeneracy, hence cardinality is less than the number of nodes). We notice that there exist high agreements among the various tests, and we use the majority vote to pick the final model for each node/edge.

Table 1: Agreement Among Model Selection Criteria in Challenge Network

cardinality + % of agreement	**Log likelihood**	**Vuong's Test**	**Log Gain**
AIC	card.: 121 78.51%	card.: 83 97.59%	card.: 96 82.29%
Log likelihood		card.: 83 100%	card.: 96 80.21%
Vuong's Test			card.:81 90.12%

4.5 Goodness of Fit Test

We further check the goodness of fit for the best model chosen using bootstrapping. For each node/edge, we take the best fit model and its fitted parameters P. We generate N (= 1000) simulated time series by bootstrapping from the best fit model. Next we estimate the parameters P^* from each of the N time series and form an empirical distribution of those fitted parameters. We obtain a 95% bootstrap confidence interval as the interval from the 2.5%-ile to 97.5%-ile of this bootstrap distribution. If P falls within the confidence interval, we conclude that the best fit is indeed a good fit for the nodes/edges time series.

5. EXPERIMENT RESULTS

In this section we present our experimental results of event detection and characterization in the simulated Cyber Challenge Network dataset and NYT news corpus dataset using our ensemble approach.

5.1 Dataset 1: Results

5.1.1 Event Detection

Recall from Section 3.2 the three methods used in our ensemble have different scoring techniques to rank the time ticks for change point detection:

- Z score is used for *EBED*.
- Inverse of the normalized sum of p-values of all nodes/ edges is used for *PTSAD*.
- Projection is used for *SPIRIT*.

As the scoring measures are completely different, it is not possible to directly combine these scores to build a ranking of the time ticks by anomalousness. Therefore, we make individual sorted ranked lists for each algorithm, and use both rank based merging and score based merging techniques described in Section 3.1.3 to come up with the consensus ranklists. Finally, we use inverse rank based merging to obtain the final ranklist.

Figure 4 shows the top ranking time points (with red bars) from all three techniques used in our ensemble to find anomalous events (from top to bottom: *EBED*, *PTSAD*, and *SPIRIT*). We notice that time tick 376 is detected as a change point by all the three algorithms and it is the highest ranked anomalous event. *EBED* and *PTSAD* also agree on time tick 1126, at which point *SPIRIT* also has a somewhat small spike. There also exist some events that only individual methods detect. Our ensemble also includes those in the final ordering.

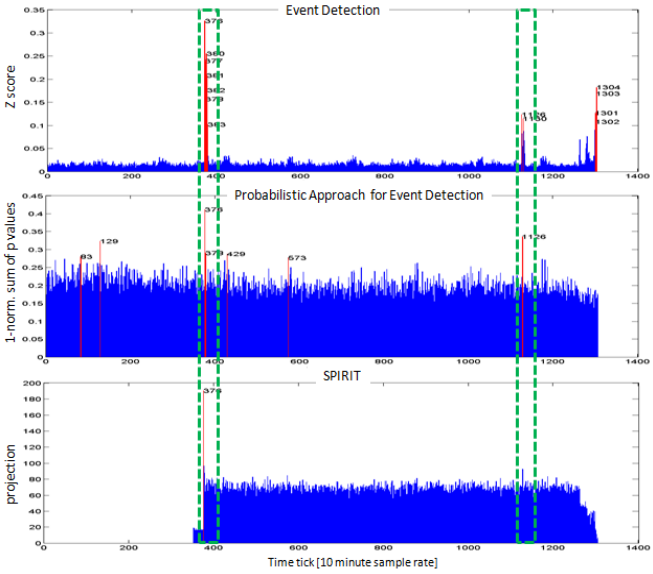


Figure 4: Change point detection in time series of challenge network for all three base algorithms in our ensemble approach. Red bars depict the top anomalous time ticks.

5.1.2 *Characterization*

After detecting anomalous events, we identify the nodes and edges which are responsible for those events. Again, different algorithms have different scoring techniques to attribute the change to specific nodes/edges. Following are the scores used by the algorithms in our ensemble.

- Relative change (% change of current eigen-behavior from previous one) is used for *EBED*.
- Normalized logarithm of p-value is used for *PTSAD*.
- Participation weight is used for *SPIRIT*.

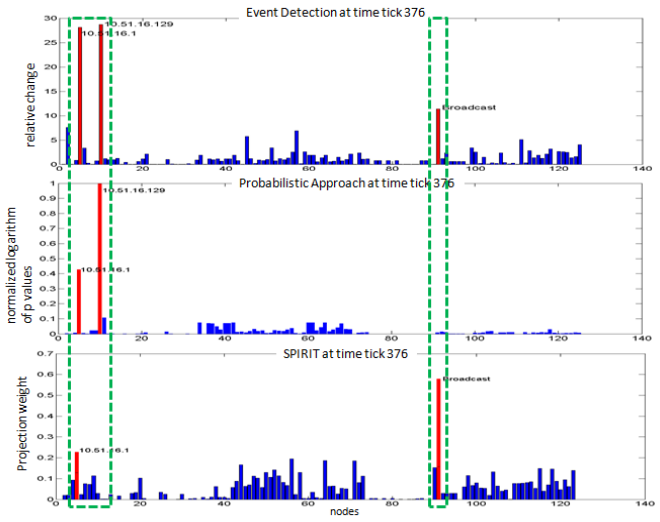


Figure 5: Anomalous nodes at time tick 376 from three algorithms. Agreements among different algorithms are marked with dashed green boxes.

Figure 5 shows the anomalous nodes for time tick 376 from all the three algorithms (from top to bottom: *EBED*, *PTSAD*, and *SPIRIT*). We realize that the results from different algorithms show considerable agreement. In particular, all the algorithms agree on IP '10.51.16.1' to be anomalous. In addition, we show the anomalous nodes for time tick 1126 in Figure 6 (as detected by *EBED* and *PTSAD*). They both agree on IP '10.50.10.14' to be anomalous, while *PTSAD* also flags several additional IPs as suspicious.

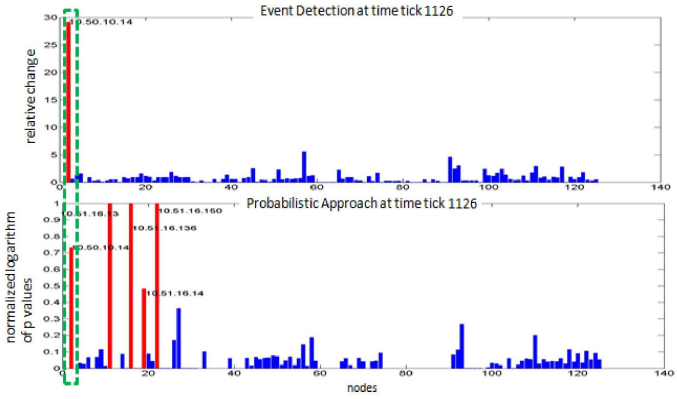


Figure 6: Anomalous nodes at time tick 1126 from *EBED* and *PTSAD*. The agreed-upon IP (node) is marked with a dashed green box.

Similarly, Figure 7 shows the anomalous edges for time tick 376 from all the algorithms. Again, we notice that the results from different algorithms have several agreements on the anomalous edges as well as individually identified ones. In particular, *EBED* and *SPIRIT* agree on edges '10.51.16.1'–'10.50.10.14' and '10.51.16.1'–'10.51.16.57'. In addition, *EBED* and *PTSAD* agree on edges '10.51.16.1'–'10.51.16.33' and '10.51.16.1'–'10.51.16.41'. Recall from Figure 5 that IP '10.51.16.1' is the top anomalous node detected

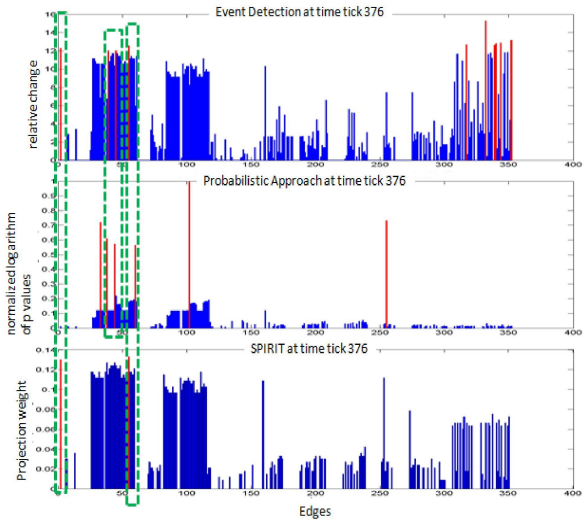


Figure 7: Anomalous edges at time tick 376 from three base algorithms. Agreements among the algorithms are marked with dashed green boxes.

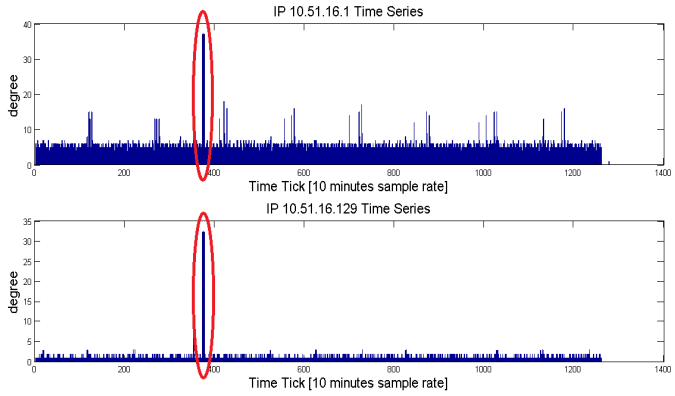


Figure 8: Time series of the top two anomalous nodes at time tick 376, when both IPs link to numerous other IPs.

by all the algorithms. The edge detection identifies the specific connections of those IPs that are most suspicious.

For further analysis we illustrate the time series of the top two node anomalies of time tick 376, i.e., IPs '10.51.16.1' and '10.51.16.129', in Figure 8. We notice that both nodes' connectivity increased significantly at the detected event. Figure 9 also depicts the graph visualization (from left to right) before, at, and after the event. From the figure we clearly see the change in the behavior of the two hosts; while being sparsely connected otherwise, they suddenly start communicating with many other hosts at time tick 376.

Essentially, the events and the specific agents identified as suspicious via our proposed approach correspond to a series of emergency event scenarios and their associated masterminds as simulated in our challenge network flow data. For data sharing agreement reasons, however, we are not allowed to elaborate on the specifics of the real-world events simulated in the data any further.

Table 2: Average Precision values for Event Detection [feature: Degree, sample rate: 10 minutes]

	Algorithms	AP
Base Algorithms	EBED	0.8333
	PTSAD	0.5722
	SPIRIT	0.7292
Consensus Rank Merging Algorithms	Inverse Rank	1.0000
	Kemeny Young	0.8095
	Unification(avg)	0.8056
	Unification(max)	0.7255
	Mixture Model(avg)	0.1684
	Mixture Model(max)	0.1684
	Final Ensemble	0.8667

Table 3: Average Precision values for Characterization [feature: Degree, sample rate: 10 minutes]

	Algorithms	Event: 376	Event: 1126
Base Algorithms	EBED	1.0000	1.0000
	PTSAD	1.0000	0.2500
	SPIRIT	0.3026	0.0213
Consensus Rank Merging Algorithms	Inverse Rank	1.0000	0.5000
	Kemeny Young	1.0000	0.2000
	Unification(avg)	1.0000	1.0000
	Unification(max)	0.8333	1.0000
	Mixture Model(avg)	1.0000	1.0000
	Mixture Model(max)	1.0000	1.0000
	Final Ensemble	1.0000	1.0000

5.1.3 *Quantitative Results*

The ground truth of our Network traffic flow data contains two major events (time tick 376 and 1126) and three associated nodes (IP '10.51.16.1', IP '10.50.10.14' and IP '10.51.16.129') responsible for those events. Table 2 and Table 3 present the *average precision* (AP) values of event detection and characterization phases, respectively, both for the individual base algorithms as well as the consensus approaches and the final ensemble (for sample rate 10 minutes, using feature *degree*). We note that the final ensemble for event detection has higher AP than all three individual ensemble components (in Table 2). Although the individual consensus approach of inverse rank based merging has higher AP than the final ensemble, the relation is reverse in characterization phase of time tick 1126 (in Table 3). Similarly, performance of mixture-modeling based consensus approach differs notably for detection versus characterization. These show that there is no single detector or consensus approach that consistently performs the best. On the other hand, the AP values for the final ensemble show that it is as good as the best individual base algorithm. Therefore, our ensemble framework proves to be an effective approach for event detection and characterization, which provides better or as good results as the best individual component.

5.2 Dataset 2: Results

5.2.1 *Event Detection*

Figure 10 shows the top ranking time points (with red bars) from all three techniques used in our ensemble with weighted degree feature to find anomalous events in NYT

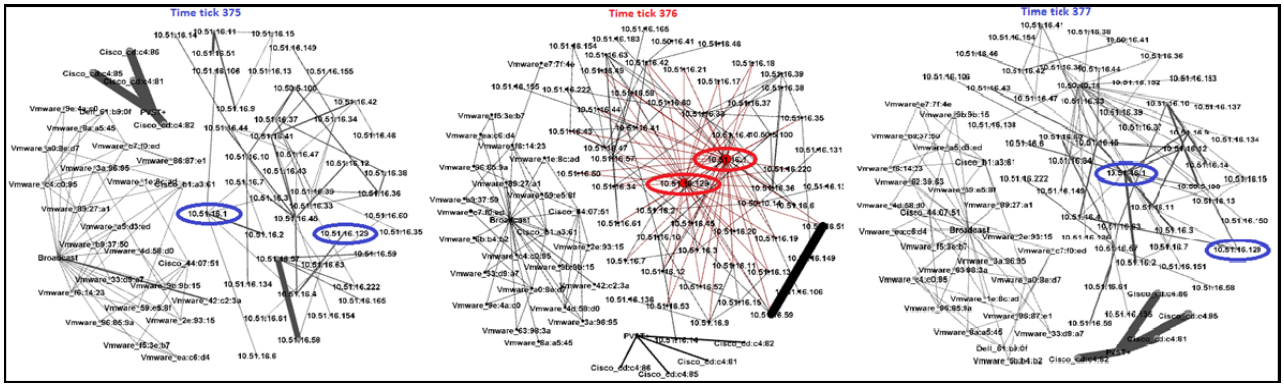


Figure 9: Cyber Challenge Network visualized (from left to right) before, at, and after the detected event at time tick 376. Notice the behavioral change of the two anomalous nodes detected in characterization step.

dataset(from top to bottom: *EBED*, *PTSAD*, and *SPIRIT*). We observe that time ticks 61, 62 are detected as change points by all the three algorithms. *EBED* and *SPIRIT* also agree on time ticks 90 and 162. The final ensemble dig out some important real-world events, such as, time ticks 61, 62 represent the events after the presidential election of USA in 2001 when George W. Bush was elected as president and was mentioned a lot in news articles, time tick 90 represents the 9/11 World Trade Center (WTC) terrorist attack in 2001 and time tick 162 represents the space shuttle Columbia (sent by NASA) disaster in 2003. There also exist some events that only individual methods detect. Our ensemble also includes those in the final ordering.

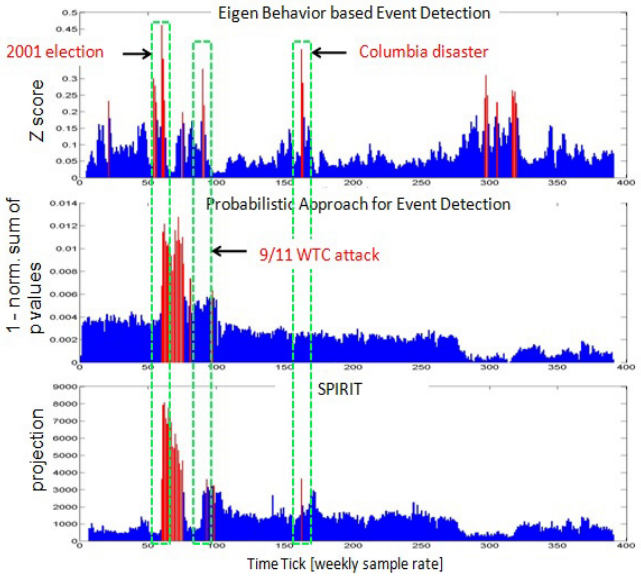


Figure 10: Change point detection in time series of NYT data for all three algorithms in our ensemble approach. Dashed green boxes depict three important real-world events ranked at the top.

5.2.2 *Characterization*

The characterization phase also finds entities associated or responsible for the detected events. For example, characterization of event 162 was able to find the seven astronauts of NASA who were killer in Columbia Space Shuttle disaster. Figure 11 shows the visualization of the change from week 161 to week 162 where suddenly the seven astronauts were highly co-mentioned with each other. We skip the detailed results of characterization phase for space limitation.

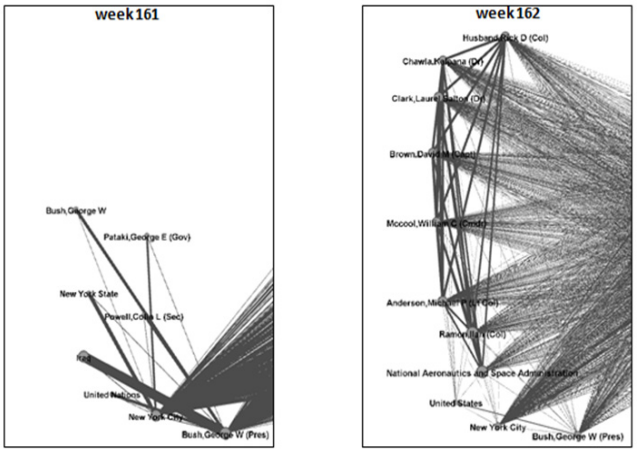


Figure 11: Characterization for 2003 Columbia Disaster. Visualization of graph structure change from time tick 161 to time tick 162 in NYT data, where a clique of NASA and seven astronauts emerges.

6. DISCUSSION

In this work, we proposed an *ensemble* approach for detecting and characterizing events in dynamically evolving graph data. This is the first work in ensemble design with dynamic graph based event detection algorithms as individual components. Our ensemble framework incorporates three different techniques; an eigen-behavior tracking based approach that we propose, a probabilistic model fitting approach, and a trend extraction based approach. Our method combines the findings in a unified manner to obtain a final consensus on the anomalousness of the time points. A key aspect of our proposed method is its focus on characterization; in addition to spotting suspicious change points, we also pinpoint the specific nodes and edges in the network that are most responsible for the changes. These are often considered as the specific entities associated with the detected events. Finally, our framework is amenable to incorporating additional techniques that exhibit event detection, ranking, and characterization properties.

We validated our proposed method on a simulated cyber network traffic dataset of hundreds of network hosts and their interactions. Our approach has successfully detected several change points in the cyber flows, as well as unearthing the hosts responsible for those changes with high accuracy. Additional experiments on a real NYT news corpus identified major events across the seven years span of the dataset and the key entities involved in those events. While our quantitative results showed that the ensemble approach outperforms the individual base algorithms and provides more accurate results, the qualitative results revealed events that agree with human interpretation.

While our experimental results have provided evidence that our proposed ensemble approach is successful for event detection and characterization with high performance in dynamic graphs both in simulated dataset with small ground truth and real dataset with large volume, further work is needed to fully analyze its performance on other large real datasets with more ground truth anomalies. Our future research will incorporate additional algorithms into the ensemble and investigate means to estimate detector accuracies and utilize weights proportional to these accuracies in combining their results. We will also extend our ensemble approach to outlier detection for high dimensional data points as well as other anomaly mining settings.

Acknowledgments

This material is based upon work supported by the Army Research Office under Contract No. W911NF-14-1-0029, an R&D grant from Northrop Grumman Aerospace Systems, an Office of Naval Research SBIR grant under Contract No. N00014-14-P-1155, and Stony Brook University Office of Vice President for Research. We thank Steve Cento and Tony Acosta for providing us with the network challenge dataset with ground truth. Any findings and conclusions expressed in this material are those of the author(s) and should not be interpreted as representing the official policies or views, either expressed or implied, of the funding parties.

7. REFERENCES

[1] C. C. Aggarwal. Outlier ensembles. In *ACM SIGKDD Explorations*, 2012.

[2] R. J. Bolton, D. J. Hand, and D. J. H. Statistical fraud detection: A review. In *Statistical Science*, 2002.

[3] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *ACM SIGMOD*, 2000.

[4] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. In *Pattern Recognition Letters*, pages 255–259, 1998.

[5] A. Cameron and P. Trivedi. *Regression Analysis of Count Data*. Cambridge Univ. Press, 1st edition, 1998.

[6] D. H. Chau, S. Pandit, and C. Faloutsos. Detecting fraudulent personalities in networks of online auntioneers. In *PKDD*, pages 103–114, 2006.

[7] T. G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, volume 1857, pages 1–15, 2000.

[8] G. Frobenius. Ueber matrizen aus nicht negativen elementen. In *Sitzungsber. Königl. Preuss. Akad. Wiss.*, 1912.

[9] R. Fujimaki, T. Yairi, and K. Machida. An approach to spacecraft anomaly detection problem using kernel feature space. In *PAKDD*, pages 401–410, 2005.

[10] J. Gao and P.-N. Tan. Converting output scores from outlier detection algorithms into probability estimates. In *ICDM*, 2006.

[11] J. Ghosh and A. Acharya. Cluster ensembles: Theory and applications. pages 551–570. 2013.

[12] N. A. Heard and D. J. Weston. Bayesian anomaly detection methods for social networks. In *The Annals of Applied Statistics*, pages 645–662, 2010.

[13] T. Ide and H. Kashima. Eigenspace-based anomaly detection in computer systems. In *ACM KDD*, 2004.

[14] P. Iglesias, E. Müller, F. Laforet, F. Keller, and K. Böhm. Statistical selection of congruent subspaces for outlier detection on attributed graphs. In *ICDM*, 2013.

[15] W. Jin, A. K. H. Tung, J. Han, and W. Wang. Ranking outliers using symmetric neighborhood relationship. In *PAKDD*, pages 577–593, 2006.

[16] J. Kemeny. Mathematics without numbers. In *Daedalus*, pages 577–591, 1959.

[17] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *VLDB*, 2004.

[18] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek. Interpreting and unifying outlier scores. In *SDM*, 2011.

[19] D. Lambert. Zero-inflated poisson regression with an application to defects in manufacturing. In *Technometrics*, pages 1–14, 1992.

[20] A. Lazarevic and V. Kumar. Feature bagging for outlier detection. In *KDD*, pages 157–166, 2005.

[21] H. V. Nguyen, H. H. Ang, and V. GopalKrishnan. Minning outliers with ensemble of heterogeneous detectors on random subspaces. In *DASFAA*, 2010.

[22] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using local correlation intergal. In *ICDE*, 2003.

[23] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708, 2005.

[24] O. Perron. Zur theorie der matrices. In *Mathematische Annalen*, 1907.

[25] M. D. Porter and G. White. Self-exciting hurdle models for terrorist actovity. In *The Annals of Applied Statistics*, pages 106–124, 2012.

[26] E. Sandhaus. The new york times annotated corpus ldc2008t19. In *Linguistic Data Consortium*, 2008.

[27] T. Seidl, E. Müller, I. Assent, and U. Steinhausen. Outlier detection and ranking based on subspace clustering. In *Uncertainty Manag. in Info. Sys.*, 2009.

[28] K. Sequeira and M. Zaki. Admit: Anomaly-based data mining for intrusions. In *ACM SIGKDD*, 2002.

[29] P. Shoubridge, M. Kraetzl, W. Wallis, and H. Bunke. Detection of abnormal change in a time series of graphs. *Interconnection Networks*, pages 85–101, 2002.

[30] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. Graphscope: Parameter-free mining of large time-evolving graphs. In *KDD*, 2007.

[31] G. Valentini and F. Masulli. Ensembles of learning machines. In *WIRN*, volume 2486, pages 3–22, 2002.

[32] Q. H. Vuong. Likelihood ratio tests for model selection and non-nested hypotheses. In *Econometrica*, 1989.

Learning Outlier Ensembles:
The Best of Both Worlds – Supervised and Unsupervised

Barbora Micenková[1,2]
barbora@cs.au.dk

Brian McWilliams[2]
brian.mcwilliams@inf.ethz.ch

Ira Assent[1]
ira@cs.au.dk

[1]Aarhus University
Aabogade 34
Aarhus, Denmark

[2]ETH Zürich
Universitätstrasse 6
Zürich, Switzerland

ABSTRACT

Years of research in unsupervised outlier detection have produced numerous algorithms to score data according to their exceptionality. However, the nature of outliers heavily depends on the application context and different algorithms are sensitive to outliers of different nature. This makes it very difficult to assess suitability of a particular algorithm without *a priori* knowledge. On the other hand, in many applications, some examples of outliers exist or can be obtained in addition to the vast amount of unlabeled data. Unfortunately, this extra knowledge cannot be simply incorporated into the existing unsupervised algorithms.

In this paper, we show how to use powerful machine learning approaches to combine labeled examples together with arbitrary unsupervised outlier scoring algorithms. We aim to get the best out of the two worlds—supervised and unsupervised. Our approach is also a viable solution to the recent problem of outlier ensemble selection.

Keywords

Outlier detection, outlier ensembles, semi-supervised outlier detection, feature construction

1. INTRODUCTION

Unsupervised outlier detection algorithms [2, 7] aim to reveal extraordinary data points in a data collection. Their original application is in pure data exploratory tasks (e.g., astrophysics, molecular biology) where almost no prior knowledge about the nature of outlierness is available and the goal is to find surprising patterns. Based on geometrical properties of the data (mostly distances and density), these algorithms assign a real-valued outlierness score to each data point thus enable a final outlier ranking.

In many applications, however, the semantics of outliers are known in advance, but not all the possible forms that they can take (e.g., in detection of fraud, intrusions, mislabeled data, measurements errors or faults). In these tasks, a small number of previously seen outliers is available in addition to a large number of unlabeled (mostly normal) data. In such a situation, unsupervised algorithms often perform poorly because there is no principled way how they can take advantage of these extra labels.

We present a new paradigm for outlier detection, *semi-supervised outlier detection*, that combines both the information from unlabeled data and supervision of some labeled data. Through a powerful learning technique, we aim at getting the best out of the two worlds—supervised and unsupervised. The strength of the proposed concept is in its universality because any existing unsupervised outlier scoring algorithm can be adapted, and, similarly, different machine learning approaches can be integrated. It also means that the concept opens a promising field of future research.

The key idea is to use the output scores of multiple unsupervised outlier detection algorithms as *transformed features* for learning with an extreme class imbalance. This transformation step is at the core of our solution. From the original attribute space, we move the problem to a transformed space where the dimensions are different types of exceptionality. In this exceptionality feature space, we employ an ensemble approach to learn feature weights and thus appropriately integrate the supervised and unsupervised information. In our initial setup, we use an ensemble of logistic regressors to learn the feature weights. Logistic regression is a convenient choice since (with an appropriate regularizer) it allows for a sparse solution where many weights can be drawn to zero. Furthermore, it outputs probabilities that can directly be used as outlier scores. This makes for an easily interpretable and verifiable result. Considering that the transformed features correspond to specific outlier detection algorithms with particular settings, the learnt feature weights can be interpreted as an outlier ensemble. We will further discuss the parallels to outlier ensembles and also the strengths and limitations of the approach (Sec. 4, 5).

The paper is organized as follows. We present our initial setup in Sec. 2 and preliminary results on two data sets in Sec. 3. After a brief description of the related work in Sec. 4, we devote an extended space to the discussion and future work in Sec. 5.

2. METHOD

Let $X \in \mathcal{X}$ be a set $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ of data points, and let $L = \{0, 1\}$ be a set of labels where 1 corresponds to an outlier and 0 to a normal data point. The number of outliers in X is much smaller than the number of normal data points. Let $l \ll n$ points in X be labeled outliers, the

ODD^2*'14,* August 24th, 2014, New York, NY, USA.

rest is unlabeled. For the sake of training, we assign label 0 (normal data class) to all unlabeled data points even if we know that some of them will be wrong. The goal is to predict labels of unlabeled and/or previously unseen data points. Since we are interested in ranking outliers according to the degree of their exceptionality, we require probabilities to be output together with the labels [9].

2.1 Logistic Regression with ℓ_1 Penalty

We instantiate our concept using logistic regression, which is a statistical classifier that models the outcome of a binary random variable, y. The probability of a data point belonging to class 1 is modeled as a linear function of variables (features) and a parameter vector, $\beta \in \mathbb{R}^d$ using the logistic function:

$$p(y = 1|x; \beta) = \frac{1}{1 + \exp(-\beta^\top x)} = \sigma(\beta^\top x). \quad (1)$$

The classifier predicts 1 if $\sigma(\beta^\top x) > 0.5$ and 0 otherwise.

To fit the parameters β based on a set of observations X and true labels y, the following cost function needs to be minimized:

$$J(\beta) = -\frac{1}{n}[\sum_{i=1}^{n} y_i \log \sigma(\beta^\top x_i) + (1 - y_i) \log(1 - \sigma(\beta^\top x_i))] + \frac{\lambda}{2n} \sum_{j=1}^{m} |\beta^j|. \quad (2)$$

The first term is the usual logistic loss, $f(\beta)$. The second term in Eq. (2) performs ℓ_1 regularization often referred to as Lasso [14]. It penalizes models with extreme parameter values and shrinks many of them to 0 which has the advantage here of removing features that do not contribute to the outlier scoring task.

Eq. (2) can be minimized using stochastic coordinate descent (SCD) methods [13]. Briefly, SCD picks features uniformly at random and updates the corresponding coordinate of β until convergence as follows

$$\beta^j = s_{4\lambda}(\beta^j - 4\nabla_j f(\beta)),$$

where $s_\lambda(z) = \text{sign}(z) \max(|z| - \lambda, 0)$ is the soft thresholding operator and $\nabla_j f(\beta)$ is the derivative of the loss with respect to the j^{th} coordinate of β.

2.2 Feature Transformation

Instead of applying the logistic regression on the original data, we extract new features. We transform $X \in \mathcal{X}$ into $\Phi(X)$ in the following way. Let $\Phi = \{\Phi_1, \ldots, \Phi_m\}$ be a set of outlier scoring functions where $\Phi_i : \mathcal{X} \to \mathbb{R}^n$. Each $\Phi_i \in \Phi$ is applied to X. If the original data matrix is $X = [x_1, \ldots, x_n]^\top$, the transformed data matrix is

$$\begin{bmatrix} \Phi_1(X) & \cdots & \Phi_m(X) \end{bmatrix} = \Phi(X). \quad (3)$$

Instead of the original data set, we now work with the transformed data set $\Phi(X)$ in the outlier score feature space.

To form the set of functions Φ, we may use any existing unsupervised outlier detection algorithm. Besides using distinct algorithms, we may also use the same algorithm under a perturbation, e.g., change of parameter settings, a distance metric, different subspaces etc. The goal is a setup where different aspects of outlierness are captured by the different scoring functions to span the transformed feature space.

2.3 Re-sampling by Bagging

A challenge for logistic regression lies in the class imbalance. To combat this issue, we propose to use a standard re-sampling method: bootstrap aggregating, also known as *bagging* [4]. Bagging produces multiple versions of a model by training on different bootstrap samples of the training set. Then, an aggregated model is acquired by averaging the outcomes of all versions (for the case of logistic regression it is output probabilities).

A bootstrap sample is a subset of data that is sampled uniformly with replacement. However, in our case we select approximately the same number of outliers as of (contaminated) normal data points. Bagging can help us achieve class balance through downsampling the majority class without losing much information. Since there is a minority of outliers, the same points will get selected multiple times while the normal data class will substantially differ across samples. This also is a reasonable setting considering the semi-supervised setup of the problem where the normal class is contaminated and thus its labels are unreliable.

An appealing property of ℓ_1-regularized logistic regression is the ability of directly performing feature selection by shrinking parameters. In practice, however, the regularization parameter must be carefully chosen. *Stability selection* [11] combines ℓ_1 penalized methods with bootstrap sampling by tracking the proportion of times a particular feature is selected across all of the subsamples. For a large enough number of bootstrap samples, this can be considered as the probability that a given feature belongs in the model.

3. EXPERIMENTS

We compare our approach on two data sets with two different baselines, reporting standard outlier detection evaluation measures: the ROC curve and the area under the ROC curve (AUC).

3.1 Competitors

We present results of two versions of our algorithm, `proposed` and `proposed+`. `proposed` only uses the transformed features while `proposed+` combines both the original and transformed features. For the first two baselines, we use the same training setup as for the proposed algorithms but we train merely on the original set of features. The difference is that `base1:orig` uses the same partially labeled training set as our method while `base2:sup` is trained on fully labeled data (it gets more information than our method and likely more than there would be available in practice). Strictly speaking, `base2:sup` is not a baseline, but we include it to demonstrate the strength of our approach.

As another baseline, `base3:ens`, we adopt a recent outlier ensemble algorithm from Schubert *et al.* [12]. It is an unsupervised approach that builds a binary target vector based on the rankings of the ensemble members and then greedily selects a subset of them to maximize weighted Pearson correlation to the target vector. For an appropriate comparison, we adapt their method such that we build the target vector from the partially labeled training set instead of their proposed heuristic to make the supervision available to this approach as well.

3.2 Data

For our experiments, we use two different data sets. The

synthetic `letter` data set is derived from the UCI letter recognition data set where letters of the alphabet are represented in 16 dimensions [3]. To get data suitable for outlier detection, we subsample data from 3 letters to form the normal class and randomly concatenate pairs of them so that their dimensionality doubles. To form the outlier class, we randomly select few instances of letters that are not in the normal class and concatenate them with instances from the normal class. The concatenation process is performed in order to make the detection much more challenging as each outlier will also show some normal attribute values. In total, we have 1500 normal data points and 100 outliers (6.25% outliers) in 32 dimensions.

The real-world `speech` data set consists of 3686 segments of English speech spoken with different accents.[1] The majority data corresponds to American accent and only 1.65% corresponds to one of seven other accents (these are referred to as outliers). The speech segments are represented by 400-dimensional so called *i-vectors* which are widely used state-of-the-art features for speaker and language recognition [8]. It is a subset of data described in [6].

We have made both data sets publicly available.[2]

3.3 Learning Setup

We split available data to a 60% training and 40% testing set. To simulate the semi-supervised scenario, we remove *half* of the outlier labels from the training set and consider them unlabeled data (which we treat as a contaminated normal class in our setup). For bagging, we construct 50 balanced samples from the training set, learn a logistic regressor on each of them and combine their outputs. At this point, no regularization has been applied for any method so that the results are more easily comparable.

For the sake of feature transformation we use a combination of established unsupervised outlier detection techniques: feature bagging [10] (selecting random subsets of features), k-NN outlier (compute sum of distances to k nearest neighbors) and LOF [5] (scores data based on local density). Precisely, it is feature bagging with LOF for the `letter` data set (50 random bags) and feature bagging with $k = 1$ and Canberra distance for the `speech` data (20 random bags). These settings are based on a coarse search for unsupervised algorithms that perform reasonably well on the training set. It is a mere starting point and alternatives should be investigated.

3.4 Results

In Fig. 1, we report the ROC curves for the `letter` data set. `proposed` and `proposed+` outperform the baselines (notice that they do especially well in the beginning of the ROC curve which is particularly important for real applications). We can see that they can even beat a fully supervised setup with original features (`base2:sup`) and that the most viable competitor is the ensemble of outlier detectors (`base3:ens`). Clearly, the outliers are better separable in the transformed than in the original domain. Fig. 2, on the other hand, shows the ROC curves for the `speech` data set. Here, the `proposed` approach performs comparably to the baselines (except for `base2:sup` which has access to all labels in the training set),

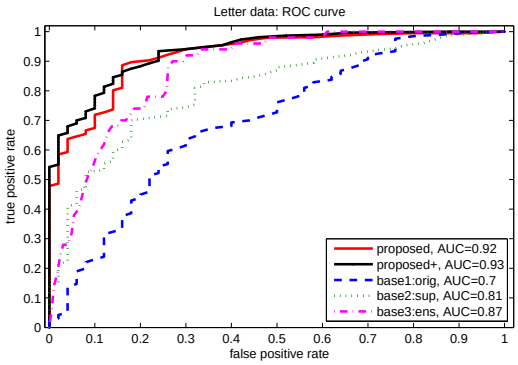


Figure 1: Letter data: ROC curves and the corresponding AUC values.

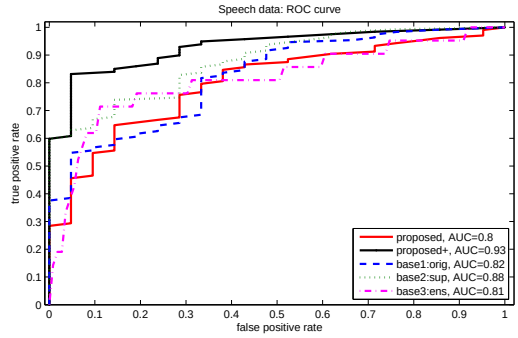


Figure 2: Speech data: ROC curves and the corresponding AUC values.

with only 20 features compared to the original 400. This suggests that both the original and the transformed domain are useful in detection of outliers and explains why `proposed+` that combines them has a superior performance to all other techniques with a great margin. The ROC curves are based on a single run; to show that there indeed is a trend, we report ROC AUC averaged over 20 runs with different train/test splits in Table 1. The proposed method with both original and transformed features outperforms all the other techniques, even the supervised setup. Undoubtedly, the performance improvement depends on the quality and diversity of the input pool of detectors that are used to construct the transformed features. Besides that, additional improvement is expected when applying regularization. For these preliminary experiments, we did not study these aspects in detail but the current results already show a great potential.

Table 1: Average ROC AUC

Method	Letter	Speech
proposed	0.926	0.783
proposed+	**0.942**	**0.875**
base1:orig	0.766	0.814
base2:sup	0.806	0.857
base3:ens	0.881	0.774

[1]The authors would like to thank to the Speech Processing Group at Brno University of Technology, Czech Republic, who provided us with the data.

[2]Download the data sets at http://goo.gl/mGg8ti?gdriveurl.

4. RELATED WORK

Outlier Ensembles

It has been shown that an appropriate combination of multiple algorithms (later called detectors) can increase outlier detection performance [12] which has recently triggered a wide interest in outlier ensembles [1, 16]. The problem of selecting (building/weighting) an ensemble is complex and hard to do in completely unsupervised settings. Open questions concern, e.g., the tradeoff between accuracy of the single detectors and their diversity, correlation among them or the method to combine their outputs [16]. The problem is magnified by the fact that outputs of different scoring algorithms are on different scales and often cannot be interpreted as outlier probabilities [9]. The approach outlined in this paper could be viewed as an ensemble selection technique where the few provided labels guide the selection process, giving an elegant solution to the above stated problems.

Class-Imbalance Learning

In addition, the proposed method complements machine learning literature on classification of extremely imbalanced data sets. Common approaches to class-imbalance learning such as sampling, bagging and boosting, one-class classification and cost-sensitive learning (references to be found e.g. in [2, 15]) can readily be complemented by the proposed scheme. Outputs of unsupervised outlier detectors can be interpreted as additional non-linear features to enhance classification in a similar spirit as kernels do. We expect that the proposed concept will be useful in situations where the rare class is not well clustered but more investigation must be carried out on this topic to confirm this hypothesis.

5. DISCUSSION AND FUTURE WORK

We have presented a new concept of semi-supervised outlier detection that is useful in applications where some outlier examples are available on top of a vast amount of unlabeled data and where new types of outliers might occur in the future. We believe that this scenario is realistic for both research and commercial applications. The idea combines unsupervised outlier detection with established machine learning techniques for classification in the transformed outlier score space. Initial experiments indicate that if we add outputs of unsupervised outlier detection algorithms as new features to the original training data, we can get a superior performance to both unsupervised and supervised techniques. However, numerous challenges to both outlier detection and machine learning researchers remain.

Some of the favourable properties of the concept are:

- any outlier scoring function can be used as a feature,
- a well-interpretable result as each feature corresponds to a specific outlier detection algorithm with particular parameter settings,
- ℓ_1-regularization can eliminate most of the features,
- output scores can directly be interpreted as outlier probabilities,

Using appropriate outlier detection algorithms and thus getting good transformed features is crucial for the performance and it is the same challenge that outlier ensemble theory faces [12]. Our limitation compared to classical supervised methods is that computing the transformed features is relatively slow. Since the same process needs to be applied at prediction time, we want to discard as many features as possible already in the training phase via regularization. This requires further investigation.

Many other new directions are open for future work. E.g., it is worth trying to apply this learning setup to select the best parameter settings for a single algorithm. It also seems to be a good framework for developing active learning strategies and interactive detection algorithms. Further, it should be clarified in which applications and machine learning setups the proposed technique is viable.

6. ACKNOWLEDGEMENT

Part of this work has been supported by the Danish Council for Independent Research—Technology and Production Sciences (FTP), grant 10-081972.

7. REFERENCES

[1] C. C. Aggarwal. Outlier ensembles: position paper. *SIGKDD Explorations*, 14(2), 2012.

[2] C. C. Aggarwal. *Outlier Analysis*. Springer, 2013.

[3] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[4] L. Breiman. Bagging predictors. *Mach. Learn.*, 24(2), 1996.

[5] M. M. Breunig, H.-P. Kriegel, R. Ng, and J. Sander. LOF: Identifying density-based local outliers. In *SIGMOD'00*, 2000.

[6] N. Brummer, S. Cumani, O. Glembek, M. Karafiát, P. Matějka, J. Pešán, O. Plchot, M. M. Soufifar, E. V. de, and J. Černocký. Description and analysis of the Brno276 system for LRE2011. In *Proc. of Odyssey 2012: The Speaker and Lang. Rec. Workshop*, 2012.

[7] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *CSUR*, 41(3), 2009.

[8] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet. Front-end factor analysis for speaker verification. *IEEE Trans. on Audio, Speech & Lang. Proc.*, 19(4), 2011.

[9] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek. Interpreting and unifying outlier scores. In *SDM'11*, 2011.

[10] A. Lazarevic and V. Kumar. Feature bagging for outlier detection. In *KDD'05*, 2005.

[11] N. Meinshausen and P. Bühlmann. Stability selection. *Journal of the Royal Stat. Society. Series B*, 72(4), 2010.

[12] E. Schubert, R. Wojdanowski, A. Zimek, and H.-P. Kriegel. On evaluation of outlier rankings and outlier scores. In *SDM'12*, 2012.

[13] S. Shalev-Shwartz and A. Tewari. Stochastic methods for l_1-regularized loss minimization. *The Journal of Machine Learning Research*, 12, 2011.

[14] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Stat. Society*, 1996.

[15] M. Wasikowski and X. wen Chen. Combating the small sample class imbalance problem using feature selection. *IEEE Trans. on Knowledge and Data Engineering*, 22(10), 2010.

[16] A. Zimek, R. J. G. B. Campello, and J. Sander. Ensembles for unsupervised outlier detection: challenges and research questions a position paper. *SIGKDD Explorations*, 15(1), 2013.