Heuristic Methods for Automating Event Detection on Sensor Data in Near Real-Time

Daniel Mauer, Barry Lai, Jennifer Casper, Peter Leveille, Jing Hu, Ronald Albuquerque and Eddy Cheung
The MITRE Corporation
Bedford, Massachusetts 01730–1420
Email:{dmauer, blai, jcasper, psl, jinghu, ralbuquerque, echeung}@mitre.org

***Abstract*—Moving Target Indicator (MTI) analysts in the field are responsible for processing the increasing amounts of live streaming data. Analysts manually access unique data sources through a set of tools, and perform analysis on the available data. Operationally, analysts can only concentrate on small areas of interest and are subject to attentional blindness. Abnormalities in the periphery are often not detected until the forensic stage. Analysts are in need of assistance in performing data analysis.**

This paper presents the implementation of a heuristic-based stream mining approach for cueing the analyst user on patterns in near real-time. This approach is designed to aid analysts in detecting noteworthy events scattered within the overabundance of data, a problem which is well-documented and recognized [1], [2]. The implementation involves two-phases: the isolation of areas of unusual activity using density grids, followed by event detection within those areas. Four events (aka. patterns) - starburst, inverse starburst, fanning, and inverse fanning - were identified for automated detection using these techniques. The event detection method was employed as a service within the Sensor Data & Analysis Framework (SDAF). The algorithm implementation and evaluation produced findings and informal user feedback. The results of this effort aids in establishing the foundation for near real-time event detection in MTI data analysis.

I. Introduction

Automating event detection on operational data is an identified problem. According to the JASON Defense Advisory Board [1]:

> As the greatest challenge will come from the need to automate analysis, the most immediate need is for algorithmic advances that can help cue the analyst and trigger closer observation as well as possible fusing of other relevant data. (p. 3)

The advisory board recognizes the need for automated analysis while understanding the initial steps, such as cueuing.

This effort presents an approach, beyond basic data characterization and filtering, to the event detection challenge in a near real-time environment. Section III describes a heuristic method used to detect four analyst-identified patterns: starburst, inverse starburst, fanning, and inverse fanning. The method is evaluated using simulated and exercise data, as discussed in Section IV. Section V reveals the findings from informal user feedback, which provide the direction for future work outlined in Section VI.

A. Problem Statement

There is a common perception that data collected by sensors is increasing at a possibly unmanageable rate within the Department of Defense (DoD) and Intelligence Community (IC). The JASON Defense Advisory Board recognizes this perception although report that "data requirements are certainly significant but not unmanageable given trends in storage technology"(p. 2). Data-intensive fields of science, such as high energy physics, are similar to the DoD and IC through the volumes of data requiring analysis. It is estimated that these data-intensive fields will experience exponential growth to the 100's of Petabytes by 2015 [1]; this estimation includes consideration of user needs. Given the similarity in data volumes handled by the data-intensive science fields and the DoD/IC, the DoD/IC may assume to experience similar growth.

Ultimately, the analyst user is at the receiving end of this data pipe. Assisting the analyst is a crucial part of the data analysis problem. The JASON Defense Advisory Board highlights the need to view the analysis timeline: "the key challenge is to empower the analyst by ensuring that results requiring rapid response are made available as quickly as possible while also insuring that more long term activities such as forensic analysis are adequately supported"(p. 2). These time requirements dictate how the data should be handled (see Table I derived from [1]).

Time Period	Method
retrospective	homogeneous data structures
intermediate	service-oriented architectures
rapid response	event-driven architectures

TABLE I
SUGGESTED DATA HANDLING BASED ON TIME REQUIREMENTS.

The data handling requirements outlined in Table I are addressed by numerous efforts. See [3] for a summary of related intermediate and rapid response approaches. The effort in [4] is an example of an event-driven service-oriented architecture designed for rapid response on near real-time data streams and historical data.

Despite the potential to manage the increasing amounts of data, an estimated 70% of collected data is not processed [1]. Processing the data, through analysis and fusion, remains a challenge.

B. Motivation

Moving Target Indicator (MTI) analysts in the field are responsible for processing the increasing amounts of live streaming data. Analysts manually access unique data sources through a set of tools, and perform analysis on the available data. Operationally, analysts can only concentrate on small areas of interest and are subject to attentional blindness [5]. Abnormalities in the periphery are often not detected until the forensic stage. Forensic analysis has proven to be very effective; therefore research in abnormality detection has primarily focused on the forensic analysis of sensor data. However, any intelligence on interesting activities gathered and reported in near real-time can prove invaluable to the situational awareness of warfighters in theatre. Used in conjunction with high fidelity forensic intelligence, near real-time intelligence extracted from streaming data will enable warfighters to proactively meet emerging threats. Assistance is needed to share the cognitive load in analyzing streaming data.

C. Contributions

Employing event detection methods on streaming data has the potential to improve operational analysts' efficiency in theatre. This increases situational awareness and shortens decision-response time by allowing analysts to quickly key in on important events in near real-time. Although event detection methods are potentially useful, the methods must be useable and proven to detect events at an acceptable rate. A benchmark, detailing usability and detection rate, is currently undefined given event detection is new to the analyst community.

This effort introduces event detection methods, beyond filtering, to the operational analyst community. The algorithm implementation and evaluation produced findings and informal user feedback. The results of this effort aids in establishing the foundation for near real-time event detection in operational analysis.

II. Related Work

Data stream processing, or *stream mining*, is an established field of research, as evidenced by the large amount of literature covering the topic. The authors in [6] provide an overview of the field. The following paragraphs present related work within data stream mining which includes systems and algorithms.

A. Stream Mining Overview

Stream mining is employed in fields that need to reveal events in incoming data, such as the financial industry or monitoring applications. The nature of streaming data involves a one-pass viewing of the data, in part due to the potential volume of incoming data. Charu Aggarwal provides the definition for streaming data while presenting a survey of the stream mining field in [7]. Throughout the survey of existing stream mining methods, Aggarwal emphasizes that there is not a direct application of multi-pass algorithms to the one-pass streaming data. This is due to the temporal locality of data streams as they evolve over time. Thus, stream mining algorithms need to be designed for data evolution in addition to one-pass processing. On the contrary, Mohammed Gaber from Monash University draws similarities between data mining and stream mining; as data stream mining can be seen as a direct subset of traditional database mining. His work in [8] presents how data mining techniques may be used on data streams.

Current and future research focus for stream mining includes mining methods such as stream clustering (nearest-neighbor searches), frequent pattern mining, change detection (anomaly search), stream synopsis (minimization of data stream by emphasizing present input), and new emerging ideas such as distributed data stream mining (separating different tasks of mining to different systems) [7].

B. Systems

The authors in [9] present the following six requirements for a stream mining system:

1) Small, constant processing time per record to prevent falling behind.
2) Use a fixed amount of memory.
3) Build a model using one-pass of the streaming data, given potential data volumes and no guarantee the data is stored.
4) A model should be available at any point in time given there may be no end to the data.
5) The model produced should be equivalent to one generated by a database mining algorithm.
6) Maintain an update model that withstands concept-drift within the streaming data (i.e. an evolving data stream) and incorporates past data.

Systems, such as the streaming query engines detailed in [10], were designed to handle querying streams in near real-time. The work in [11] provides detail of a system that consumes real-time radio-frequency identification (RFID) data and supports querying. The authors in [12] consider a framework for producing trends within data streams.

The effort detailed in [3], [4], [13] takes a slightly different approach through providing an integrated query on streaming and historical data simultaneously; the intent was to reduce time analysts spent manually querying different data sources. The Sensor Data & Analysis Framework (SDAF) is an event-driven service-oriented architecture (SOA) which manages the logistics of executing a user's spatial-temporal query. This simplifies querying for the user by removing the need to know how to access individual data sources. SDAF was designed to handle streaming data, thus attempts to comply with the requirements listed above. Analysts were interested in querying and being alerted of patterns within the data. SDAF was modified to support event detection services [4]. SDAF provided the framework by which the effort presented in this paper was implemented and evaluated.

C. Algorithms

Data clustering has been extensively studied in data mining, as evident in [14]. However, its application on data streams has proven problematic because the dataset can only be seen

by the algorithm at most once for real-time mining. K-means and Fuzzy-C are two general methods used. Many of these algorithms deal with determining distance between points, and also using a predetermined definition of distance for each case of data. In [15], the authors provide an overview of general techniques for data stream clustering algorithms. The authors interestingly comment that the clustering technique provided by [16] only outputs cluster centers and does not store the metadata of which cluster a specific point has been assigned to. This method reduces overhead, but may be problematic in maintaining data pedigree. The authors in [17] present an example of a more recent clustering algorithm which uses time windows. The benefit of time windows is under debate.

Many efforts have addressed frequent pattern mining, including [18]–[21]. Frequent pattern mining is useful in gaining knowledge of how often an event occurs. The authors of chapter four in [7] cite memory management and computational costs as the emblematic problems of frequent pattern mining on data streams. Algorithms described in the chapter attempt to find useable error bounds in order to approximate results, minimize memory, and computational usage. The challenge with frequent pattern mining is that it is often case specific.

Outlier detection can be thought of as a specialized case of clustering or the inverse of clustering. Instead of finding near-neighboring data points, outlier detection searches for data largely differing from past, current, or future. The authors in [22] provide a brief synopsis of outlier detection methods, including a grid technique for maintaining statistical data. The authors in [23] created a distributed algorithm using a kernel density function for observing outliers in a sensor network intended to detect broken sensors. The authors in [24] present two contrasting algorithms for outlier detection. The first algorithm focuses on accurate results at the expense of increasing storage space. The second algorithm uses the sliding window for approximate answer with bounded errors. The paper also comments on the querying approach for outlier detection. The authors suggest that a continuous analysis of the data stream for outliers may lead to inaccurate results due to concept drift of incoming data. Their solution was to use a single query approach to analyze the data stream only when wanted. The effort in [25] generalizes the outlier detection problem to consider anomalies within streaming data representing moving targets; the algorithms used include local clustering.

Other groups have considered methods alternative to those discussed above. The authors in [26] focus on density-based clusters and distance-based outliers using neighbor-based patterns on MTI data. Machine learning techniques are also options for stream mining. The authors of [27] directed the effort on the development of a tool to work with the analyst. The tool would learn the intelligence analyst's behaviors in order to serve the analyst. The intent was to compliment the analyst's cognitive strengths with the tool speed and pattern recognition ability. The authors in [28] provide a survey of uncertain data mining techniques. These methods include the classic techniques above.

The algorithm methods discussed in this section may be of use for MTI data. Filtering and data characterization are two current methods used to indicate potential events in MTI data. An analyst may use filtering to view data only in a certain area and of a specified type. Behaviors or objects may be inferred given specific data characteristics as well. This is challenging in that large amounts of data with behavior and object ground truth need to be available to build and test the data characterizations for certainty. The effort detailed in Section III goes beyond simple filtering and data characterization by making use of heuristic techniques to cue on spatial-temporal patterns recognized by analysts.

III. Two-Phase Approach for Live Data Streams

The primary constraint for this effort was imposed by the users' need for near real-time information. Near real-time is defined by the National Communications System as "pertaining to the delay introduced, by automated data processing, between the occurrence of an event and the use of the processed data" [29]. There is a time lag between when the event being picked up by the ground MIT (GMTI) sensor and the data being viewed by the analyst. The following steps occur before the analyst views the data:

1) GMTI sensors collect the data containing the event.
2) Data is transferred to a ground station.
3) Ground station re-broadcasts the data for consumption.
4) Data is translated and ready for processing.
5) Data processing is completed.

This is a variable time lag that constitutes a GMTI data stream being classified as near real-time. Event detection processing is not useful if too much time is added to the period between event occurrence and analyst viewing. Analysts will ultimately define an acceptable detection response time after the capability is available and in testing.

In order to address the near real-time constraint, a two-phase method was devised to minimize computation time:

1) **Dot Group** - Determine which groups of dots (aka. GMTI detections) constitute an individual event.
2) **Event Type** - Analyze dot groups to determine whether any specific recognizable types of events were present in the group.

A. An Alternative to Traditional Clustering

Initially, standard clustering algorithms were considered as a means of grouping dots. These algorithms included hierarchical clustering, K-means and derivatives, and Quality Threshold (QT) clustering [7]. Each method presented challenges; either they were simply too slow to seem viable in a near real-time context, or had other limitations that conflicted with the analyst field. For instance, K-means requires specifying the number of clusters in advance. Analysts are concerned with evaluating an area for events; the number of events are not known advance.

The aforementioned two-phase method is feasible within the operational analysis environment and had the potential to tolerate live data streams. The first phase is an alternative to traditional clustering. It utilizes a density grid to divide

the map into approximately equal-sized regions. Rather than looking at apparent clusters of dots, the system considers each region as the potential location of an event. The intuition behind this method is the promise of feasibility in a near real-time context, while still being effective in extracting individual, regional events from a large area.

Multiple methods were considered for the second phase. An initial concept was to look at each potential event as something approximating a time-lapse image of the activity in the region. This opens up the option of employing machine-learning-based computational vision techniques for event detection. While this idea is still a possible avenue for later experimentation, the technique that was implemented for this effort was a heuristic one. A known set of events were requested by analysts for detection, and the simplest, most straightforward mathematical description of what each detectable event looks like provided the detection foundation. For example, the starburst pattern can be simply described as a group of dots in close proximity to each other that diverge over time. Such a time-based pattern is relatively simple to translate into the language of mathematics; therefore, it can be represented algorithmically.

B. Isolating Unusual Activity using Density Grids

The analyst's area of interest (AOI) is divided into *g* overlapping grids with square (or nearly square, based on geospatial measurements) cells. Every dot on the map, therefore, falls into g different cells, one from each overlapped grid. The intent is to avoid the possibility of a single noteworthy activity occurring across multiple cells within a grid, which would result in the system seeing two separate "events", neither of which capture the true character of the actual event.

The number of overlapping grids and the cell granularity are not limited per se, but more and finer grids require more processing power. The grids themselves are represented as *DensityGrid* objects within the implementation. Each DensityGrid object contains one or more overlapping grids of equal granularity and uniform spacing. Multiple DensityGrid objects can be used to define grids of varying granularity. The grid cells are represented as *GridSquare* objects.

Average activity represents the number of dots per GridSquare per second. This is measured over a time window for all GridSquares. The time window for determining average activity is generally one to two hours. This time window was found to work well with progressive traffic, but can be easily adjusted to fit different environments. The assumed average activity for GridSquares which have not yet been "seen", may be dealt with in various ways. For instance, assume average activity is equal to the global average, or a regional average. Figure 1 presents a visual notion of a density grid containing uniform squares with computed activity averages.

Current activity is measured in dots per GridSquare per second as seen in the past *u* seconds. U is a user-defined parameter. It represents the duration of the current activity window. Current activity is used to determine unusual activity level. Figure 2 shows the current activity for GridSquares.

Fig. 1. Density grid with uniform grid squares and computed activity averages.

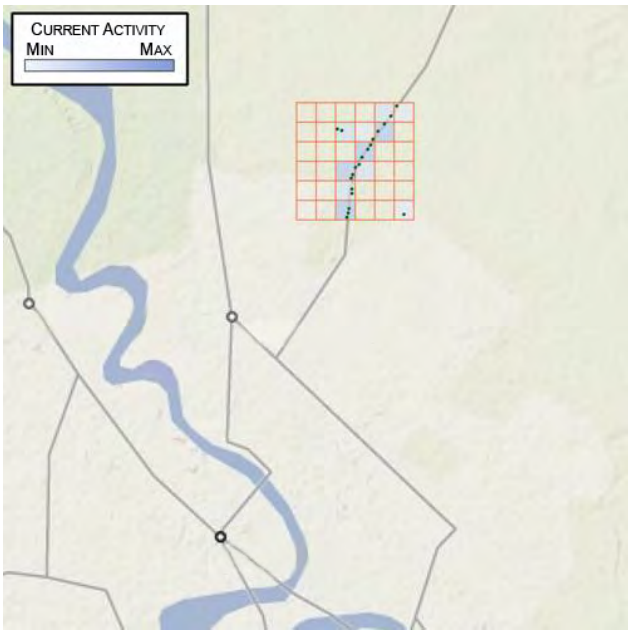

Fig. 2. Density grid with uniform grid squares and computed current averages.

The amount of *unusual* activity, defined as the current activity divided by the average activity, is calculated for each GridSquare. A small smoothing constant is added to account for noise in the data. Figure 3 presents the unusual activity levels calculated for a DensityGrid.

The GridSquares exhibiting an unusual activity level above a user-defined threshold are flagged for further processing as individual units. In order to make efficient use of resources, unflagged GridSquares are ignored; exhaustively processing

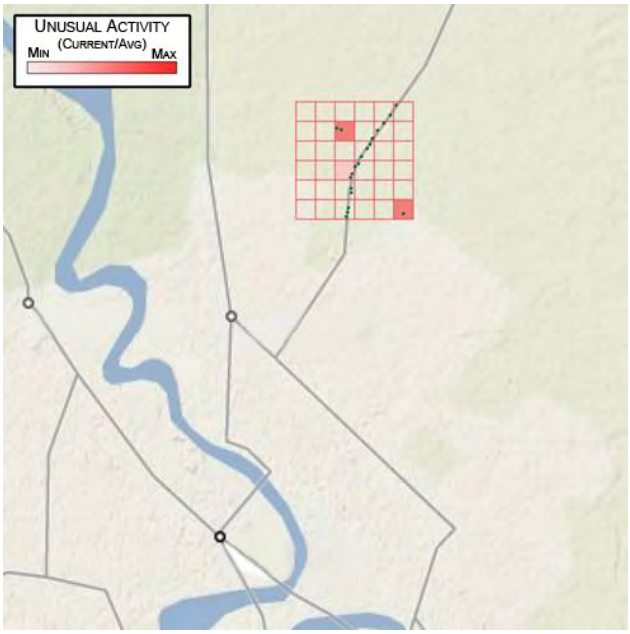

Fig. 3. Density grid with uniform grid squares and computed unusual activity levels.

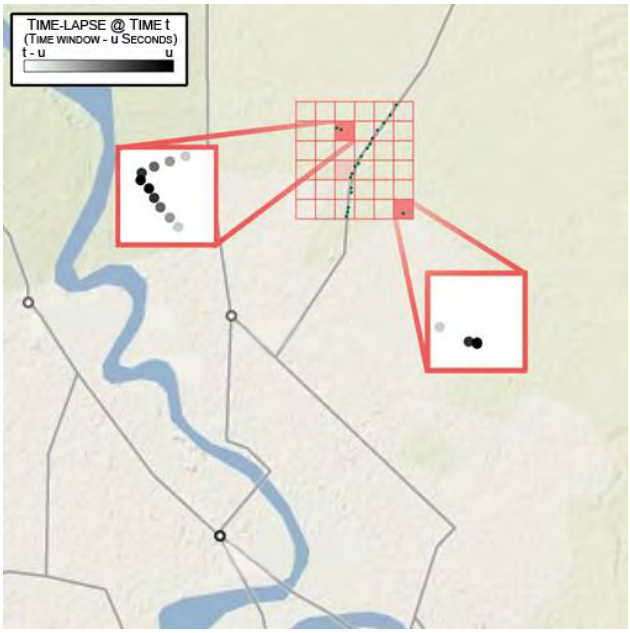

Fig. 4. Notional time-lapse visual of dots appearing within the current time window.

every GridSquare would likely be far too computationally expensive to fit within the near real-time constraint. The result is a collection of areas on the map, each with an unusual level of activity, and each containing the GMTI dots detected within the currently active time window. A visual representation of each GridSquare over the current activity window of size u would look like a time-lapse image of the dots as they progressively appeared (see Figure 4).

C. Event Detection within Unusual Activity GridSquares

Once a GridSquare is flagged, it is sent through a series of tests to detect the following patterns:

1) **Starburst** - dots diverging in various directions from a common location
2) **Inverse Starburst** - dots converging on a common location from various directions
3) **Fanning** - dots diverging from a common location in a fan pattern with uniformly spaced directions; this looks like a moving, directed starburst
4) **Inverse Fanning** - dots converging on a common location from uniformly spaced directions; this looks like an organized moving, directed starburst

There are two primary dot traits used to identify these four patterns: *change in dispersion* and *average dot position*. Starburst and inverse starburst patterns are detected by measuring the change in dispersion of dots within a GridSquare between the early and late parts of the time window. Fanning and inverse fanning patterns are detected by measuring a combination of change in dispersion and change in average position.

Definition 1. *With a geospatial temporal data sequence containing collections of longitude (x), latitude (y), and time stamp (t) triplets, $d = \{x, y, t\}$, is called a* ***dot****.*

Definition 2. *A* ***GridSquare*** *contains a set of dots within a defined region over a time window, $GS = \{d_1 \dots d_k, d_{k+1} \dots d_n\}$, where d_k is the last dot detected in the first half of the current time window and d_n is the last dot detected in the second half.*

The classification method used to detect starburst, inverse starburst, fanning, and inverse fanning events uses the dots within a GridSquare as input. It contains four steps given the following:

- d_i is the i^{th} dot detected
- τ_μ is the gross movement to distinguish between starburst behavior and fanning behavior
- τ_σ is the threshold for the minimum dispersion required to report starburst andor fanning behavior

1) Calculate Early Centroid:

$$\mu_{x_1} = \frac{\sum_{i=1}^{k} d_{i_x}}{k} \quad \mu_{y_1} = \frac{\sum_{i=1}^{k} d_{i_y}}{k} \tag{1}$$

2) Calculate Late Centroid:

$$\mu_{x_2} = \frac{\sum_{i=k+1}^{n} d_{i_x}}{n-k} \quad \mu_{y_2} = \frac{\sum_{i=k+1}^{n} d_{i_y}}{n-k} \tag{2}$$

3) Calculate Gross Movement:

$$\Delta_\mu = \sqrt{(\mu_{x_1} - \mu_{x_2})^2 + (\mu_{y_1} - \mu_{y_2})^2} \tag{3}$$

4) Calculate Change in Dispersion:

$$\sigma_{GS} = \left(\frac{\sum_{i=1}^{k} \mid d_{i_x} - \mu_{x_1} \mid}{k} + \frac{\sum_{i=1}^{k} \mid d_{i_y} - \mu_{y_1} \mid}{k} \right) - \left(\frac{\sum_{i=k+1}^{n} \mid d_{i_x} - \mu_{x_2} \mid}{n-k} + \frac{\sum_{i=k+1}^{n} \mid d_{i_y} - \mu_{y_2} \mid}{n-k} \right) \tag{4}$$

5) Interpretation of Resulting Values:

$$\begin{aligned} \sigma_{GS} > \tau_\sigma \; and \; \Delta_\mu > \tau_\mu &: Fanning \\ \sigma_{GS} > \tau_\sigma \; and \; \Delta_\mu \leq \tau_\mu &: Starburst \\ \sigma_{GS} < -\tau_\sigma \; and \; \Delta_\mu > \tau_\mu &: Inverse \; Fanning \\ \sigma_{GS} < -\tau_\sigma \; and \; \Delta_\mu \leq \tau_\mu &: Inverse \; Starburst \\ Level \; of \; \text{“}confidence\text{”} \; reported \; in \; [0,1] &= \mid \sigma_{GS} \mid \end{aligned} \quad (5)$$

IV. Evaluation

The GridSquare model has proven effective for picking out and detecting areas of interest which likely contain a particular event type. The present implementation simply alerts the user every time a grid square matches the user's query. However, this simple model has limitations. First, a list of individual GridSquare detections often provides an overabundance of information. If there is an sudden increase in activity over a larger area, for example, the user would be alerted to a large number of GridSquares containing unusual activity levels; a better solution would be to alert the user to the area which contains all those GridSquares. As such, we propose a further level of abstraction above the grid square concept; that of an interesting area (IA).

To implement the IA model and better tie together multiple DensityGrid objects, ActivityMap class is needed. A single ActivityMap will consist of any number of DensityGrids, and will keep a list of all IAs in memory. Rather than sending alerts each time a new noteworthy GridSquare is detected, the ActivityMap will simply give the user a customized view of this list. Figure 5 represents a single ActivityMap instance, containing two DensityGrids. One DensityGrid has a 2x overlap and a smaller unit size; the other is larger and has no overlap. Based on the ActivityMap in Figure 5, two IAs are found (see Figure 6.

Each of the GridSquares (colored red, blue and green) that fall within the IAs (colored purple) would be processed to determine if any recognized event types are taking place within them. The IAs themselves, then, would be marked with those event types. Each time a new GridSquare is determined to be noteworthy, that GridSquare will either be added to an existing IA (if it falls within the IA borders), or will trigger the creation of a new IA containing only that GridSquare.

The user interface corresponding to the ActivityMap would provide the user with a method of viewing not only all currently active IAs, but also to browse IAs which have come and gone. The user would also be able to direct the system either to disregard a specific IA until a new GridSquare is added to it, or to disregard it permanently. This system should eliminate much of the extraneous or redundant data inherent in the currently-implemented model.

A. Operational Exercise Data Used

The event detection services were evaluated against exercise data. The first data set tested against was from a military field exercise conducted in 2008. This data was replayed in a lab and fed into the system to see where events would be detected in a live scenario.

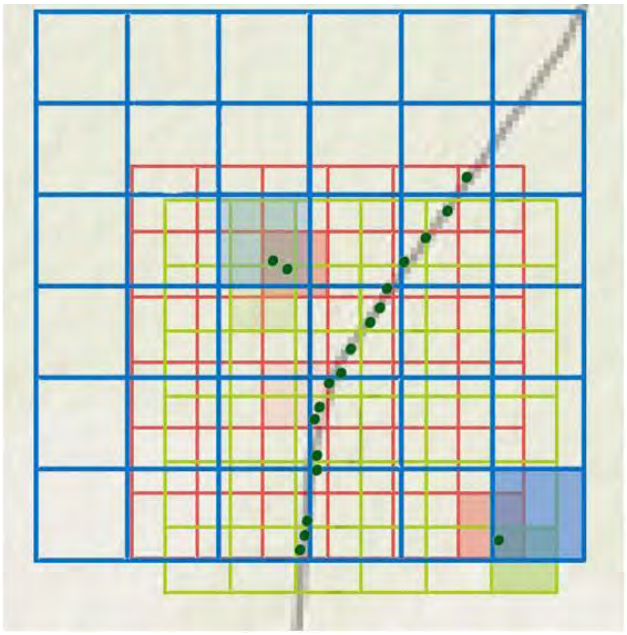

Fig. 5. Activity map containing two density grids.

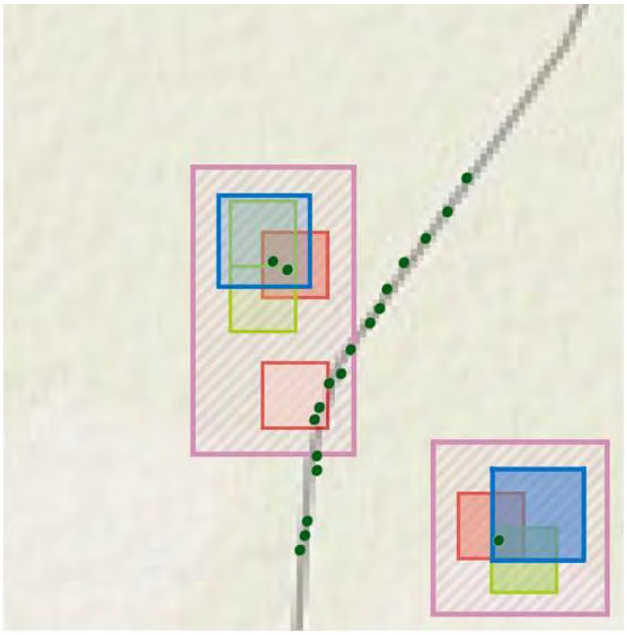

Fig. 6. Two interesting areas derived from the ActivityMap.

The system was deployed in Empire Challenge 2009 (EC09) [30] as part of the Army operational analysis group. The exercise included data feeds from Virtual Surveillance Target Attack Radar System (VSTARS) [31], Littoral Surveillance Radar System (LSRS) [32], and King Air [33]. This allowed informal evaluation of the event detection services in a real-time environment. See Section V for findings from feedback and observations.

B. Affect on Near Real-Time Detection

In order to support near real-time event detection, the SDAF system was modified to accommodate more complex queries [4]. The complex queries made use of the event detection implementation detailed in Section III when requested by the user. Queries utilizing these algorithms must reach a level of efficiency in order to keep pace with incoming data. The following tests were performed with the primary objective of comparing simple query performance with queries of increasing complexity due to the processing required for event detection: DotEvent throughput, overhead in Border/Road detection, and overlapping grid squares. These tests provide a baseline indicating how the event detection implementation affects near real-time analysis.

Testing was performed on the following hardware and software:

- Intel Xeon E5345 2.33 GHz
- 4.0 GB RAM
- Microsoft Windows XP Professional x64 Service Pack 2
- Java 1.6.0_13
- JBoss AS 5.1.0.GA

Query parameters for event detection were defined as follows:

- Threshold: 6.0
- Limit: 20
- Grid Size: 1000
- Overlap: 2
- Current Window Size: 180
- Full Window Size: 7200
- Boundary Distance: 100

1) DotEvent Throughput: The test objective was to compare the dot event throughput between a base query with minimal processing to the throughput of the complex event detection queries. A dot event represents a moving target data point detected by radar (see [3] for details on the framework and data ingestion). The base query includes minimal processing which simply forwards all events received by the system to the client. The complex queries include detecting starburst, inverse starburst, fanning, and inverse fanning patterns. The full event detection query involves simultaneous detection of the four patterns as well as unusual border and road activity. Each test lasted one hour with a restart of the application server and deletion of temporary JBoss files before subsequent tests. GMTI dots were streamed to SDAF via an in-house application that parses GMTI input files. Data used for testing consisted of four different simulated NatoEx files streamed simultaneously.

Note that the number of detected events for each query is only determined by the number of dot events representing GMTI. For example, event detection queries would produce user alert events that signal the positive analysis for a GMTI pattern (resulting in total number of events to be GMTI dot events + user alert events). Approximately one thousand to two thousand user alert events were found per event detection query in each of the hour long tests.

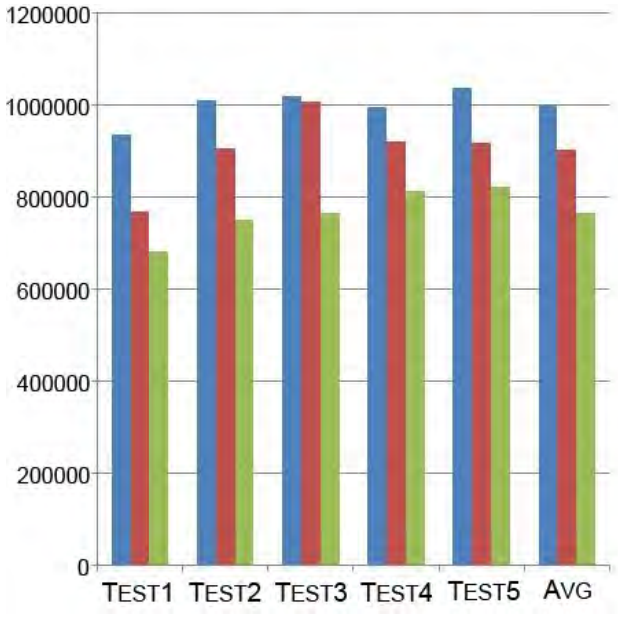

Fig. 7. Query dot event throughput comparison between the base query (blue), event detection without border activity (red), and full event detection (green).

With the base query, 100% of dots sent through the system were successfully processed by the system in the hour long time span. The system easily handled dots at an average rate of 278 dots per second with observed peaks of approximately 800 dots per second.

As expected, the complex queries had lower throughput than the less complex queries. There is an observed 9% decrease in throughput performance for the event detection query from the base query and a 23% decrease in throughput performance for the full event detection query which includes border activity. While these numbers suggest that the algorithms are unable to keep up with the live stream at this rate, they may just be indicative of modifications needed within SDAF. For instance, performance optimizations may be made in the event queue management within SDAF to improve the throughput [4]. However, a test run at an assumed realistic rate of approximately 100-300 dots per second showed the system processing 100% of data in a 15 hour interval, analyzing over 8 million MTI dot events and discovering over 50,000 user alert events.

2) Overhead in Border/Road Detection: The objective of this test was to understand the change in throughput when performing border/road activity detection, in addition to the four patterns. Detecting border or road activity was implemented as a service within SDAF that utilized shapefiles to calculate dot proximity from borders and roads. This was an event option for users to select when querying activity in an AOI. Although the border/road detection is not the focus of this paper, understanding the throughput changes when this filter is used in addition to the event detection reveals how performance is affected when event detection is combined with another filter.

The “full” query includes the event detection plus border

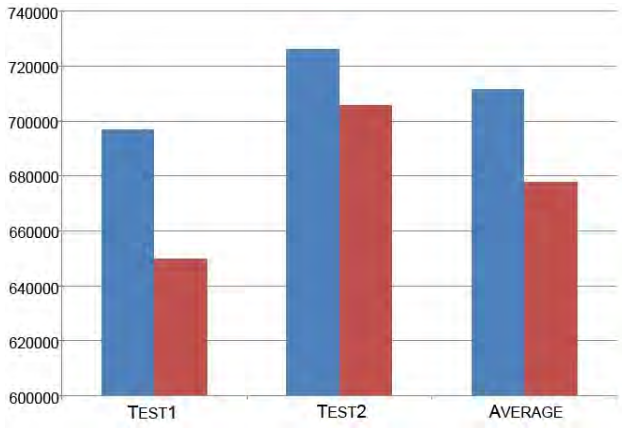

Fig. 8. Dot event throughput for three tests comparing overlap parameter values of three (blue) and four (red).

and road activity. The overhead of calculating whether or not a grid square crosses a road or border occurs towards the beginning of runtime since the calculation is only performed once per grid square creation. Since a grid square is only created if a dot event appears in a new location, it would be wise to use border/road activity queries with a spatial operator; i.e., in a spatial query blocking off an area of interest. After this initial overhead, performance becomes marginally slower than that of the event detection query without border/road activity. Thus, while there is a significant throughput difference from the full event detection query from the base query, a longer testing time would close the performance gap and come closer to the reasonable 9% throughput decrease. A Java profiler was used to monitor the CPU usage of the JBoss services with results in parallel to this trend. The Border Activity service consumed 36.9% of the CPU time at the beginning of query runtime; this decreased to 13.4% after the initial overhead.

3) Overlapping Grid Squares: The objective of this test was to observe the performance effects of increasing the number of overlapping grid squares used for analysis. An interesting parameter in the event detection analysis is the number of overlapping grids. This gives the operator some ability to specify the granularity of the event analysis. The ideal value for the overlapping grid parameter was empirically defined to equal 3. The event detection query with no border or road activity was used with an overlapping grid parameter of 3 and 4. Tests were performed with the same data sources and hour long runtime as before.

From the observed results, there is indeed a large drop in throughput with more overlapping grids. Compared to the average results of the queries previously performed at 2 overlapping grids, the average throughput drops 22% with 3 overlapping grids and 27% with 4 overlapping grids (see Figure 8).

V. Findings from Informal User Feedback and Observation

The SDAF system, with event detection, was deployed in the Danville combined air operations center (CAOC) along with Distributed Common Ground System-Army (DCGS-A) operators at Empire Challenge 2009. The analysts provided feedback throughout the four week exercise. All analysts expressed interest in the event detection algorithms and stated that they could see utility in using the automated alerting mechanism to help them identify locations where interesting activity may be occurring.

The analysts suggested several **additional types of activity that would be useful for alerting**. For instance, the demonstrated event detection alerted users when levels of activity increased to unusual levels. Analysts commented that it was also useful to know if there was a sudden decrease of activity in an area as well. The analysts also stated an interest in detecting certain correlations between friendly-force activity and the GMTI patterns.

A recurring theme was the fact that **analysts would have far more trust for services which were trying to detect simple events (e.g. starburst pattern) then complex events (e.g. explicit human behaviors)**. If the analyst could understand what the algorithm was trying to do, they would more easily trust the results of the algorithm.

Analysts also noted that the current **system sent out too many alerts using a single modality**. For instance, each detection of border activity would generate an alert. This meant that a single vehicle traveling across a border could generate dozens of alerts. This deluge of alerts made the alerting system unmanageable, and prevented analysts from following up on individual areas of interest.

The EC09 exercise provided an opportunity to observe how the event detection methods worked on GMTI data streaming directly off the sensor. It was noted that cloud formations and sand storms can create pockets of false detects which interfere with the event detection services, creating a significant number of false event detections. Within the clean GMTI, several false positives of starburst activity were noted. Further evaluation on data with ground truth is needed to understand accuracy.

VI. Conclusions and Future Work

This paper describes the implementation of a heuristic-based stream mining approach for alerting the analyst user on patterns in near real-time, designed to aid analysts in detecting noteworthy events scattered within the overabundance of data, a problem which is well-documented and recognized [1], [2]. The implementation involves two-phases: the isolation of areas of unusual activity using density grids, followed by event detection within those areas. Four events (aka. patterns) - starburst, inverse starburst, fanning, and inverse fanning - were identified for automated detection using these techniques. The event detection method was employed as a service within the Sensor Data & Analysis Framework (SDAF). SDAF, an event-driven SOA designed to handle streaming and historical data, managed the user queries including event detection, data sources, and client applications for viewing results. The following tests were performed with the primary objective of comparing simple query performance with queries of increasing complexity due to the processing required for event detec-

tion: DotEvent throughput, overhead in Border/Road detection, and overlapping grid squares. These tests provide a baseline indicating how the event detection implementation affects near real-time analysis. In addition, the system was deployed at the 2009 Empire Challenge exercise. Informal user feedback provided early support for continued efforts to advance this needed event detection capability.

The directions for future work are driven by the results, findings and informal feedback presented in this paper. First, we will further examine the efficacy of the event detection implementation through testing with recently collected data sets that include ground truth. This will help increase the confidence in using event detections methods. Second, analysts have identified additional patterns of interest. It would be useful to grow the event detection service for MTI. Third, we will explore other options for event detection on MTI data such as machine learning and pattern recognition.

The authors of [1] encourage contributions in near real-time event detection through the following finding: "there is insufficient investment in software to more effectively process data as opposed to hardware to both collect and store data"(p. 3). We will continue this work in an effort to move towards an accepted, automated data analysis.

ACKNOWLEDGMENT

The authors would like to thank Mr. William Harris for his input, support, and introductions. Also, the authors thank the Joint Test Force analysts who participated in Empire Challenge 2009. Their feedback is priceless.

REFERENCES

[1] JASON Defense Advisory Panel, "Data analysis challenges," JASON, 7515 Colshire Dr. McLean, VA, 22102, Tech. Rep. JSR-08-142, 2008.

[2] H. McFadden, "Defining specific needs for automated tools that support manual analysis," presented at the S/GMTI Community of Practice, 2009.

[3] R. Albuquerque, J. Casper, E. Cheung, R. Couture, B. Lai, P. L. J. Hu, and D. Mauer, "Improving sensor data analysis through diverse data source integration," in *Proceedings of the IEEE Military Communications*. Boston, MA, USA: IEEE, 2009.

[4] R. Albuquerque, B. Lai, J. Hu, D. Mauer, J. Casper, E. Cheung, and P. Leveille, "Event-driven soa: Performance under diverse high volume data streams," The MITRE Corporation, 202 Burlington Rd. Bedford, MA, 01803, Tech. Rep. MTR100007, 2010.

[5] P. J. Durlach, "Change blindness and its implications for complex monitoring and control systems design and operator training," *Human-Computer Interaction*, vol. 19, no. 4, pp. 423–451, 2004.

[6] J. Gehrke, "Data stream processing - when you only get one look," *Communications of the ACM*, vol. 52, no. 10, p. 96, 2009.

[7] C. C. Aggarwal, *Data Streams: Models and Algorithms*. New York City, NY: Springer-Verlag, 2007.

[8] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: A review," *ACM SIGMOD*, vol. 34, no. 2, p. 26, 2005.

[9] P. Domingos and G. Hulten, "Catching up with the data: Research issues in mining data streams," in *Proceedings of the 201 Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2001, pp. 47–51.

[10] J. Hyde, "Data in flight," *Communications of the ACM*, vol. 53, no. 1, pp. 48–52, 2010.

[11] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," in *Proceedings of the Special Interest Group on Management of Data*. Chicago, Illinois, USA: ACM, 2006.

[12] S. Gollapudi and D. Sivakumar, "Framework and algorithms for trend analysis in massive temporal data sets," in *Proceedings of the Thirteeth Conference on Information and Knowledge Management*. Washington, DC, USA: ACM, 2004.

[13] E. Cheung, "Sensor data & analysis framework (sdaf) data warehouse," The MITRE Corporation, 202 Burlington Rd, Bedford MA, 01803, Tech. Rep. MTR070028, 2007.

[14] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM computing surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[15] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 515–528, 2003.

[16] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, "Incremental clustering and dynamic information retrieval," in *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*. ACM, 1997, pp. 626–635.

[17] D. K. Tasoulis, N. M. Adams, and D. J. Hand, "Unsupervised clustering in streaming data," in *Sixth IEEE International Conference on Data Mining Workshops*, 2006, pp. 638–642.

[18] C. Cormode and M. Hadjieleftheriou, "Finding the frequent items in streams of data," *Communications of the ACM*, vol. 52, no. 10, pp. 97–105, 2009.

[19] C. C. Aggarwal, Y. Li, J. Wang, and J. Wang, "Frequent pattern mining with uncertain data," in *Proceedings of the Fifteenth Conference on Knowledge Discovery and Data Mining*. Paris, France: ACM SIGKDD, 2009.

[20] R. Dass and V. Kumar, "Kall - a real time stream mining algorithm," in *Proceedings of the 43rd Hawaii International Conference on System Sciences*. Koloa, Kauai, Hawaii: IEEE Computer Society, 2010.

[21] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman, "Efficient pattern matching over event streams," in *Proceedings of the Special Interest Group on Management of Data*.

[22] H. Chi, "Online outlier detection over data streams," Master's thesis, Simon Frasier University, British Columbia, Canada, 2005.

[23] T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos, "Distributed deviation detection in sensor networks," *ACM SIGMOD*, vol. 32, no. 4, p. 82, 2003.

[24] F. Angiulli and F. Fassetti, "Detecting distance-based outliers in streams of data," in *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management*. ACM, 2007, pp. 811–820.

[25] Y. Bu, L. Chen, A. W.-C. Fu, and D. Liu, "Efficient anomaly monitoring over moving object trajectory streams," in *Proceedings of the Fifteenth Conference on Knowledge Discovery and Data Mining*. Paris, France: ACM SIGKDD, 2009.

[26] D. yan, E. Rundensteiner, and M. Ward, "Neighbor-based pattern detection for windows over streaming data," in *Proceedings of the 12th International Conference on Extending Database Technology*, Saint Petersburg, Russia, 2009.

[27] G. Tecuci, M. Boicu, C. Ayers, and D. Cammons, "Personal cognitive assistants for military intelligence analysis: Mixed-initiative learning, tutoring, and problem solving," in *Proceedings of the First International Conference on Intelligence Analysis*, 2005, pp. 2–6.

[28] C. C. Aggarwal and P. S. Yu, "A survey of uncertain data algorithms and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 5, pp. 609–623, May 2009.

[29] National Communication System. (1996) near real time. http://www.its.bldrdoc.gov/fs-1037/dir-024/_3492.htm. [Online; accessed 03-June-2010].

[30] United States Joint Forces Command. (2010) Posts tagged empire challenge. http://usjfcom.dodlive.mil/tag/empire-challenge/. [Online; accessed 28-June-2010].

[31] G. Marchand, "Vstars: A step success story," in *International Test and Evaluation Association Modeling and Simulation Workshop*. Joint Advanced Distributed Simulation Joint Test Force, 1998.

[32] C4ISR Journal. (2009) Revealing radar: U.S. Navy partially lifts shroud on insurgent-tracking system. http://www.c4isrjournal.com/story.php?F=3816224. [Online; accessed 28-June-2010].

[33] Aerospace Technology. (2010) Hawker beechcraft beech king air 350 twin-turboprop passenger aircraft, usa. http://www.aerospace-technology.com/projects/beach_king_air350/. [Online; accessed 28-June-2010].