myjournal manuscript No.
(will be inserted by the editor)

On-board Vehicle Data Stream Monitoring using MineFleet and Fast Resource Constrained Monitoring of Correlation Matrices

Hillol Kargupta[1], **Vasundhara Puttagunta**[2], **Martin Klein**[3], **Kakali Sarkar**[4]

[1] Agnik, LLC, 8840 Stanford Blvd. Suite 1300, Columbia, MD 21227, USA
[2] Agnik, LLC, 8840 Stanford Blvd. Suite 1300, Columbia, MD 21227, USA
[3] Agnik, LLC, 8840 Stanford Blvd. Suite 1300, Columbia, MD 21227, USA
[4] Agnik, LLC, 8840 Stanford Blvd. Suite 1300, Columbia, MD 21227, USA

Received: date / Revised version: date

Abstract This paper considers the problem of monitoring vehicle data streams in a resource-constrained environment. It particularly focuses on a monitoring task that requires frequent computation of correlation matrices using lightweight on-board computing devices. It motivates this problem in the context of the *MineFleet Real-Time* system and offers a randomized algorithm for fast monitoring of correlation (FMC), inner product, and Euclidean distance matrices among others. Unlike the existing approaches that compute all the entries of these matrices from a data set, the proposed technique works using a divide-and-conquer approach. This paper presents a probabilistic test for quickly detecting whether or not a subset of coefficients contains a significant one with a magnitude greater than a user given threshold. This test is used for quickly identifying the portions of the space that contain significant coefficients. The proposed algorithm is particularly suitable for monitoring correlation and related matrices computed from continuous data streams.

1 Introduction

Many on-board real-time data stream monitoring applications require frequent computation of statistical aggregates such as correlation, inner product, and distance matrices. Sensor networks [18], on-board scientific payloads [6] [19] and on-board vehicle mining systems [14] are some examples. This paper describes one such application where computation of such aggregates plays a critical role.

This paper presents a brief review of the MineFleet system and focuses on a particular aspect of this system that requires continuous monitoring of correlation

Send offprint requests to:

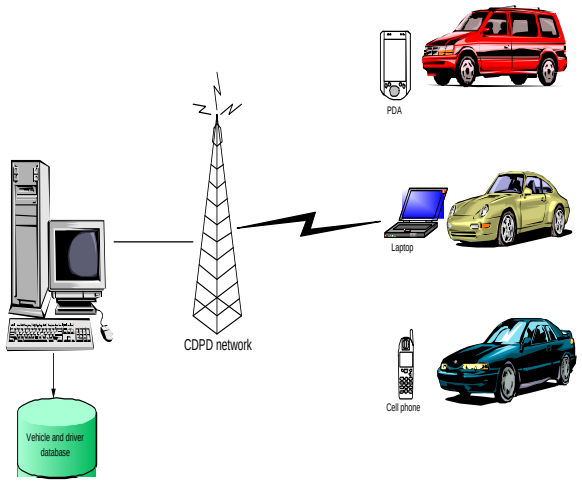

Fig. 1 A conceptual depiction of the MineFleet system.

and other related matrices for analyzing the statistical properties of the underlying system. *MineFleet Real-Time* is designed for monitoring and mining vehicle data streams in real-time. It monitors vehicle performance using on-board PDA/cell-phone or similar "light-weight" hardware-based data stream mining system and other remote desktop-based monitoring modules connected through wireless networks. MineFleet's on-board module monitors the vehicle-health and driving characteristics. This involves computing various empirical properties of the data distribution such as correlation, inner-product, and Euclidean distance matrices in a resource-constrained environment. This paper notes the need for computing these statistical aggregate matrices from continuous data streams observed on-board the *MineFleet Real-Time* system. It describes the Fast Monitoring of Correlation (FMC) matrix algorithm, a novel randomized technique for detecting changes in the correlation and other related matrices. FMC can also be used for approximately identifying the portions of these matrices that contain significantly changed coefficients.

Section 2 presents a brief overview of the *MineFleet Real-Time* system for motivating the need of correlation matrix monitoring problem in a real-life application. Section 3 relates the correlation computation problem to the MineFleet system. Section 4 formulates the problem correlation matrix computation from data streams. Section 5 discusses the related work. Sections 6 and 7 discuss the proposed technique. Section 8 presents the experimental results. Finally, Section 9 presents the conclusions and the future work.

2 MineFleet: A Vehicle Data Stream Mining and Monitoring System

MineFleet Real-Time is a mobile data stream mining environment where the resource-constrained "small" computing devices need to perform various non-trivial data

management and mining tasks on-board a vehicle in real-time. MineFleet analyzes the data produced by the various sensors present in most modern vehicles. It continuously monitors data streams generated by a moving vehicle using an on-board computing device, identifies the emerging patterns, and if necessary reports these patterns to a remote control center over a low-bandwidth wireless network connection. This section presents a brief overview of the architecture of the system and the functionalities of its different modules.

The current implementation of *MineFleet Real-Time* analyzes and monitors only the data generated by the vehicle's on-board diagnostic system and the Global Positioning System (GPS). It is currently implemented for embedded computers and mobile devices like Personal Digital Assistants, cell-phones, and hand-held computers. The overall conceptual-process-diagram of the system is shown in Figure 1. It shows multiple vehicles installed with the MineFleet software which can be concurrently monitored by a central site. The vehicles can have different types of computing devices ranging from PDA's to special-purpose tablet PCs monitoring, collecting, and analyzing the data generated by the vehicle. Any standard commercial data network can be used for the wireless communication. The *MineFleet Real-Time* system is comprised of four important components:

1. On-board hardware module: Hardware interface for the on-board diagnostic (OBD-II) data bus that couples with the MineFleet onboard software running on a PDA. A GPS device can also be connected with this module. It also offers several types of communication channels.
2. On-board data stream management and mining module: This module offers a run-time environment for performing on-board data analysis and management. The on-board module monitors, manages the data stream, and triggers actions when unusual activities are observed. The on-board module connects to the desk-top-based remote *MineFleet Control Center* through a wireless network. The system allows the fleet managers to monitor and model vehicle behavior remotely without necessarily down-loading all the data to the remote central monitoring station over the expensive wireless connection. However, if necessary, MineFleet supports analyzing data at the central control station after down-loading the data for any given period.
3. Remote control center module: The remote desktop-based control station for fleet managers. The *MineFleet Control Center* supports the following main operations: (i) interacting with the on-board module for remote management, monitoring, and mining of vehicle data streams; (ii) interactive statistical data analysis of the down-loaded data (if desired); (iii) interactive online vehicle health regime monitoring using different projection techniques (e.g. principal component analysis [13]); (iv) fuel consumption analysis; (v) visualization of the driving characteristics generated by various time series data analysis techniques. It also offers a whole range of fleet-management related services that are not directly related to the main focus of this paper.
4. Privacy management module: This module plays an important role in the implementation of the privacy policies. For example, drivers of a commercial fleet may have a quite justifiable objection against continuous monitoring of their driving behavior. However, they may be willing to allow the management to

analyze the data for detecting drunk drivers as long as the privacy of the sober drivers is not compromised.

In order to monitor the vehicle data streams using the on-board data management and mining module we need continuous computation of several statistics. For example, the MineFleet on-board system has a module that continuously monitors the spectral signature of the data which requires computation of covariance and correlation matrices on a regular basis. The on-board driving behavior characterization module requires frequent computation of similarity/distance matrices for data clustering and monitoring the operating regimes. Since the data are usually high dimensional, computation of the correlation matrices or distance (e.g. inner product, Euclidean) matrices is difficult to perform using their conventional algorithmic implementations. The incoming data sampling rate supported by the data bus limits the amount of time we get for processing the observed data. This usually means that we have only a few seconds to quickly analyze the data using the on-board hardware (when the mining is done on-board). If our algorithms take more time than what we have in hand, we cannot catch up with the incoming data rate. In order to handle this situation, we need address the following issues:

1. We need fast ”light-weight” techniques for computing and continuously monitoring various statistical aggregates. This paper particularly focuses on the correlation, covariance, inner product, and distance matrices that are frequently used in data stream mining applications. The relevance of these specific statistical aggregates will be discussed later in this paper.
2. We also need anytime algorithms [21] that will do something useful when the running time is constrained. In other words, we allow the data mining algorithm to run for a fixed amount of time and expect it to return some meaningful information. For example, we give the correlation matrix computation algorithm 1 second of CPU time for identifying the coefficients with magnitude greater than 0.7. If that time is not sufficient for computing all the correlation coefficients in the matrix then the algorithm should at least identify the portions of the matrix that may contain significant coefficients. The precision of this identification is likely to depend on the amount of computing resources allocated to the algorithm.

The following section presents a brief overview of a few of the MineFleet-modules that require frequent computation of correlation matrices on-board the vehicle using a resource-constrained environment.

3 Frequent Computation of Correlation Matrices and Operating Regime Monitoring

Frequent computation of correlation, inner product, and distance matrices plays an important role in the performance of the *MineFleet Real-Time* system. In this section, we offer a brief exposure of some of the MineFleet modules that require frequent computation of correlation matrices and monitoring changes in these matrices. First, let us consider the on-board vehicle health monitoring module of

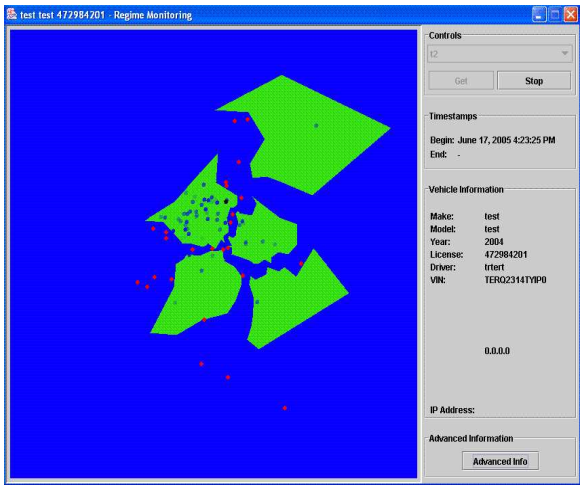

Fig. 2 Vehicle sub-system operating regime monitoring and outlier detection module. The polygons represent safe operating regimes of vehicle subsystem.

MineFleet. This module is responsible for tracking the operating characteristics of the vehicle and detecting abnormal patterns from the vehicle health data.

Among other things, this module estimates the distribution of the data using different incremental parametric and non-parametric techniques. In this section, we discuss only one of them that makes use of an incremental operating regime identification technique. It identifies the safe operating regime of the vehicle in the low dimensional eigenspace of the covariance matrix by first clustering the data and then capturing the clusters using techniques from computational geometry. We assume that initially when the vehicle is certified to be in good health condition, we can observe its behavior, gradually generate the clusters, and then use the stable cluster descriptions to define the healthy operating regimes of the vehicle. Later, during the monitoring phase, the module simply notes whether or not the observed data point falls within the safe operating regime in the projected state space of the vehicle. If it does not then the module raises a flag and reports unexpected behavior. The actual implementation of this module of MineFleet is more complex and it involves different statistical testing and confidence factor computation. However, we do not discuss those here since the focus of this paper is something else.

The main steps of this performance regime monitoring process are as follows: (1) Principal Component Analysis(PCA)-based projection of the data; (2) incremental clustering in the projected space; (3) construction of cluster descriptions using techniques from computational geometry. Figure 2 shows the two-dimensional state-space monitoring of the high dimensional feature space using the above approach. It shows a collection of polygons which represents the underlying typical operating regimes empirically constructed from the driving data. A single dot in this space represents the state of the vehicle at any given time. In this particular case, the outlier points were generated by vehicle engine cylin-

der misfiring problems. Monitoring this state space of the vehicle works under the assumption that the underlying basis set defined by the dominant eigenvectors remain invariant. When the basis vectors change we need to quickly identify that and compute the appropriate eigenvectors for representing the state space. The eigenvectors change when the correlation/covariance matrix changes. Therefore, the performance-regime learning and monitoring module must monitor the changes in the correlation matrix and re-compute it, if necessary.

MineFleet also has a module for monitoring statistical dependency among different components of a vehicle system. Usually, the correlation matrix changes when something changes in the underlying functional relationship among the attributes. Such changes introduce non-stationary properties which are often an indicator of something unusual happening in the vehicle systems.

The rest of this paper deals with this problem of resource-constrained data mining specifically in the context of frequently computing sparse correlation matrices and monitoring changes in the correlation matrices computed from different windows of data streams. However, before discussing the proposed algorithms, let us formally define the problem of computing the correlation and distance matrices and review the related literature.

4 Problem Definition

The Pearson Product-Moment Correlation Coefficient or correlation coefficient for short is a measure of the degree of linear relationship between two random variables: a and b. The correlation between a and b is commonly defined as follows: $Corr(a,b) = \frac{Cov(a,b)}{\sigma_a \sigma_b}$, where $Cov(a,b)$ is the covariance between a and b; σ_a and σ_b are the standard deviations of a and b respectively. The correlation coefficient takes a value between -1 and +1. A correlation of +1 implies a perfect positive linear relationship between the variables. On the other hand, a correlation of -1 implies a perfect negative linear relationship between the variables. A correlation coefficient that is zero implies that the two variables vary independently. In this paper, we call a correlation coefficient significant if its magnitude is greater than or equal to a user-given threshold.

In data mining applications we often estimate the correlation coefficient of a pair of features of a given data set. A data set comprised of features two X and Y, that has m observations, has m pairs (x_i, y_i) where x_i and y_i are the i-th observations of X and Y respectively. The following expression is commonly used for computing the correlation coefficient:

$$Corr(X,Y) = \frac{\sum x_i y_i - \frac{\sum x_i \sum y_i}{m}}{\sqrt{\left(\sum x_i^2 - \frac{(\sum x_i)^2}{m}\right)\left(\sum y_i^2 - \frac{(\sum y_i)^2}{m}\right)}}$$

If the data vectors have been normalized to have 0 mean and unit length (ℓ_2 norm), the resulting expression for the correlation coefficient is a lot simpler.

$$Corr(X', Y') = \sum_{i=1}^{m} x_i y_i \tag{1}$$

In the rest of this paper, we assume that the data sets have been normalized first. Therefore, if U is the $m \times n$ data matrix with the m rows corresponding to different observations and the n columns corresponding to different attributes, the correlation matrix is an $n \times n$ matrix $U^T U$. This paper also occasionally uses the term correlation-difference matrix in the context of continuous data streams. If $Corr_t(X, Y)$ and $Corr_{t+1}(X, Y)$ are the correlation coefficients computed from the data blocks observed at time t and $t+1$ respectively then the correlation-difference coefficient is defined as $|Corr_t(X, Y) - Corr_{t+1}(X, Y)|$. When there are more than two data columns corresponding to different attributes, we have a set of such coefficients that can be represented in the form of a matrix. This matrix will be called the correlation-difference matrix.

Also note that the problem of computing the Euclidean distance matrix is closely related to the correlation matrix and inner product computation problem. The Euclidean distance between the data vectors corresponding to X and Y,

$$\begin{aligned}\sum_{i=1}^{m}(x_i - y_i)^2 &= \sum_{i=1}^{m}(x_i^2 + y_i^2 - 2x_i y_i) \\ &= \sum_{i=1}^{m} x_i^2 + \sum_{i=1}^{m} y_i^2 - 2\sum_{i=1}^{m} x_i y_i.\end{aligned}$$

The correlation coefficient computation is also very similar to the problem of computing the inner product [9] [20]. Therefore, in rest of this paper we present the proposed algorithm only in the context of the correlation computation problem. The following section discusses the related work.

5 Related Work

An efficient technique for computing the correlation matrix is equally applicable to the inner product and Euclidean distance computation problem. These statistical computing primitives are directly useful for clustering, principal component analysis, and many other related statistical and data mining applications. Therefore, although the rest of the paper considers only the correlation matrix monitoring problem, the results have direct implications on solving many other related problems.

Efficient computation of the correlation matrix has been addressed in the literature. Zhu and Shasha exploited [22] an interpretation of the correlation coefficient as a measure of Euclidean distance between two data vectors in the Fourier representation. Computation of correlation coefficients in the Fourier domain is also frequently used in the signal processing literature. They developed the StatStream

system which has been applied to compute correlation matrices from continuous data streams. Their results show scalable improved performance compared to the naive way to compute the correlation coefficients. This technique is designed for desktop applications and the overhead makes it unsuitable for the resource-constrained environments where monitoring of the correlation matrix is the primary objective. Alqallaf et al. [5] considered the problem of robust estimation of the covariance matrix for data mining applications. These techniques are designed for desktop applications and the overhead is by far not appropriate for a PDA-like resource constrained device onboard the vehicle. There also exist a body of related literature on computing and monitoring aggregates (e.g. dominance norms) [7,8] from data streams.

The following section offers a novel algorithm to address this problem. Unlike the traditional correlation matrix computation approach, the algorithm presented in the following sections of this paper offer the following capabilities:

1. Quickly check whether or not the correlation matrix has changed using a probabilistic test.
2. Apply this test and a divide-and-conquer strategy to quickly identify the portions of the correlation matrix that contain the significantly changed coefficients.

6 Computing Sparse Correlation Matrices Efficiently

The main objective of this section is to develop the algorithmic foundation of the technique for Fast Monitoring of Correlation (FMC) matrix. However, the first step will be to develop an algorithm for computing sparse correlation matrices. This technique will be directly used later for the monitoring application.

Given an $m \times n$ data matrix U with m observations and n features, the correlation matrix is computed by $U^T U$ assuming that the columns of U are normalized to have zero mean and unit length. We are particularly interested in sparse correlation matrices because of the monitoring application we have in mind. However, such correlation matrices are also widely prevalant since in most real-life high dimensional applications features are not highly correlated with every other feature. Instead only a small group of features are usually highly correlated with each other. This results in a sparse correlation matrix. In most stream applications, including vehicle data stream monitoring, the difference in the consecutive correlation matrices generated from two subsequent sets of observations is usually small, thereby making the difference matrix a very sparse one.

A straight-forward approach to compute the correlation matrix using matrix multiplication takes $O(mn^2)$ number of multiplications. The objective of this section is to present FMC, a more efficient technique for computing and monitoring sparse correlation matrices. In order to achieve this, we first demonstrate how we can estimate the sum of squared values of the elements in the correlation matrix that are above the diagonal. We define this sum as $C = \sum_{1 \leq j_1 < j_2 \leq n} Corr^2(j_1, j_2)$, where $Corr(j_1, j_2) = \sum_{i=1}^{m} u_{i,j_1} u_{i,j_2}$ represents the correlation coefficient between the feature corresponding to the j_1–th and j_2–th columns of the data matrix

U. In the remainder of this section, we estimate C using an approach that is similar in spirit to that of [4].

Theorem 1 *Consider an $m \times n$ data matrix U and an n-dimensional random vector $\sigma = [\sigma_1, \sigma_2, \cdots, \sigma_n]^T$, where each $\sigma_j \in \{-1, 1\}$ is independently and identically distributed. Let v_i be a random projection of the i-th row of the data matrix U using this random vector σ, i.e.*

$$v_i = \sum_{j=1}^{n} u_{i,j}\sigma_j$$

Define a random variable $Z = (\sum_{i=1}^{m} v_i^2 - n)/2$ and $X = Z^2$. Then,

$$E[X] = \sum_{1 \le j_1 < j_2 \le n} Corr^2(j_i, j_2) = C \tag{2}$$

and

$$Var[X] \le 2C^2.$$

where $E[X]$ and $Var[X]$ represent the expectation and the variance of the random variable X, respectively.

Proof:

First note that,

$$v_i = \sum_{j=1}^{n} u_{i,j}\sigma_j$$

$$v_i^2 = \sum_{j=1}^{n} u_{i,j}^2 + 2 \sum_{1 \le j_1 < j_2 \le n} u_{i,j_1}u_{i,j_2}\sigma_{j_1}\sigma_{j_2}.$$

Because, the columns of U have been normalized to have unit norm,

$$\begin{aligned}\sum_{i=1}^{m} v_i^2 &= \sum_{i=1}^{m}\sum_{j=1}^{n} u_{i,j}^2 + 2\sum_{i=1}^{m} \sum_{1 \le j_1 < j_2 \le n} u_{i,j_1}u_{i,j_2}\sigma_{j_1}\sigma_{j_2} \\ &= \sum_{j=1}^{n}\sum_{i=1}^{m} u_{i,j}^2 + 2 \sum_{1 \le j_1 < j_2 \le n} \sigma_{j_1}\sigma_{j_2} \sum_{i=1}^{m} u_{i,j_1}u_{i,j_2} \\ &= n + 2 \sum_{1 \le j_1 < j_2 \le n} \sigma_{j_1}\sigma_{j_2} Corr(j_i, j_2).\end{aligned}$$

Now we can write,

$$\begin{aligned}X = Z^2 &= \left(\sum_{1 \le j_1 < j_2 \le n} \sigma_{j_1}\sigma_{j_2} Corr(j_i, j_2) \right)^2 \\ &= \sum_{1 \le j_1 < j_2 \le n} Corr^2(j_i, j_2) + 2 \sum_{\substack{1 \le j_1 < j_2 \le n \\ 1 \le l_1 < l_2 \le n \\ (j_1, j_2) \ne (l_1, l_2)}} \sigma_{j_1}\sigma_{j_2}\sigma_{l_1}\sigma_{l_2} Corr(j_i, j_2) Corr(l_i, l_2)\end{aligned}$$

First note that since $\sigma_j \in \{+1, -1\}$, we have $E[\sigma_j] = 0$ for all j. Further, from the basics of random variables we have: if W_1 and W_2 are independent random variables, then $E[W_1 W_2] = E[W_1]E[W_2]$. Therefore having σ_j's that are independent and from identical distribution (i.i.d), we have the following properties: $E[\sigma_{j_1}\sigma_{j_2}] = E[\sigma_{j_1}]E[\sigma_{j_2}] = 0$ for all $j_1 < j_2$ and $E[\sigma_{j_1}\sigma_{j_2}\sigma_{l_1}\sigma_{l_2}] = 0$ for at least two of j_1, j_2, l_1, l_2 are distinct. Then the expected value of X is

$$E[X] = \sum_{1 \le j_1 < j_2 \le n} Corr^2(j_i, j_2) = C \tag{3}$$

Similarly $E[\sigma_{j_1}, \sigma_{j_2}, \cdots, \sigma_{j_8}] = 0$ as long as at least two of the j's are distinct. Thus the variance of X is

$$\begin{aligned} Var[X] &= E\left[(X - E[X])^2\right] \\ &= 4 \sum_{\substack{1 \le j_1 < j_2 \le n \\ 1 \le l_1 < l_2 \le n \\ (j_1, j_2) \ne (l_1, l_2)}} Corr^2(j_1, j_2) Corr^2(l_1, l_2) \\ &\le 2C^2. \end{aligned}$$

■

We have considered σ to be a vector of ± 1's all of which are independently chosen from a uniform random distribution. This means that all the elements of σ are *fully* independent. The notion of k–wise independence (as well as *almost* k–wise independence) in randomized algorithms is becoming very important [16, 1–3]. By k–wise independence, we mean that when we pick any k positions (out of n) in the vector σ these are independently chosen with respect to each other. k–wise independent distributions have the advantage that they can be generated using very little space. 4-wise independent distributions were used in [4] for estimating the second frequency moment. The property that $E[\sigma_{j_1}\sigma_{j_2}] = 0$ for all $j_1 < j_2$ can be achieved by having σ_j's that are pair–wise independent. Similarly, the property that $E[\sigma_{j_1}\sigma_{j_2}\sigma_{l_1}\sigma_{l_2}] = 0$ for all distinct j_1, j_2, l_1, l_2 can be achieved by having σ_j's that are 4–wise independent. Note that the proof holds as long as σ_j's are 8-wise independent.

Lemma 1 provides results that lay the foundation of a probabilistic technique to estimate $\sum_{1 \le j_1 < j_2 \le n} Corr^2(j_i, j_2)$ which will be used later for testing the existence of a significant correlation coefficient. The following discussion further analyzes this approach and describes the algorithm in details.

Consider a set of $s = s_1 s_2$ random σ vectors that are from i.i.d. or 8–wise independent. Corresponding to each of the σ vectors, we maintain random variables $X_{i,j}$ using the data matrix. Let $Y_1, Y_2, \cdots, Y_{s_2}$ be such that Y_i is the mean of $X_{i,j} : 1 \le j \le s_1$. Let Y be the median of the Y_i's. We output the median Y as the estimate for C.

The idea behind doing this is the following. By considering Y_i that is a mean of s_1 random variables $X_{i,j=1:s_1}$, we reduce the variance of Y by a factor of s_1, so that $Var[Y] = Var[X]/s_1$. Then we make use of Chebyshev's inequality to bound the probabilty that Y_i deviates from the expected value by too much. By

using the median of s_2 we ensure using the Chernoff bound that the probability that more than half of the Y_i's exceed λC is arbitrarily small. Such an approach is very common in randomized algorithms (see for example [4,10,11]). Motwani and Raghavan [17] provides several techniques for randomized algorithms.

Chebyshev's inquality for a random variable W with a standard deviation $Std[W]$ is stated as,

$$Prob[|W - E[W]| \geq \lambda Std[W]] = Prob[|W - E[W]|^2 \geq \lambda^2 Var[W]] \leq 1/\lambda^2.$$

Let $s_1 = 16/\lambda^2$. Then, for each fixed p, $1 \leq p \leq s_2$

$$\begin{aligned} Prob[|Y_p - C| \geq \lambda C] &= Prob[|Y_p - C|^2 \geq \lambda^2 C^2] \\ &= Prob[|Y_p - C|^2 \geq (\lambda^2 C^2 / Var[Y_p]) Var[Y_p]] \\ &\leq \frac{Var[Y_p]}{\lambda^2 C^2} = \frac{Var[X]}{s_1 \lambda^2 C^2} \leq \frac{2C^2}{s_1 \lambda^2 C^2} = \frac{1}{8}. \end{aligned}$$

Let W_i be an indicator variable that is 1 when Y_i deviates from C by more than λC, i.e. when $|Y_i - C| \geq \lambda C$. Then $E[W_i] \leq 1/8$. Let $W = \sum_{i=1}^{s_2} W_i$. Then $E[W] \leq s_2 E[W_i] = s_2/8$. Chernoff bound is given by

$$Prob[W > (1+\delta)E[W]] < \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}} \right)^{E[W]}.$$

Let $s_2 = 2\lg(1/\epsilon)$. Then, the probability that the median of Y_i's deviates from C by more than λC, $Prob[W > s_2/2 > (1+3)E[W]]$ is at most

$$\left(\frac{e^3}{(1+3)^{(1+3)}} \right)^{E[W]} \leq \left(\frac{e^3}{(1+3)^{(1+3)}} \right)^{\lg(1/\epsilon)/4} \leq \left(\frac{e^4}{4^4} \right)^{\lg(1/\epsilon)/4} = \epsilon$$

Therefore we have the following theorem.

Theorem 2 *The probability that the median of $\{Y_1, Y_2, \cdots, Y_{s_2}\}$ where each Y_i is the mean of $\{X_{i,1}, X_{i,2}, \cdots, X_{i,s_1}\}$, deviates from C by more than λC is at most ϵ, when $s_1 = 16/\lambda^2$ and $s_2 = 2\lg(1/\epsilon)$.*

Next, we describe how this approach can be used to identify the significant coefficients of the correlation matrix. Given a set $L = \{j_1, j_2, \cdots, j_k\}$ of k–columns from the data matrix U, let U_L be the data matrix with only data column vectors from the set L, i.e. $U_L = [u_{j_1}, u_{j_2}, \cdots u_{j_k}]$, in order to detect if any of these columns are strongly correlated we first estimate C for U_L using the above approach. Let C_L be the true value of C over this pruned dataset and Y_L be the estimated value. If any of the correlation coefficients has a magnitude greater than θ then the true value of C_L must be a value greater than or equal to θ^2. We use this test to determine whether or not there are any significant correlations among the data columns in U_L. If the estimated value Y_L is less than θ^2, we declare that the columns $L = \{j_1, j_2, \cdots, j_k\}$ are not significantly correlated.

This way, the above technique can be used to design a tree-based divide and conquer strategy that first checks the possible existence of any significant correlation coefficient among a set of data columns before actually checking out every

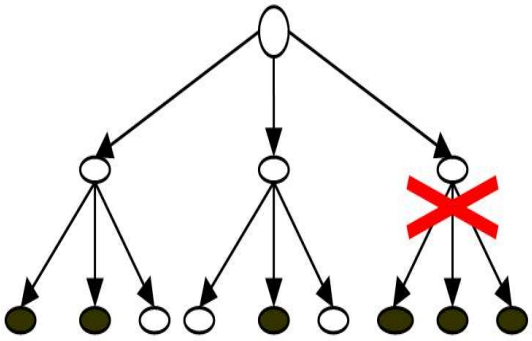

Fig. 3 A graphical representation of the divide-and-conquer-strategy-based algorithm for computing the signifi cant correlation coeffi cients.

pair-wise coefficient. If the test turns out to be negative then we discard the corresponding correlation coefficients for further consideration. Figure 3 shows the intuitive idea behind this Fast Monitoring of Correlation matrix (FMC) algorithm. The algorithm performs a tree-search in the space of all correlation coefficients. Every leaf-node of this tree is associated with a unique coefficient; every internal node a is associated with the set of all coefficients corresponding to the leaf-nodes in the subtree rooted at node a. The algorithm tests to see if the estimated $C_a \geq \theta^2$ at every node starting from the root of the tree. If the test determines that the subtree is not expected to contain any significant coefficient then the corresponding sub-tree is discarded. Search proceeds in that sub-tree otherwise. Algorithms 6.0.1–6.0.3 present the pseudo-code of FMC.

Next, we analyze the running time of the FMC algorithm. Given the motivation for energy efficient algorithms for energy constrained mobile devices, it may be important to consider the number of additions, multiplications and comparisions seperately. Therefore, we consider them seperately where ever possible.

At a node with L features the test-of-significance involves computation of v_i's for $i = 1, \cdots, m$. Because our random vectors contain only $\{+1, -1\}$, computing each v_i involves only $|L|$ additions or subtractions. Therefore the time to compute each of the Z's, is the time for $O(|L|m)$ additions and $O(m)$ multiplications. There are $s_1 s_2$ of them. Given the Z's, the time to compute estimate of C_L is the time for $O(s_1 m)$ additions and multiplications and $O(s_2 \log s_2)$ comparisons. Therefore the time to perform a test at a node with $|L|$ attributes is the time taken for $O(s_1|L|m)$ additions, $O(s_1 m)$ multiplications and $O(s_2 \log s_2)$ comparisions. Note that this is much faster than the naive approach that requires $O(m|L|^2)$ additions and multiplications.

Consider a correlation matrix with $O(c)$ number of significant coefficients. In that case, there must be exactly $O(c)$ number of leaf nodes in the corresponding tree representation, discussed earlier in this section. The depth of each of these leaf nodes is $O(\log n)$. There are $O(\log n)$ nodes along the path to each of the leaves at

Algorithm 6.0.1 The main Compute-Corr(L, Coeffs, θ) function that in turn calls Compute-Corr(L1, L2, Coeffs, θ) and Contains-Sig-Coeffs(L, θ) for comuting the significant correlation coefficients.

```
COMPUTE-CORR(L, Coeffs, θ)
  ▷ L is a list of features and Coeffs is a list of
  ▷ correlation coeffi cients whose absolute value is
  ▷ greater than θ.
  if L contains only two elements a and b
    then
        c = Correlation(a, b)
        if |c| > θ
          then Append(Coeffs, c)

  elseif Contains-Sig-Coeffs(L, θ)==true
    then
        L1 = fi rst-half(L)
        L2 = second-half(L)

        if Contains-Sig-Coeffs(L1, θ)==true
          then Compute-Corr(L1, Coeffs, θ)

        if Contains-Sig-Coeffs(L2, θ)==true
          then Compute-Corr(L2, Coeffs, θ)

        if Contains-Sig-Coeffs(L1, L2, θ)==true
          then Compute-Corr(L1 , L2, Coeffs, θ)
  return
```

which the test-of-significance is performed. Therefore there are $O(c \log n)$ nodes where the test is performed. Further, the number of features at a node that is at a depth 0 (root), $1, 2, \cdots, \log n$ is $n, n/2, \cdots, 1$ respectively. The cost of multiplications depends only on the number of nodes at which the test is carried out and not the number of features at the node. Therefore the cost of the FMC algorithm in terms of multiplications is $O(c \log n)O(s_1 m) = O(s_1 cm \log n)$. However, the number of additions carried out at a node depends on the number features at that node. In fact this is $O(s_1 m|L|)$ additions at a node with $|L|$ features. Therefore nodes closer to the root have more number of features and hence costs more number of additions. However nodes with large number of features are few in number, thus balancing out the number of additions across nodes along a path to the significant coefficient. More precisely, the number of additions that the FMC algorithm performs is $O(cs_1 nm)$. The FMC algorithm has a space requirement of $O(c \log n)$ to store the significance-test results at each of the nodes where the tests are performed. In addition we should also consider the storage requirement for storing the seeds and generating the random σ's for each of the $s_1 s_2$ Z's at any node. Note that

Algorithm 6.0.2 The Compute-Corr(L1, L2, Coeffs, θ) function.

```
COMPUTE-CORR(L1, L2, Coeffs, θ)
  L = L1 ⋃ L2
  if L contains only two elements a and b
    then
        c = Correlation(a, b)
        if |c| > θ
          then Append (Coeffs, c)
    else
        L11 = first-half(L1); L12 = second-half(L1)
        L21 = first-half(L2); L22 = second-half(L2)

        if (Contains-Sig-Coeffs(L11, L21, θ)==true)
          then Compute-Corr(L11, L21, Coeffs, θ)

        if (Contains-Sig-Coeffs(L11, L22, θ)==true)
          then Compute-Corr(L11, L22, Coeffs, θ)

        if (Contains-Sig-Coeffs(L12, L21, θ)==true)
          then Compute-Corr(L12, L21, Coeffs, θ)

        if (Contains-Sig-Coeffs(L12, L22, θ)==true)
          then Compute-Corr(L12, L22, Coeffs, θ)
  return
```

Algorithm 6.0.3 The Contains-Sig-Coeffs(L, θ) function.

```
CONTAINS-SIG-COEFFS(L, θ)
  if already-tested(L)
    then return already-tested-result(L)

  E = estimate-sum-of-Variance-square(L)
  already-tested(L) = true    ▷ to avoid estimation in future
  if E > θ^2
    then already-tested-result(L) = true      ▷ store result
    else already-tested-result(L) = false     ▷ store result

  return already-tested-result(L)
```

the standard matrix-multiplication-based technique for computing the correlation matrix requires $O(n^2)$ space and $O(mn^2)$ additions and $O(mn^2)$ multiplications.

The following section extends the FMC algorithm to the stream monitoring scenario which is the main focus of the current application.

7 Dealing with Data Streams

This section extends the sparse-correlation matrix computation technique to a stream data environment for monitoring the correlation difference matrices, the targetted application for the research reported here.

Consider a multi-attribute data stream scenario where each time stamp is associated with a window of observations from the stream data. More specifically, let $U^{(t)}$ and $U^{(t+1)}$ be the consecutive data blocks at time t and $t+1$ respectively. Let $Corr(j_1^{(t)}, j_2^{(t)})$ be the correlation coefficients between j_1–th column and j_2–th column of $U^{(t)}$ and similarly let $Corr(j_1^{(t+1)}, j_2^{(t+1)})$ be the correlation coefficients between j_1–th column and j_2–th column of $U^{(t+1)}$. Along the same lines, let $Z^{(t)}$ and $Z^{(t+1)}$ be the estimated values of Z for the two data blocks at time t and $t+1$ respectively. Let $X^{(t)}$ and $X^{(t+1)}$ be the corresponding estimated values of X. Note that we use the same σ's for computing $X^{(t)}$ as well as $X^{(t+1)}$. Let us define,

$$\Delta^{(t+1)} = Z^{(t+1)} - Z^{(t)}$$

Now we can write,

$$\begin{aligned}\Delta^{(t+1)} &= \sum_{1 \le j_1 < j_2 \le n} \sigma_{j_1}\sigma_{j_2} Corr(j_i^{(t+1)}, j_2^{(t+1)}) - \sum_{1 \le j_1 < j_2 \le n} \sigma_{j_1}\sigma_{j_2} Corr(j_i^{(t)}, j_2^{(t)}) \\ &= \sum_{1 \le j_1 < j_2 \le n} \sigma_{j_1}\sigma_{j_2} \left(Corr(j_i^{(t+1)}, j_2^{(t+1)}) - Corr(j_i^{(t)}, j_2^{(t)}) \right)\end{aligned}$$

Then we can find the expected value of $\left(\Delta^{(t+1)}\right)^2$ in a manner similar to finding $E[X]$ described earlier.

$$E[\left(\Delta^{(t+1)}\right)^2] = \sum_{1 \le j_1 < j_2 \le n} \left(Corr(j_i^{(t+1)}, j_2^{(t+1)}) - Corr(j_i^{(t)}, j_2^{(t)}) \right)^2$$

This can be used to directly look for significant changes in the correlation matrix. We should note that the difference correlation matrix (i.e. the changes in the matrix) is usually very sparse since most of the time vehicle systems do not perform unusually; rather they work following well understood principles of mechanical and electrical systems. The following section presents experimental results documenting the performance of the proposed algorithm in the context of the MineFleet application.

8 Experimental Results

This section presents the experimental results documenting the performance of the FMC algorithm using data streams from different vehicles. We primarily used a data set collected from a 2003 Ford Taurus car. This data set has 64 features and 831 rows. Whenever, we used other data sets for the experiments, we report that accordingly.

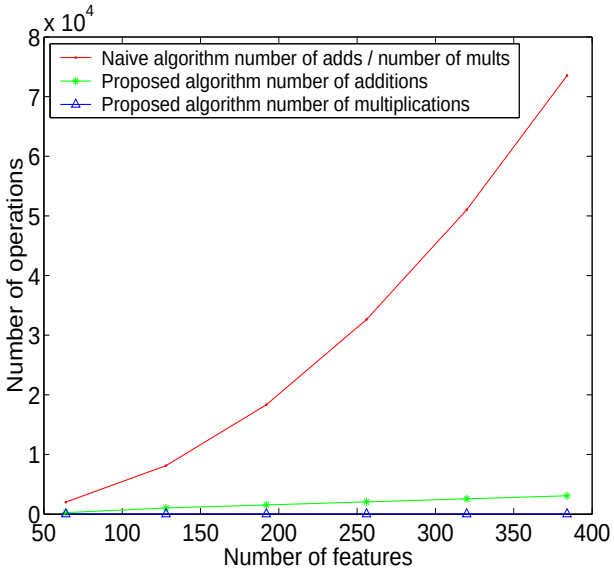

Fig. 4 Number of multiplications and additions (with a scaling factor of the number-of-data-rows) performed by the enumerative algorithm and FMC for correctly detecting no changes in the correlation matrix.

First we consider the problem of monitoring the correlation matrices computed from different data windows sampled from the data streams. Our objective is to compare the performance of the naive enumerative algorithm and the FMC in correctly detecting the following scenarios:

1. No changes in the correlation matrix over two consecutive data windows sampled from the streams.
2. No significant changes in the correlation matrix. Note that this is different from the previous scenario since in this case the correlation matrices do not stay invariant although the changes are insignificant with respect to a given threshold.
3. Detecting significant changes in the correlation matrix.

The following sections consider each of these scenarios.

8.1 Detecting No Changes

In this section our objective is to study the performance of the FMC algorithm when the correlation matrix stays invariant. We performed several experiments using data stream windows that produce the same correlation matrices. All of these experiments made use of the 831×64-dimensional Ford Taurus data set. FMC works very well, as suggested by the analytical results. It always identifies no change by performing the very first test at the root node of the tree. As a result the running time is constant compared to the quadratic order running time for

	Number of Additions		Number of Multiplications		Number of Nodes	
r	μ	σ	μ	σ	μ	σ
2	875.6	564.7	10.4	7.7	5.2	3.8
4	722.4	468.2	3.4	2.6	1.8	1.3
6	1692.6	1376.9	6	6.2	3	3.1
8	1025	0	2	0	1	0
10	1281	0	2	0	1	0

Table 1 Number of multiplications and additions (with a scaling factor of number-of-data-rows) performed by FMC for correctly detecting no signifi cant changes in the correlation matrix. Threshold value is 0.6.

the naive enumerative approach. As Figure 4 shows, the performance of FMC is significantly better than that of the naive approach.

8.2 Detecting No Significant Changes

This section considers the scenario where the correlation matrices are slightly different, resulting in a correlation-difference matrix that is not a null matrix but it does not contain any significantly changed (with respect to the given threshold) coefficients either. Table 1 shows the performance of the proposed algorithm for overlapping windows from the stream with insignificant but non-zero changes. The naive enumerative algorithm requires (2016 $\times$ number-of-data-rows) multiplications and additions. FMC detects no significant changes with approximately half the number of additions and a very small fraction of multiplications. The FMC algorithm clearly outperforms the enumerative algorithm on this ground.

8.3 Detecting Significant Changes

This section considers the problem of detecting significant changes when some of the coefficients in the correlation matrix have changed beyond the given threshold. In this situation, the algorithm has the two following goals:

1. Detect that something has indeed changed in the correlation matrix
2. Identify the portions of the matrix that are likely to contain the significantly changed coefficients.

This section first reports the results of experiments with overlapping windows of data where the difference-correlation matrix contains exactly 6 significant entries and the magnitude of the difference is greater than 0.6.

In all the experiments reported here, FMC returns the correct answer for the first problem (1) listed above. Our experiments were carried out with finite resource constraints. Table 2 shows the number of multiplications and additions when the algorithm is allowed to explore only 8 nodes in the tree. Even with this

	Number of Significant Coefficients Detected		Number of Multiplications		Number of Additions	
r	μ	σ	μ	σ	μ	σ
2	4.8	1.09	12.8	4.3	1081.6	282.6
4	5.2	1.09	16	0	2456	250.4
8	4.8	1.09	14.4	3.5	4717.6	917.6
12	4.8	0.89	14.4	3.5	7072.8	1833.5
16	4.4	0.89	14.4	3.5	9428	1833.5
20	4.4	0.89	16	0	128008	0

Table 2 Number of multiplications and additions (with a scaling factor of number-of-data-rows) performed by the naive algorithm and FMC for correctly detecting significant changes in the correlation matrix and identifying the portions of the matrix with the significantly changed coefficients.

restriction on computation, the algorithm could detect the regions of the correlation matrix with most of the significantly changed coefficients. Figure 5 shows the 64×64 dimensional correlation-difference matrix. Since the matrix is symmetric, the matrix is divided into two different regions with different gray shadings. The right-upper triangle shows four different regions (with different gray shades) corresponding to the regimes defined by the nodes of the tree constructed by the algorithm. Six bright dots in the right-upper triangle correspond to the significant entries. The lighter-shaded lower-left traingle is correctly discarded by the algorithm since it does not contain any significant entry. Note that all the six significant coefficients are covered by the nodes selected by the FMC algorithm.

Next we examine the performance of the algorithm as the number of significant changes in the correlation matrix varies. We consider vehicle data collected from a 2003 Ford Taurus. This dataset contains 831 rows and we will use 64 of its columns. We create two blocks of data, each containing the 831 rows of the Ford Taurus data. We make the correlation matrices between the two blocks different by altering some of the columns in the first block. This is performed for introducing a controlled amount of significant changes in the correlation matrices. After each alteration we run our algorithm and record the average number of additions and multiplications required as it searches for differences between the two correlation matrices corresponding to the two blocks of data. In this experiment we fix the number of random vectors to $r = 8$ and consider a coefficient significant if its magnitude is greater than 0.6. We perform these experiments using finite resource constraints in order to quickly identify regions (subtrees of the overall search tree used by FMC) of the correlation difference matrix that contain significantly changed coefficients without performing an exhaustive search of these regions for exactly identifying the changed coefficients.

Figure 6 and Figure 7 present the results of the experiment described above. The traditional correlation matrix computation requires ($2016 \times$number-of-rows) additions and multiplications. The FMC algorithm requires a fairly large number

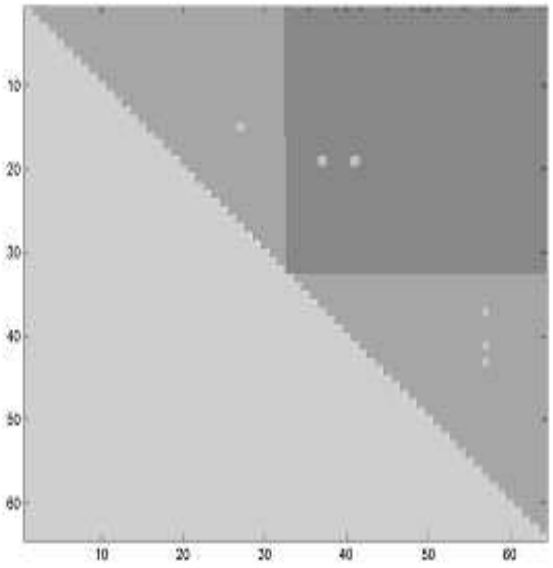

Fig. 5 This figure shows the 64×64 dimensional correlation-difference matrix. Six bright dots represent the significant entries.

of additions, but it requires considerably fewer multiplications than the naive approach. Due to the finite resource constraints, the number of additions does not increase drastically as the number of significant coefficients increases. The bounds on the numbers of additions and multiplications is fixed a priori. The goal is to identify the regimes (i.e. the sub-trees) of the significantly changed coefficients as precisely as possible. Results clearly show that the algorithm is able to locate these regions using fewer multiplications then required by the naive enumerative approach and a modest, relatively fixed number of additions.

8.4 Computing Sparse Correlation Matrices

This section considers the problem of computing sparse correlation matrices. This is relevant to the vehicle data stream mining application considered in this paper. However, it is also equally relevant to any other application where large sparse correlation matrices must be computed efficiently.

In order to study the performance of the algorithm in detecting significant coefficients from a sparse correlation matrix we first report some experiments where the sparseness can be controlled. The first set of experiments reported here used

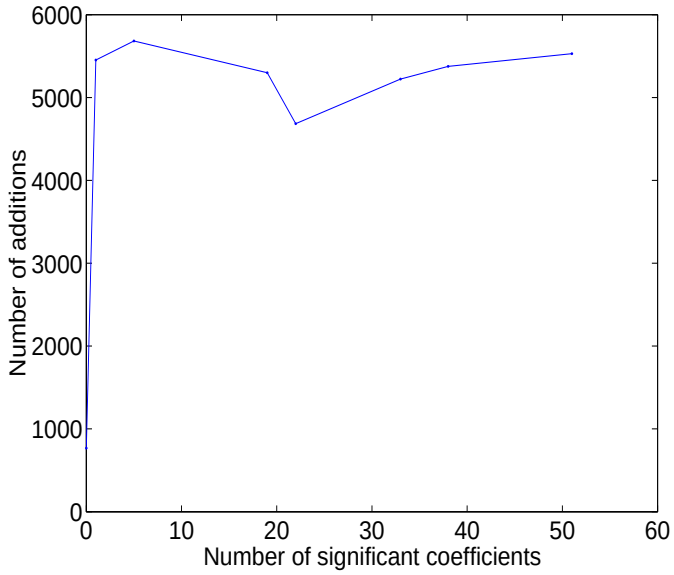

Fig. 6 Average number of additions performed by the proposed algorithm (scaled by the number of rows) vs. the number of signifi cant changes in the correlation matrix.

a controlled data set generated using correlated stock market data[1] and randomly generated uncorrelated data. More specifically we considered the highs for each day for a period of 1 year for each ticker. We used 50 randomly generated data vectors and 6 stock market data vectors in order to obtain a data set with a total of 56 attributes. In order to make a fair assessment, we shuffle the order of the features randomly. Therefore the correlation matrix is a 56×56 matrix in this case and traditional method of computing the correlation coefficients that are significant will require computing all the $n(n-1)/2$ coefficients which is 1540 in this case. We fix the threshold to be 0.49. In this case there were 10 significant coefficients. We observe the results as we increase the number of random vectors, r used. For each value of r, we ran our algorithm 25 times and observed the following:

1. The average number of significant coefficients that are found.
2. The number of times (out of the 25 trials) we obtain all the 10 significant coefficients.
3. The number of times we obtain at least 8 out of the 10 significant coefficients.
4. The number of times we obtain at least 5 of the 10 significant coefficients.
5. The average size of the tree (or the total number of nodes in the computation tree).

Next we report more detailed experimental results using a vehicle data stream and offer some more data to quantify the computational characteristics of the proposed approach. This data was collected from another 2003 Ford Taurus model.

[1] Obtained from http://kumo.swcp.com/stocks/

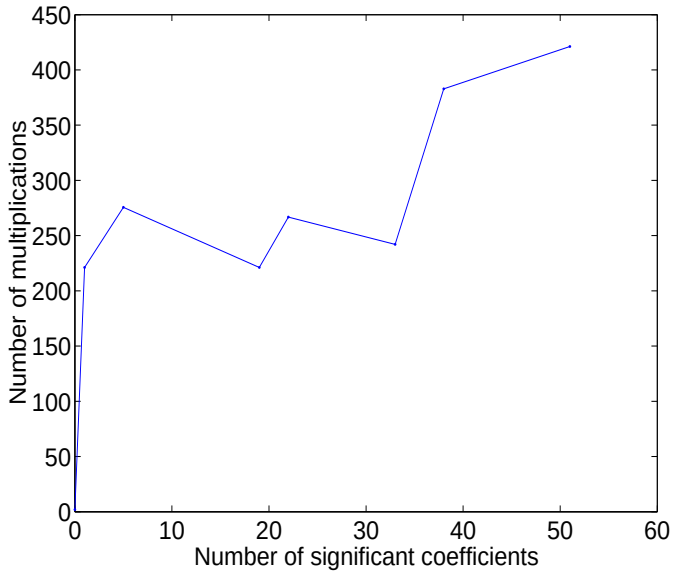

Fig. 7 Average number of multiplications performed by the proposed algorithm (scaled by the number of rows) vs. the number of signifi cant changes in the correlation matrix.

The particular data set used here has 48 features and 831 rows. We used a threshold value of 0.85. This resulted in a correlation matrix with 30 significant coefficients among a total of 1128 unique entries. In other words, the matrix contains $1128 - 30 = 1098$ insignificant coefficients. Computing all the $C = 1128$ coefficients explicitly would require $C \times m = 1128 \times 831 = 937,368$ multiplications and the same number of additions. Tables 3a and 3b present these results, the notation used in the tables is described below.

Let S be the significant number of coefficients found by the algorithm. S gives a measure of accuracy; C' be the number of exact coefficients computed at the leaves of the tree constructed by the algorithm; D be the number of nodes in the tree. Therefore, $D - C'$ gives the number of nodes in the tree where the proposed fast-estimate-test is performed. Let A and M be the total number of column additions and multiplications. Parameters M and A give a measure of how fast the method is compared to C, the number of correlation coefficients computed by the standard way. Tables 3a and 3b document the trade-off between the accuracy and the computation-cost necessary for running the FMC algorithm.

We also performed additional experiments using the vehicle data with artificially controlled feature-space and sparseness of the correlation matrix. We generated a data set using 50 random vectors and 10 real data features. The ordering of the columns is randomly chosen each time. The results are reported over 10 trials. The correlation matrix contains a total of 22 significant coefficients for a threshold value of = 0.7. The matrix contains a total of C = 60(60-1)/2 = 1770 unique coefficients. Tables 4a and 4b present the experimental results. These results also

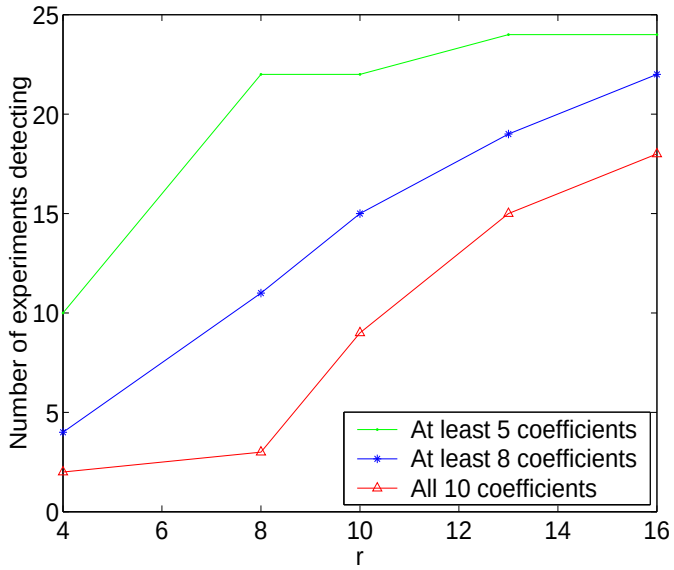

Fig. 8 Performance of the proposed algorithm in detecting the signifi cant coeffi cients. Total number of trials is 25 and the correlation matrix contains 10 signifi cant coeffi cients.

	S		D		C'	
r	μ	σ	μ	σ	μ	σ
4	11.4	4.00	304.4	90.44	131.8	45.74
8	14.6	5.46	423.3	170.15	208.0	89.65
12	17.4	6.30	478.2	133.79	244.9	71.60
16	20.2	4.40	580.2	166.53	302.1	91.21
20	21.6	5.04	629.6	171.86	333.6	89.70
24	22.9	4.41	744.6	170.85	400.5	101.09

Table 3a Performance of the proposed algorithm in detecting the signifi cant coeffi cients using the vehicle data.

clearly demonstrate that the FMC algorithm offers an approximate but relatively cheap way to identify the sparsely distributed significant entries in a correlation matrix.

Figure 9 shows the comparative running time of the proposed and the naive algorithm on a Dell Axim PDA. The experiments are performed for increasing number of features. The proposed algorithm detects no changes in the correlation matrix at a very minimal cost saving several seconds of clock time. A few seconds of saving in running time is a major achievement in a resource constrained environment like what the on-board module of MineFleet uses.

	M		A	
r	μ	σ	μ	σ
4	304.4	90.44	4712.0	1041.07
8	423.3	170.15	10582.5	3753.73
12	478.2	133.79	16691.4	4229.64
16	580.2	166.53	25871.4	6400.18
20	629.6	171.86	33793.6	8249.27
24	744.6	170.85	45739.8	8178.53

Table 3b Performance of the proposed algorithm in detecting the significant coefficients using the vehicle data.

	S		D		C'	
r	μ	σ	μ	σ	μ	σ
4	8.1	3.81	145.0	55.70	39.2	15.52
8	12.6	4.08	213.8	46.49	59.1	17.87
12	14.8	4.96	242.1	66.01	66.7	24.36
16	15.6	3.32	238.3	45.11	68.6	14.95
20	18.0	3.74	277.7	47.82	79.6	18.91
24	19.0	3.97	271.4	60.93	83.0	19.28
28	20.2	2.75	304.5	42.93	88.6	20.63

Table 4a Performance of the proposed algorithm in detecting the significant coefficients using the vehicle data with artificially controlled sparseness of the correlation matrix.

	M		A	
r	μ	σ	$\mu \times 100$	$\sigma \times 100$
4	145.0	55.70	41.6	14.0
8	213.8	46.49	115.4	19.9
12	242.1	66.01	186.0	35.4
16	238.3	45.11	244.8	38.3
20	277.7	47.82	342.3	41.9
24	291.1	48.41	415.3	60.8
28	314.7	34.91	520.4	39.7

Table 4b Performance of the proposed algorithm in detecting the significant coefficients using the vehicle data with artificially controlled sparseness of the correlation matrix.

9 Conclusions

This paper presented a brief overview of *MineFleet Real-Time*, a vehicle data stream mining and monitoring system. MineFleet is one among the first data stream mining systems that is designed for mobile applications. We believe that this technology will find many other applications in different domains where resource-constrained monitoring of time-critical data streams is important and central collection of data is an expensive proposition.

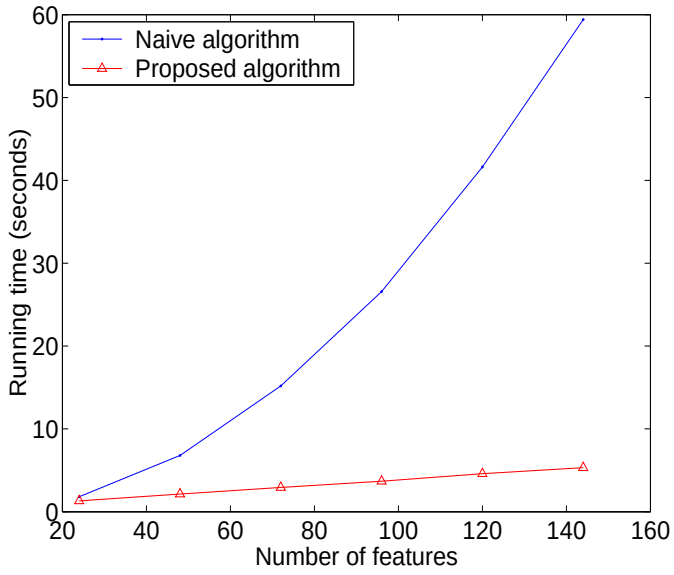

Fig. 9 Comparison of the running time of the proposed and the naive algorithm for detecting no changes in the correlation matrix.

The paper mainly focused on a particular aspect of the MineFleet system—monitoring correlation and distance matrices. It offered a probabilisitc technique for efficiently detecting changes in the correlation matrix and identifying portions of the matrix that are likely to contain the significantly changed coefficients. MineFleet contains a proprietary version of this algorithm and it plays a critical role in the real-time performance of the vehicle on-board module for data analysis.

The proposed technique adopts a divide-and-conquer strategy that makes use of a test to check whether or not a subset of correlation coefficients contains any significant coefficient. The test allows us to prune out those subsets of coefficients that do not appear to contain any significant one. The technique is particularly suitable for efficiently monitoring changes in coefficient matrices and computing large sparse correlation matrices. The proposed algorithm made a tangible difference in the performance of the MineFleet system.

Our preliminary experimental results with controlled and real-life vehicle stream data appear promising. However, there are several issues that need to be addressed in order to make the performance of the proposed algorithm more attractive. As we noted earlier, the accuracy of the algorithm depends on the value of r, i.e. number of different randomized trials. For relatively larger values of r, the accuracy is usually excellent; however, the running time goes up accordingly. Therefore, it will be nice if we can construct a deterministic version of the test that does not require multiple trials at every node of the tree. The performance of the algorithm also depends on the overhead necessary for maintaining and manipulating the tree structure. Therefore, paying attention to the systems issues is important,

particularly for the run-time performance on-board a PDA-like device. The current implementation of the technique is primarily designed for monitoring changes in the correlation matrix. Once we identify that the matrix has significantly changed it is usually better to use the naive correlation coefficient computation technique to generate the new matrix exactly. The algorithm also seems to work well when the matrix is not significantly changing frequently. In general, if the application requires continuous low-overhead monitoring of occasionally changing correlation or distance matrices then FMC appears to be a quite appropriate choice.

Acknowledgments

The authors would like to acknowledge partial support from United States Air Force Grant F33615-03-M-4120 for performing this research.

References

1. N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986.
2. N. Alon, O. Goldreich, J. Hastad, and R. Peralta. Simple constructions of almost k-wise independent random variables. In *IEEE Symposium on Foundations of Computer Science*, pages 544–553, 1990.
3. N. Alon, O. Goldreich, and Y. Mansour. Almost k-wise independence versus k-wise independence. *Inf. Process. Lett.*, 88(3):107–110, 2003.
4. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *In Proceedings of the ACM Symposium on Theory of Computing*, pages 20–29, 1996.
5. F. Alqallaf, K. Konis, R. Martin, and R. Zamar. Scalable robust covariance and correlation estimates for data mining. In ACM Press, editor, *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 14–23, 2002.
6. S. Chien, T. Debban, C. Yen, R. Sherwood, R. Castano, B. Cichy, A. Davies, M. Burl, A. Fukunaga, R. Greeley, T. Doggett, K. Williams, V. Baker, and J. Dohm. Revolutionary deep space science missions enabled by onboard autonomy. *International Symposium on Artifi cial Intelligence, Robotics, and Automation in Space (i-SAIRAS)*, 2003.
7. G. Cormode and S. Muthukrishnan. Estimating dominance norms of multiple data streams. Technical report, DIMACS, 2002. DIMACS TR 2002-35.
8. G. Cormode and S. Muthukrishnan. What is new: Finding signifi cant differences in network data streams. In *Proceedings of the INFOCOM04*, 2004.
9. R. Falk and A. Well. Many faces of the correlation coeffi cient. *Journal of Statistics Education*, 5(3), 1997.
10. J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate l^1 - difference algorithm for massive data streams. In *IEEE Symposium on Foundations of Computer Science*, pages 501–511, 1999.
11. S. Ganguly. Estimating frequency moments of data streams using random linear combinations. In *APPROX-RANDOM*, pages 369–380, 2004.
12. D. L. Hall and D. Culler. *Handbook of Multi-Sensor Data Fusion*, 2001.

13. H. Hotelling. Relation between two sets of variants. *Biometrika*, 28:322–377, 1936.
14. H. Kargupta, R. Bhargava, K. Liu, M. Powers, P. Blair, S. Bushra, J. Dull, K. Sarkar, M. Klein, M. Vasa, and D. Handy. Vedas: A mobile and distributed data stream mining system for real-time vehicle monitoring. In *Proceedings of the SIAM International Data Mining Conference, Orlando*, 2004.
15. H. Kargupta and K. Sivakumar. *Existential pleasures of distributed data mining, Next Generation Data Mining: Future Directions and Challenges*. MIT/AAAI Press, 2004.
16. M. Luby. A simple parallel algorithm for the maximal independent set problem. In *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 1–10. ACM Press, 1985.
17. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
18. G. Pottie and W. Kaiser. Embedding the internet: wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.
19. A. N. Srivastava and J. Stroeve. Onboard detection of snow, ice, clouds, and other geophysical processes using kernel methods. In *Proceedings of the ICML 2003 Workshop on Machine Learning Technologies for Autonomous Space Sciences*, 2003.
20. K. L. Weldon. A simplifi ed introduction to correlation and regression. *Journal of Statistics Education*, 8(3), 2000.
21. S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.
22. Y. Zue and D. Shasha. Statistical monitoring of thousands of data streams in real time. In *In Proceedings of the 28th VLDB Conference*, Hong Kong, China, 2002.