AFFINITY: Efficiently Querying Statistical Measures on Time-Series Data

Saket Sathe
EPFL, Switzerland
saket.sathe@epfl.ch

Karl Aberer
EPFL, Switzerland
karl.aberer@epfl.ch

***Abstract*— Computing statistical measures for large databases of time series is a fundamental primitive for querying and mining time-series data [1]–[6]. This primitive is gaining importance with the increasing number and rapid growth of time series databases. In this paper, we introduce a framework for efficient computation of statistical measures by exploiting the concept of *affine relationships*. Affine relationships can be used to infer statistical measures for time series, from other related time series, instead of computing them directly; thus, reducing the overall computational cost significantly. The resulting methods exhibit at least one order of magnitude improvement over the best known methods. To the best of our knowledge, this is the first work that presents an unified approach for computing and querying several statistical measures at once.**

Our approach exploits affine relationships using three key components. First, the AFCLST algorithm clusters the time-series data, such that high-quality affine relationships could be easily found. Second, the SYMEX algorithm uses the clustered time series and efficiently computes the desired affine relationships. Third, the SCAPE index structure produces a many-fold improvement in the performance of processing several statistical queries by seamlessly indexing the affine relationships. Finally, we establish the effectiveness of our approaches by performing comprehensive experimental evaluation on real datasets.

I. INTRODUCTION

In the recent years we are experiencing a dramatic increase in the amount of available time-series data. This development calls for scalable data management techniques that enable efficient querying and analysis of large amounts of time-series data in real-time and archival settings. Primary sources of time-series data are sensor networks, medical monitoring, financial applications, news feeds and social networking applications. A typical processing need on such data is statistical querying and mining in order to analyze trends and detect interesting correlations. In this paper, we propose the AFFINITY framework that supports efficient processing of statistical queries on large time-series databases, based on the use of *affine relationships* among different time series. Before rigorously developing the technical approaches, let us, in the following, introduce the concept of affine relationships and motivate why they are a powerful tool to improve efficiency of statistical querying over time-series data.

Computing statistical measures.

An important challenge concerning time-series data processing is computing and storing statistical measures. For example, stock traders frequently compute the correlation coefficient between stocks as follows [7]–[10]:

Problem 1: Given the intra-day stock quotes of n stocks obtained at a sampling interval Δt, return the correlation coefficients of the $\frac{n(n-1)}{2}$ pairs of stocks on a given day.

As an example, let us consider daily time series of three stocks (i.e., $n = 3$), Intel Corporation (INTC), Advanced Micro Devices (AMD) and Microsoft Corporation (MSFT) on 2^{nd} January 2003 (refer Fig. 1). Let us denote the stock price at time i of INTC, AMD and MSFT as s_{i1}, s_{i2} and s_{i3} respectively, where $1 \leq i \leq m$. Using the integers 1, 2, and 3 to identify the time series $\boldsymbol{s}_1$, $\boldsymbol{s}_2$ and $\boldsymbol{s}_3$[1], we can form three pairs of the time series: $(1,2)$, $(2,3)$ and $(1,3)$. A naïve approach for solving Problem 1 is to compute the correlation coefficients for all the pairs of stocks for the requested day. Clearly, for high values of n this method is not scalable, since it computes $\frac{n(n-1)}{2}$ correlation coefficients from scratch, which are in the order of $\mathcal{O}(n^2)$.

The first idea that this paper proposes in order to enhance the naïve approach is to exploit *affine relationships* between pairs of time-series data. For every $1 \leq i \leq m$, an affine relationship between pairs $(1,3)$ and $(2,3)$ is defined by using an affine transformation:

$$\begin{pmatrix} s_{i2} \\ s_{i3} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} s_{i1} \\ s_{i3} \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, \quad (1)$$
$$= \mathbf{A}_e \begin{pmatrix} s_{i1} \\ s_{i3} \end{pmatrix} + \boldsymbol{b}_e.$$

[1] $\boldsymbol{s}_1 = (s_{11}, s_{21}, \ldots, s_{m1})^\top$ is a vector of size m-by-1. Similarly for $\boldsymbol{s}_2$ and $\boldsymbol{s}_3$.

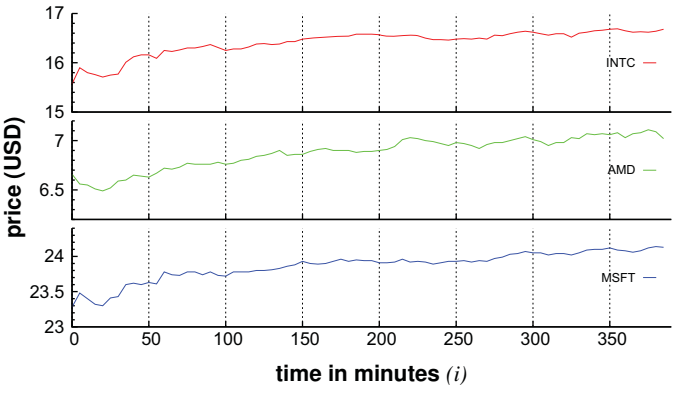


Fig. 1. Stock prices for INTC, AMD and MSFT on 2^{nd} January 2003.

This work was supported by NCCR-MICS (http://www.mics.org), a center supported by the Swiss National Science Foundation under grant number 5005-67322, and by the OpenSense project (reference number 839_401) supported by the Nano-Tera initiative (http://www.nano-tera.ch).

The matrix $\mathbf{A}_e$ is known as the *transformation matrix* and the vector $\boldsymbol{b}_e$ represents a translation. Let us assume for the moment that the relationship between pairs of time series can be described at all time instants i using the same affine relationship. Obviously, as it can be seen from Fig. 1, this is not true, but we will deal with this issue subsequently. Then, given the affine relationship in Eq. (1), the correlation coefficient between a *related* pair $(2, 3)$ could be computed directly from the correlation coefficient between pair $(1, 3)$, without accessing the time series. Concretely, consider the covariance matrix for the pair $(1, 3)$, denoted as Σ_{13} and defined as:

$$\Sigma_{13} = \begin{pmatrix} \sigma_1^2 & \rho_{13}\sigma_1\sigma_3 \\ \rho_{13}\sigma_1\sigma_3 & \sigma_3^2 \end{pmatrix}, \tag{2}$$

where ρ_{13} denotes the correlation coefficient between time series $\boldsymbol{s}_1$ and $\boldsymbol{s}_3$, similarly σ_1^2 and σ_3^2 are the variances of the time series $\boldsymbol{s}_1$ and $\boldsymbol{s}_3$ respectively. Now, given the following two inputs: transformation matrix $\mathbf{A}_e$ from Eq. (1) and covariance matrix Σ_{13} from Eq. (2), we can compute the desired correlation coefficient ρ_{23} as follows [11]:

$$\rho_{23} = \frac{\boldsymbol{a}_1^\top \Sigma_{13} \boldsymbol{a}_2}{\sqrt{\boldsymbol{a}_1^\top \Sigma_{13} \boldsymbol{a}_1 \cdot \boldsymbol{a}_2^\top \Sigma_{13} \boldsymbol{a}_2}}, \tag{3}$$

where $\boldsymbol{a}_1 = (a_{11}, a_{21})^\top$ and $\boldsymbol{a}_2 = (a_{12}, a_{22})^\top$.

It is important to observe the following two advantages regarding the computation of ρ_{23} using Eq. (3): first, the computation for ρ_{23} is significantly more efficient as compared to its computation using the original time series $\boldsymbol{s}_2$ and $\boldsymbol{s}_3$ [11]; second, since we do not need the original time series $\boldsymbol{s}_2$ and $\boldsymbol{s}_3$, we require significantly lower memory for computing ρ_{23}. In Section VI, we experimentally demonstrate that these advantages manifest a many-fold increase in performance.

Similarly, many other measures of correlation and similarity, can be computed using affine relationships. Thus, by utilizing affine relationships, our approach provides an elegant and highly-efficient solution for computing and querying a wide range of statistical measures. As a consequence, our methods not only increase the efficiency of computing the correlation coefficient, but of many other statistical measures.

Measuring quality of affine relationships.
Now, let us turn our attention to the issue that exact affine relationships are unlikely to occur over longer real-world time series. However, we have found that, in practice, such relationships hold approximately, when time series are strongly correlated. For illustration, let us come back to the three stocks from our introductory example. We can compute an approximate affine relationship $\mathbf{A}_e = \left(\begin{smallmatrix} 0.75 & -0.3 \\ 0 & 1 \end{smallmatrix}\right)$ and $\boldsymbol{b}_e = \left(\begin{smallmatrix} 1.6 \\ 0 \end{smallmatrix}\right)$. This relationship is highly accurate between time 150 and 200, but produces errors between time 0 and 50.

Therefore, for characterizing such approximation errors we propose a distance metric, *Least Significant Frobenius Distance (LSFD)*, which takes as input the values from stocks $\boldsymbol{s}_1$, $\boldsymbol{s}_2$, and $\boldsymbol{s}_3$ in a specific time window and quantitatively judges the quality of affine relationships. Additionally, we propose the AFCLST clustering algorithm that uses the LSFD metric for clustering the time series, such that high-quality affine relationships could be found for computing statistical measures.

Indexing affine relationships.
Consider a slightly modified version of Problem 1, where a stock trader is interested to find all pairs of stocks that have correlation coefficient greater than τ. One way of evaluating this query is to compute – either from scratch or using affine relationships – the correlation coefficient for all the $\frac{n(n-1)}{2}$ pairs, and then return the pairs having correlation coefficient greater than τ. This approach, again, is not scalable for large values of n.

For circumventing the computation of all the pairwise correlation coefficients we propose to index the affine relationships. We call this index the SCAPE index. Before indexing, the SCAPE index establishes a way of ordering the affine relationships. Such an ordering eliminates unnecessary computation and directly gives us the pairs having correlation coefficient greater than τ. Notably, the ordering established by the SCAPE index holds for many statistical measures. As a result, the SCAPE index can be used for querying them simultaneously.

Contributions.
To the best of our knowledge, this is the first work that exploits multi-dimensional affine transformations for time-series data management. The fundamental contribution of this paper is the introduction of affine relationships for efficiently querying and computing several statistical measures. Compared to the existing state of the art methods [1], [3], which use the Discrete Fourier Transform (DFT) to approximate the correlation coefficient, our methods use affine relationships that are amenable to indexing; thus resulting in orders of magnitude performance improvement over the state of the art methods. Furthermore, our methods are more general and can be used for computing many other statistical measures, with even better performance gains than the DFT-based computation of the correlation coefficient.

Overall, this paper makes the following contributions:

- We propose a distance metric (i.e., LSFD) for characterizing the quality of affine relationships.
- We present a novel clustering algorithm (i.e., AFCLST) capable of clustering the given data, such that high-quality (low LSFD) affine relationships could be found within the cluster members.
- We introduce the SYMEX algorithm for efficiently generating the affine relationships on-the-fly, by utilizing the output of the AFCLST clustering algorithm.
- We show that using the SCAPE index for indexing affine relationships results in orders of magnitude performance improvement for processing statistical queries.
- We extensively evaluate our methods by performing experiments on two real datasets.

We begin by presenting the details of the AFFINITY framework in Section II. In Section III, we propose the LSFD metric and the AFCLST clustering algorithm for finding high-quality affine relationships in time-series data. In Section IV,

we introduce the SYMEX algorithm for generating affine relationships. While in Section V, we propose the SCAPE index for indexing affine relationships. Lastly, comprehensive experiments are presented in Section VI, followed by a discussion on extensions and future work in Section VII, and the review of related studies in Section VIII.

II. Foundation

In this section we define the basic concepts and establish the notation used in the rest of the paper (refer Table I). Most importantly, we discuss the notion of affine transformations and examine their properties. We, then, define the queries that are processed by the AFFINITY framework.

Framework Overview.
Fig. 2 shows the architecture of the AFFINITY framework. It consists of various time series, like, financial market data, RSS news feeds, sensor network data, etc., that are being stored using a DBMS. The AFFINITY framework consists of two key components: the affine relationships and the SCAPE index. The affine relationships are inferred using the data_matrix table, and are indexed using the SCAPE index.

Let us assume that the AFFINITY framework has n time series and m values per time series, which are stored in the data_matrix table. We compose a matrix consisting of m rows by concatenating the n column vectors as $\mathbf{S} = [\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n] \in \mathbb{R}^{m \times n}$. We refer to matrix $\mathbf{S}$ as the *data matrix*.

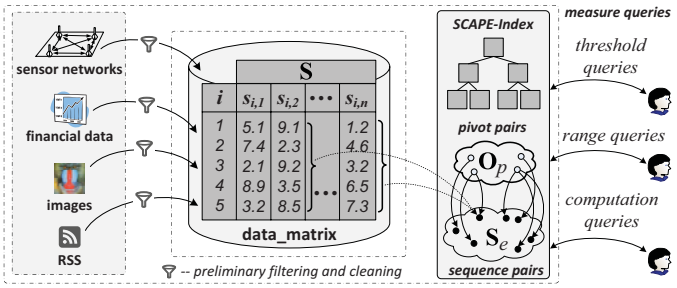


Fig. 2. Architecture of the AFFINITY framework.

A. Statistical Measures

In this paper, we consider three popular classes of statistical measures. The first type of measures are the *location measures* or $\mathcal{L}$-measures that define the central tendency of data (e.g., mean, median, etc.). The second type of measures characterize the joint or pairwise variability in the data and are called *dispersion measures* or $\mathcal{T}$-measures (e.g., covariance, dot product, etc.). The third type are the *derived measures* or $\mathcal{D}$-measures that are derived by normalizing a dispersion measure; for example, the correlation coefficient is derived by normalizing the covariance.

The $\mathcal{T}$- and $\mathcal{D}$-measures considered in this paper are required to be computed on pairs of time-series data. Thus, for conveniently identifying the time series in such scenarios, we define the following two sets. Let $\mathcal{I} = \{u | 1 \leq u \leq n\}$ be the set containing integers $1, 2, \ldots, n$ that identify the time series $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n$ respectively. We refer to $\mathcal{I}$ as the *series identifier set* and each of its elements as a *series identifier*. Similarly, let $\mathcal{P} = \{e = (u, v) | u < v \text{ and } e \in \mathcal{I} \times \mathcal{I}\}$ be the set containing unique pairs of series identifiers. We refer to $\mathcal{P}$ as the *sequence pair set* and each of its elements as a *sequence pair*.

TABLE I
SUMMARY OF NOTATIONS.

Symbol	Description
$\mathbf{A}, \ldots$	Matrices (uppercase boldface)
a_{ij}	Entry at row i and column j of matrix $\mathbf{A}$
$\boldsymbol{x}$ or $\boldsymbol{x}_1$	Column vectors (lowercase boldface)
x_i or x_{i1}	element i of a vector $\boldsymbol{x}$ or $\boldsymbol{x}_1$ respectively
$\mathbf{S}$	Data matrix of size $m \times n$
$\mathcal{L}(\mathbf{S}), \mathcal{T}(\mathbf{S}), \mathcal{D}(\mathbf{S})$	Location, dispersion, and derived measures
e, p	Sequence pair and pivot pair
$\mathbf{S}_e$, $\mathbf{O}_p$	Sequence pair matrix and pivot pair matrix
$\mathbb{R}^n$	Set of n-dimensional real column vectors
$\mathbb{R}^{m \times n}$	Set of m-by-n real matrices
$[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_w]$	Column-wise concatenation of w vectors

A sequence pair is used for uniquely identifying a pair of time series in the data matrix $\mathbf{S}$. Furthermore, the matrix that is formed by concatenating the time series defined by the sequence pair $e = (u, v) \in \mathcal{P}$ is known as the *sequence pair matrix* and is denoted as $\mathbf{S}_e = [\boldsymbol{s}_u, \boldsymbol{s}_v]$, $\mathbf{S}_e \in \mathbb{R}^{m \times 2}$.

We denote the $\mathcal{L}$-, $\mathcal{T}$-, and $\mathcal{D}$-measures of the matrix $\mathbf{S}$ as $\mathcal{L}(\mathbf{S})$, $\mathcal{T}(\mathbf{S})$ and $\mathcal{D}(\mathbf{S})$ respectively. Here, $\mathcal{L}(\mathbf{S})$ is a vector of size n, and $\mathcal{T}(\mathbf{S})$ and $\mathcal{D}(\mathbf{S})$ are symmetric matrices of size $n \times n$. In the matrix $\mathcal{T}(\mathbf{S})$, the entry found at row u and column v is the dispersion measure between the time series $\boldsymbol{s}_u$ and $\boldsymbol{s}_v$ of the matrix $\mathbf{S}$. The entry found at position (u, v) of $\mathcal{T}(\mathbf{S})$ is denoted as $\mathcal{T}_{uv}(\mathbf{S})$, and since $e = (u, v)$ we also denote it as $\mathcal{T}_e(\mathbf{S})$. Furthermore, $\mathcal{T}_e(\mathbf{S})$ is equal to $\mathcal{T}_{12}(\mathbf{S}_e)$, that is the entry at position (u, v) in the matrix $\mathcal{T}(\mathbf{S})$ is equal to the entry at position (1,2) of the matrix $\mathcal{T}(\mathbf{S}_e)$. Notations similar to the $\mathcal{T}$-measure are also used for the $\mathcal{D}$-measure; for example, the derived measure between the time series u and v is denoted as $\mathcal{D}_{uv}(\mathbf{S})$, and so on.

In this paper, we consider three $\mathcal{L}$-measures: mean, mode, and median. We consider two $\mathcal{T}$-measures: the covariance matrix and the dot product matrix, which are of size n-by-n and are denoted as $\Sigma(\mathbf{S})$ and $\Pi(\mathbf{S})$ respectively. We also consider one $\mathcal{D}$-measure: the (Pearson) correlation coefficient matrix, denoted as $\rho(\mathbf{S})$. In all these notations, subscripts are used to denote specific entries. For example, $\Pi_{uv}(\mathbf{S})$ denotes the dot product between time series $\boldsymbol{s}_u$ and $\boldsymbol{s}_v$, and $\mathcal{L}_u(\mathbf{S})$ denotes a location measure of the time series $\boldsymbol{s}_u$.

Moreover, all the approaches proposed in this paper are also applicable to a large number of other derived measures that are derived by normalizing the dot product. Examples of such measures include Jaccard coefficient, Dice coefficient, cosine similarity, harmonic mean, etc.

B. Query Types

The AFFINITY framework considers three important and frequently-used statistical queries that are posed on time-series data. Since our approach supports many statistical measures

simultaneously, we generalize the queries by making them independent of the statistical measures.

The first query computes a given statistical measure over a requested set of time series, we define this query as follows:

Query 1: **Measure computation (MEC) query.** Given a set of series identifiers $\psi \subseteq \mathcal{I}$ and a $\mathcal{L}$-, $\mathcal{T}$-, or $\mathcal{D}$-measure, the *measure computation query* returns the value of the given statistical measure for the time series identified by ψ.

For the $\mathcal{T}$- and $\mathcal{D}$-measures, the response of a MEC query is a $|\psi|$-by-$|\psi|$ matrix, and for $\mathcal{L}$-measures the response is a vector of size $|\psi|$.

Query 2: **Measure threshold (MET) query.** Given a statistical measure $\mathcal{L}$ ($\mathcal{T}$ or $\mathcal{D}$) and the user-defined threshold τ. The *measure threshold query* returns the set Λ_T consisting of the series identifiers u (sequence pairs e) where $\mathcal{L}_u(\mathbf{S})$ ($\mathcal{T}_e(\mathbf{S})$ or $\mathcal{D}_e(\mathbf{S})$) is greater or lesser than the threshold τ.

The third query is a range query adaptation of Query 2. We define it as follows:

Query 3: **Measure range (MER) query.** Given a statistical measure $\mathcal{L}$ ($\mathcal{T}$ or $\mathcal{D}$) and the user-defined lower and upper bounds τ_l and τ_u respectively. The *measure range query* returns the set Λ_R of the series identifiers u (sequence pairs e) where $\mathcal{L}_u(\mathbf{S})$ ($\mathcal{T}_e(\mathbf{S})$ or $\mathcal{D}_e(\mathbf{S})$) is in between the lower bound τ_l and upper bound τ_u.

An example of the above query could be: *return all sequence pairs where the covariance is in between τ_l and τ_u.*

C. Affine Transformations

Consider any two m-by-2 matrices $\mathbf{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2]$ and $\mathbf{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2]$, where $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{y}_1, \boldsymbol{y}_2$ are column vectors of size m. Then, an affine transformation between $\mathbf{X}$ and $\mathbf{Y}$ is defined as:

$$\mathbf{Y} \triangleq \mathbf{X}\mathbf{A} + \mathbb{1}_m \boldsymbol{b}^\top, \tag{4}$$

where $\mathbf{A} \in \mathbb{R}^{2\times 2}$ is non-singular, $\boldsymbol{b} \in \mathbb{R}^2$, and $\mathbb{1}_m = (1, 1, \ldots, 1)^\top \in \mathbb{R}^m$ (refer Fig. 3). We denote the above affine transformation as $(\mathbf{A}, \boldsymbol{b})$. In addition, we denote the first and second column of $\mathbf{A}$ as $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ respectively. We refer to $\mathbf{X}$ as the *source pair matrix* and $\mathbf{Y}$ as the *target pair matrix*. The difference between an affine transformation and a linear transformation is that an affine transformation is a combination of a linear transformation ($\mathbf{A}$) and a translation ($\boldsymbol{b}$). Therefore, an affine transformation can be considered as a generic form of a linear transformation.

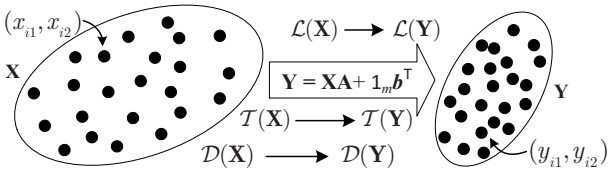


Fig. 3. Illustration of an affine transformation.

Interestingly, all the statistical measures that we consider are well-behaved under the action of an affine transformation [11]. Concretely, given the location measure $\mathcal{L}(\mathbf{X})$ of the source pair matrix $\mathbf{X}$, $\mathcal{L}(\mathbf{Y})$ can be computed as:

$$\mathcal{L}(\mathbf{Y})^\top = \mathcal{L}(\mathbf{X})^\top \mathbf{A} + \boldsymbol{b}^\top. \tag{5}$$

Similarly, the covariance and the dot product are also well-behaved under the action of an affine transformation. Given the covariance matrix $\Sigma(\mathbf{X})$, $\Sigma(\mathbf{Y})$ can be computed as follows:

$$\Sigma(\mathbf{Y}) = \mathbf{A}^\top \Sigma(\mathbf{X})\mathbf{A}, \qquad \Sigma_{12}(\mathbf{Y}) = \boldsymbol{a}_1^\top \Sigma(\mathbf{X})\boldsymbol{a}_2. \tag{6}$$

The dot product is well-behaved under the action of an affine transformation as follows [11]:

$$\Pi_{12}(\mathbf{Y}) = \boldsymbol{a}_1^\top \cdot \Pi(\mathbf{X}) \cdot \boldsymbol{a}_2 + \boldsymbol{b}^\top \mathbf{A}^\top \begin{pmatrix} h_1(\mathbf{X}) \\ h_2(\mathbf{X}) \end{pmatrix}, \tag{7}$$

where $h_1(\mathbf{X}) = \sum_{i=1}^m x_{i1}$, $h_2(\mathbf{X}) = \sum_{i=1}^m x_{i2}$.

The $\mathcal{D}$-measures are derived by normalizing one of the $\mathcal{T}$-measures. For example, the correlation coefficient is derived by normalizing the covariance as follows:

$$\rho_{12}(\mathbf{Y}) = \frac{\Sigma_{12}(\mathbf{Y})}{\mathcal{U}_{12}}, \quad \rho_{12}(\mathbf{Y}) = \frac{\boldsymbol{a}_1^\top \cdot \Sigma_{12}(\mathbf{X}) \cdot \boldsymbol{a}_2}{\mathcal{U}_{12}}, \tag{8}$$

where $\mathcal{U}_{12}$ is the normalizer and is equal to $\sqrt{\Sigma(\boldsymbol{y}_1)\Sigma(\boldsymbol{y}_2)}$. Observe that the normalizer is separable: $\Sigma(\boldsymbol{y}_1)$ and $\Sigma(\boldsymbol{y}_2)$ can be computed separately. Thus, we simply compute and store $\Sigma(\boldsymbol{y}_1)$ and $\Sigma(\boldsymbol{y}_2)$ separately and then combine them to form $\mathcal{U}_{12}$ as required. We denote the normalizer of the sequence pair e as $\mathcal{U}_e$.

Recall that all the above properties assume that the affine transformation $(\mathbf{A}, \boldsymbol{b})$ perfectly (i.e., with zero error) transforms $\mathbf{X}$ into $\mathbf{Y}$. As discussed in Section I, it is rarely possible to find a perfect affine transformation. To rectify this problem, in the next section, we propose clustering techniques that enhance the quality of affine transformations.

III. Affine Clustering

The naïve approach for computing the covariance for all the sequence pairs is to compute it from scratch. But, computing covariances from scratch is inefficient because it requires scanning of all the sequence pair matrices $\mathbf{S}_e$, which are of order $\mathcal{O}(n^2)$, where n is the number of time series.

We reduce the $\mathcal{O}(n^2)$ complexity by selecting a nearly-linear number of time series pairs, which are called the *pivot pairs*, and the m-by-2 matrices formed by them are called the *pivot pair matrices*; we will shortly describe the selection procedure for the pivot pairs. Then, we compute the covariance for all the pivot pair matrices and determine the affine transformations between each sequence pair matrix and one of the pivot pair matrix. Next, with the help of Eq. (6) and the affine transformations, we approximate the covariance for all the sequence pair matrices from the covariance of the pivot pair matrices. Here, the pivot pair matrix and the sequence pair matrix play the role of the matrices $\mathbf{X}$ and $\mathbf{Y}$ from Eq. (6). The same procedure can be used for computing other statistical measures. Next, we shall discuss the techniques for selecting the pivot pairs.

A. Computing the Dot Product

As a special case, we can show that the approximation error for computing the dot product can be completely eliminated. Let us assume that the affine transformation $(\mathbf{A}, \boldsymbol{b})$ is computed using the least-squares method, and it transforms

$\mathbf{X}$ to $\mathbf{Y}'$, instead of $\mathbf{Y}$, where $\mathbf{Y}' = [\boldsymbol{y}_1', \boldsymbol{y}_2']$. Then, the approximation error for computing the dot product $\boldsymbol{y}_2^\top \boldsymbol{y}_1$ can be eliminated using the following lemma:

Lemma 1: The dot product $\boldsymbol{y}_2^\top \boldsymbol{y}_1$ is preserved when the affine transformation $(\mathbf{A}, \boldsymbol{b})$ is computed using the least-squares method and it transforms $\boldsymbol{y}_1$ with zero error.

Proof: Let the hyperplane spanned by vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ be denoted as $\mathcal{H}_x$. Since $\boldsymbol{y}_2'$ is the least-squares approximation of $\boldsymbol{y}_2$, $\boldsymbol{y}_2 = \boldsymbol{y}_2' + \epsilon_p$, where ϵ_p is perpendicular to $\mathcal{H}_x$. Then $\boldsymbol{y}_2^\top \boldsymbol{y}_1 = \boldsymbol{y}_2'^\top \boldsymbol{y}_1 + \epsilon_p^\top \boldsymbol{y}_1$. Since $\boldsymbol{y}_1$ is part of the hyperplane $\mathcal{H}_x$, $\epsilon_p^\top \boldsymbol{y}_1 = 0$. Hence, $\boldsymbol{y}_2^\top \boldsymbol{y}_1 = \boldsymbol{y}_2'^\top \boldsymbol{y}_1$ ■

Obviously, Lemma 1 holds even if we replace $\boldsymbol{y}_1$ by $\boldsymbol{y}_2$ and $\boldsymbol{y}_2'$ by $\boldsymbol{y}_1'$. A straightforward way of guaranteeing the transformation of $\boldsymbol{y}_1$ with zero error is to have $\boldsymbol{y}_1$ common between the source pair and target pair matrices. In addition, as we shall show in Section IV, having a common time series dramatically reduces the number of pivot pairs generated using the SYMEX algorithm.

B. Computing Other Measures

For other dispersion and derived measures, the exact computation using affine transformations is, in general, not possible. Therefore, we propose a distance measure for measuring the error in affine transformations, and then a clustering algorithm that helps us identify high-quality affine relationships.

The LSFD Metric.
The *Least Significant Frobenius Distance* (LSFD) metric, when minimized using the clustering algorithm, results in high-quality (i.e., low error) affine transformations between the members of a given cluster. A small LSFD between the matrices $\mathbf{X}$ and $\mathbf{Y}$, indicates that $\mathbf{X}$ is almost perfectly transformable into $\mathbf{Y}$. The LSFD metric is defined as follows:

Definition 1: **LSFD metric.** Suppose $\hat{\mathbf{X}}$ and $\hat{\mathbf{Y}}$ are the zero-mean counterparts of the matrices $\mathbf{X}$ and $\mathbf{Y}$ respectively. Then the LSFD metric is defined as:

$$\mathfrak{D}_F(\mathbf{X}, \mathbf{Y})^2 \triangleq \lambda_3^2 + \lambda_4^2, \tag{9}$$

where λ_3 and λ_4 are the third and fourth singular values of the matrix $[\hat{\mathbf{X}}, \hat{\mathbf{Y}}]$, which is a matrix obtained by column-wise concatenation of $\hat{\mathbf{X}}$ and $\hat{\mathbf{Y}}$.

Definition 1 assumes that the vectors in $\hat{\mathbf{X}}$ are linearly independent. Therefore, if the third and the fourth singular values of the matrix $[\hat{\mathbf{X}}, \hat{\mathbf{Y}}]$ are zero, then it signifies that vectors $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ are linearly dependent and can be expressed as linear combinations of vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Thus, a zero-error affine transformation between $\mathbf{X}$ and $\mathbf{Y}$ can be computed. Intuitively, the magnitude of the third and the fourth singular value of $[\hat{\mathbf{X}}, \hat{\mathbf{Y}}]$ quantifies the effort required for making $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ linearly dependent on $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.

The LSFD is a metric; we formally prove this in the following theorem:

Theorem 1: $\mathfrak{D}_F(\mathbf{X}, \mathbf{Y})$ obeys the triangular inequality:

$$\mathfrak{D}_F(\mathbf{X}, \mathbf{Y}) \leq \mathfrak{D}_F(\mathbf{X}, \mathbf{Z}) + \mathfrak{D}_F(\mathbf{Z}, \mathbf{Y}). \tag{10}$$

Proof: Let us consider three matrices $\mathbf{I}_{\hat{X}\hat{Y}} = [\hat{\mathbf{X}}, \hat{\mathbf{Y}}]$, $\mathbf{I}_{\hat{X}\hat{Z}} = [\hat{\mathbf{X}}, \hat{\mathbf{Z}}]$, and $\mathbf{I}_{\hat{Z}\hat{Y}} = [\hat{\mathbf{Z}}, \hat{\mathbf{Y}}]$. Let $\tilde{\mathbf{I}}_{\hat{X}\hat{Y}}$ be the rank two approximation of the matrix $\mathbf{I}_{\hat{X}\hat{Y}}$. The Frobenius norm of $\|\mathbf{I}_{\hat{X}\hat{Y}} - \tilde{\mathbf{I}}_{\hat{X}\hat{Y}}\|_F$ is $\sqrt{\lambda_3^2 + \lambda_4^2}$. Similarly, let $\tilde{\mathbf{I}}_{\hat{X}\hat{Z}}$ and $\tilde{\mathbf{I}}_{\hat{Z}\hat{Y}}$ be the rank two approximations of the matrices $\mathbf{I}_{\hat{X}\hat{Z}}$ and $\mathbf{I}_{\hat{Z}\hat{Y}}$ respectively. Then,

$$\mathbf{I}_{\hat{X}\hat{Y}} = \mathbf{I}_{\hat{X}\hat{Z}} + \mathbf{I}_{\hat{Z}\hat{Y}} + [-\hat{\mathbf{Z}}, -\hat{\mathbf{Z}}]. \tag{11}$$

Let $\mathbf{B} = [-\hat{\mathbf{Z}}, -\hat{\mathbf{Z}}] + \tilde{\mathbf{I}}_{\hat{X}\hat{Z}} + \tilde{\mathbf{I}}_{\hat{Z}\hat{Y}}$. From Eq. (11):

$$\|\mathbf{I}_{\hat{X}\hat{Y}} - \mathbf{B}\|_F \leq \|\mathbf{I}_{\hat{X}\hat{Z}} - \tilde{\mathbf{I}}_{\hat{X}\hat{Z}}\|_F + \|\mathbf{I}_{\hat{Z}\hat{Y}} - \tilde{\mathbf{I}}_{\hat{Z}\hat{Y}}\|_F.$$

Using the Eckart-Young low-rank matrix approximation theorem [12] and the definition of LSFD in Definition 1:

$$\|\mathbf{I}_{\hat{X}\hat{Y}} - \tilde{\mathbf{I}}_{\hat{X}\hat{Y}}\|_F \leq \|\mathbf{I}_{\hat{X}\hat{Z}} - \tilde{\mathbf{I}}_{\hat{X}\hat{Z}}\|_F + \|\mathbf{I}_{\hat{Z}\hat{Y}} - \tilde{\mathbf{I}}_{\hat{Z}\hat{Y}}\|_F,$$
$$\mathfrak{D}_F(\mathbf{X}, \mathbf{Y}) \leq \mathfrak{D}_F(\mathbf{X}, \mathbf{Z}) + \mathfrak{D}_F(\mathbf{Z}, \mathbf{Y}).$$

Since LSFD obeys the triangular inequality, it is a metric. ■

Next, we shall see how the LSFD is used as a distance metric for clustering time-series data.

C. The Affine Clustering Algorithm

The affine clustering algorithm (AFCLST) clusters the time series in the data matrix $\mathbf{S}$ into k clusters, such that it becomes easier to construct a pivot pair matrix with low LSFD error to the given sequence pair matrix. Let us understand how we can construct such a pivot pair matrix. From Lemma 1, we should have one time series common among the sequence pair and pivot pair matrices. Suppose we have the first time series common, then the second time series in the pivot pair matrix is set to the cluster center assigned to the second time series in the sequence pair matrix. We show that by using the cluster center for constructing the pivot pair matrix, the LSFD between the pivot pair matrix and the sequence pair matrix is minimized, resulting in high-quality affine transformations.

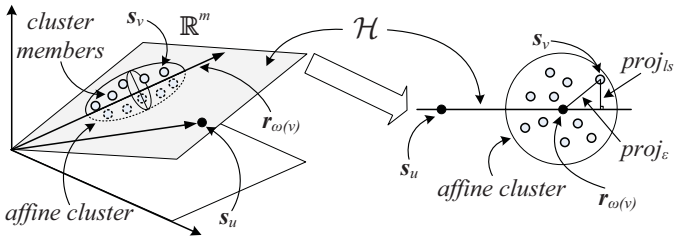


Fig. 4. (a) 2-D hyperplane $\mathcal{H}$ spanned by $\boldsymbol{s}_u$ and $\boldsymbol{r}_{\omega(v)}$, and (b) directional view of the hyperplane $\mathcal{H}$.

The AFCLST algorithm is shown in Algorithm 1. It starts by initializing the cluster centers $\boldsymbol{r}_\ell$, where $\ell = (1, \ldots, k)$ (Line 2). In the assignment phase, each time series $\boldsymbol{s}_v$ is assigned to the cluster having the least orthogonal projection error $proj_\epsilon$ between that cluster's center and $\boldsymbol{s}_v$ (Lines 9, 13, and Fig. 4(*b*)). The lesser the projection error $proj_\epsilon$, the more accurately a time series is approximated by a linear combination of its cluster center; leading to a lower LSFD between the sequence pair matrix and the pivot pair matrix.

In the update phase, the cluster centers $\boldsymbol{r}_\ell$ are re-computed by forming a matrix $\mathbf{R}_\ell$ of the time series assigned to cluster ℓ. The updated cluster center is equal to the left singular vector

Algorithm 1 The AFCLST affine clustering algorithm.

Input: Data matrix $\mathbf{S}$, maximum iterations γ_{max}, number of clusters k, minimum cluster changes δ_{min}.
Output: Cluster centers $\boldsymbol{r}_\ell$ and cluster assignment function $\omega(v)$.

1: **for** $\ell = 1$ to k **do** ▷ Initialization phase
2: $\boldsymbol{r}_\ell \leftarrow$ `randcol`($\mathbf{S}$), $\boldsymbol{r}_\ell \leftarrow \frac{\boldsymbol{r}_\ell}{\|\boldsymbol{r}_\ell\|}$ ▷ choose a random column
3: $nChg \leftarrow -1$
4: **for** $iter = 1$ to γ_{max} **do**
5: $minProj_\epsilon \leftarrow \infty$, $clustID \leftarrow 0$
6: **for** $j = 1$ to m **do** ▷ Assignment phase
7: **for** $\ell = 1$ to k **do**
8: $proj_{\boldsymbol{r}_\ell} \leftarrow (\boldsymbol{r}_\ell \boldsymbol{r}_\ell^\top)\boldsymbol{s}_j$, $proj_\epsilon \leftarrow ||proj_{\boldsymbol{r}_\ell} - \boldsymbol{s}_j||$
9: **if** $proj_\epsilon < minProj_\epsilon$ **then**
10: $clustID \leftarrow \ell$
11: **if** $\omega(j)$!= $clustID$ **then**
12: $currNChg \leftarrow currNChg + 1$
13: $\omega(j) \leftarrow clustID$
14: **if** $|nChg - currNChg| \leq \delta_{min}$ **then**
15: **break** ▷ Converged
16: **for** $\ell = 1$ to k **do** ▷ Update phase
17: $\mathbf{R}_\ell \leftarrow \emptyset$
18: **for** $j = 1$ to m **do**
19: **if** $\omega(j) == \ell$ **then**
20: $\mathbf{R}_\ell \leftarrow [\mathbf{R}_\ell, \boldsymbol{s}_j]$
21: $\boldsymbol{r}_\ell \leftarrow$ `SVDLV`($\mathbf{R}_\ell$) ▷ Largest left singular vector
22: **return** $\boldsymbol{r}_\ell, \omega(u)$

associated with the largest singular value of $\mathbf{R}_\ell$. Intuitively, this vector minimizes the sum of the errors $proj_\epsilon$ between the cluster center $\boldsymbol{r}_\ell$ and the members of the cluster ℓ.

The AFCLST algorithm terminates when the cluster membership changes are less than δ_{min} or γ_{max} iterations are completed. It returns two quantities: (a) cluster centers $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_k$ and (b) a cluster assignment function $\omega(v) : v \mapsto \ell$, which returns the cluster identifier ℓ for a given series identifier v.

Observe from Fig. 4 that the least-squares error $proj_{ls}$, which is obtained from projecting $\boldsymbol{s}_v$ onto the 2-D hyperplane $\mathcal{H}$ spanned by vectors $\boldsymbol{s}_u$ and $\boldsymbol{r}_{\omega(v)}$, is always less than $proj_\epsilon$. This is due to Pythagoras theorem. And, since we minimize $proj_\epsilon$ using the AFCLST algorithm, approximating $\boldsymbol{s}_v$ by projecting it on $\mathcal{H}$ is at least as better as the result from the AFCLST algorithm.

Next, we present formal, crisp definitions of a pivot pair and a pivot pair matrix:

Definition 2: **Pivot pair and pivot pair matrix.** The *pivot pair* associated to the sequence pair $e = (u, v)$ is defined as $p = (u, \omega(v))$. It is obtained by replacing the series identifier v in e by its cluster identifier $\omega(v)$. The *pivot pair matrix*, denoted as $\mathbf{O}_p$, is the matrix obtained by concatenating $\boldsymbol{s}_u$ with the cluster center $\boldsymbol{r}_{\omega(v)}$ as follows:

$$\mathbf{O}_p \triangleq [\boldsymbol{s}_u, \boldsymbol{r}_{\omega(v)}]. \tag{12}$$

Admittedly, $(\omega(u), v)$ is also considered a pivot pair, but for reasons of brevity we only use Definition 2 of a pivot pair. We end this section by defining the most crucial concept proposed in this paper – *affine relationship*:

Definition 3: **Affine relationship.** An affine relationship associates the sequence pair e to its pivot pair p. It is defined as an affine transformation between the sequence pair matrix $\mathbf{S}_e$ and the pivot pair matrix $\mathbf{O}_p$:

$$\mathbf{S}_e \triangleq \mathbf{O}_p \mathbf{A}_e + \mathbb{1}_m \boldsymbol{b}_e^\top, \tag{13}$$

where $\mathbf{A}_e \in \mathbb{R}^{2\times 2}$ is non-singular, $\boldsymbol{b}_e \in \mathbb{R}^2$, and $\mathbb{1}_m = (1, \ldots, 1)^\top \in \mathbb{R}^m$. We denote an affine relationship as $(\mathbf{A}, \boldsymbol{b})_e$.

IV. Computing Affine Relationships

In this section we propose an algorithm for generating the pivot pairs p for the given sequence pairs e. Secondly, we propose a method for efficiently computing the affine relationships between the selected pivot and sequence pairs.

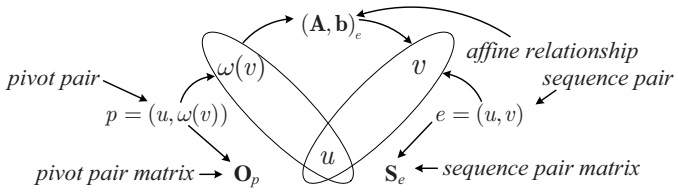


Fig. 5. Procedure for generating the pivot pairs.

The proposed algorithm is as follows (refer Fig. 5):

1. Select a sequence pair $e = (u, v)$ that is not associated with a pivot pair.
2. Create the two pivot pairs for e: $(u, \omega(v))$ and $(\omega(u), v)$.
3. Associate the pivot pair $(u, \omega(v))$ to the sequence pair e, and form a new sequence pair by changing the second component of e to another member of the cluster $\omega(v)$.
4. Repeat Step 3 with the newly formed sequence pair, until all the members of the cluster $\omega(v)$ have been associated the pivot pair $(u, \omega(v))$.
5. Use the other pivot pair $(\omega(u), v)$ and repeat Step 3, now changing the first component of e.
6. Jump to Step 1 if there are more sequence pairs that have not been associated with a pivot pair.

A formal algorithm of the Steps 1-6 above is presented in Algorithm 2. In Algorithm 2, as opposed to the Step 1 above, we select the unassociated sequence pairs more systematically. The algorithm starts processing the sequence pair set $\mathcal{P}$ using two fixed sequence pairs: e_e and e_w. Then, it systematically generates new sequence pairs (Line 6 and Line 9). Loops on Line 13 and 16 scan each component of the new sequence pair, until the boundary of the set $\mathcal{P}$ is reached.

During each step, a sequence pair e is associated to a pivot pair p, only if it was not associated earlier (Line 20). On Line 21, affine relationship $(\mathbf{A}, \boldsymbol{b})_e$ is computed, and is stored in the hash map affHash. affHash is returned by the algorithm along with another hash map pivotHash, which stores the generated pivot pairs. Since Algorithm 2 systematically selects the sequence pairs, we refer to it as the SYMEX (Systematic Exploration) algorithm.

The SYMEX algorithm produces maximum nk pivot pairs, where k is the number of affine clusters. But, in practice, we found that $k << n$, thus the SYMEX algorithm produces pivot pairs nearly linear in n. Moreover, the complexity of the SYMEX algorithm is $\mathcal{O}(g)$, where g is the number of affine relationships. In Section VI, we experimentally demonstrate the linear scalability of the SYMEX algorithm.

Algorithm 2 The SYMEX algorithm.

Input: Data matrix $\mathbf{S}$, AFCLST parameters k, γ_{max}, and δ_{min}.
Output: Hash maps affHash and pivotHash, containing the affine relationships and the pivot pairs respectively.

1: $(\boldsymbol{r}_\ell, \omega(u)) \leftarrow$ `AFCLST`$(\mathbf{S}, k, \gamma_{max}, \delta_{min})$
2: $e_e \leftarrow (0, n)$, $e_w \leftarrow (\frac{n-1}{2}, \frac{n-1}{2} + 1)$ ▷ sequence pairs
3: $flip \leftarrow 0$
4: **while** e_e != e_w **do**
5: **if** $flip == 0$ **then**
6: $e_e \leftarrow e_e + (1, -1)$, $flip \leftarrow 1$ ▷ move towards e_w
7: `CreatePivots`(e_e, affHash)
8: **else if** $flip == 1$ **then**
9: $e_w \leftarrow e_w + (-1, 1)$, $flip \leftarrow 0$ ▷ move towards e_e
10: `CreatePivots`(e_w, affHash)
11: **return** affHash
12: **function** `CreatePivots`($e_z = (u_z, v_z)$, affHash)
13: **for** $v = u_z + 1$ to n **do** ▷ Scan second component
14: $e \leftarrow (u_z, v)$, $p \leftarrow (u_z, \omega(v))$
15: `SolveInsert`($\mathbf{O}_p$, $\mathbf{S}_e$, affHash)
16: **for** $u = 0$ to v_z **do** ▷ Scan first component
17: $e \leftarrow (u, v_z)$, $p \leftarrow (\omega(u), v_z)$
18: `SolveInsert`($\mathbf{O}_p$, $\mathbf{S}_e$, affHash)
19: **function** `SolveInsert`($\mathbf{O}_p$, $\mathbf{S}_e$, affHash)
20: **if** affHash.*lookup*(e) == $\emptyset$ **then**
21: $(\mathbf{A}, \boldsymbol{b}) \leftarrow$ `LeastSquares`($\mathbf{O}_p$, $\mathbf{S}_e$)
22: affHash.*insert*(e, $(\mathbf{A}, \boldsymbol{b})$) ▷ $insert(key, value)$
23: **if** pivotHash.*lookup*(p) == $\emptyset$ **then**
24: pivotHash.*insert*(p, $\emptyset$) ▷ null hash values
25: **function** `LeastSquares`($\mathbf{O}_p$, $\mathbf{S}_e$)
26: $pinv \leftarrow$ `PseudoInv`($[\mathbf{O}_p, \mathbb{1}_m]$) ▷ Pseudo-inverse
27: $(\mathbf{A}, \boldsymbol{b}) \leftarrow pinv \cdot \mathbf{S}_e$
28: **return** $(\mathbf{A}, \boldsymbol{b})$

Pseudo-inverse Cache.
Notice that, on Line 26, the SYMEX algorithm computes the pseudo-inverse of the matrix $[\mathbf{O}_p, \mathbb{1}_m]$. Since there are many sequence pairs associated to a single pivot pair, the same pseudo-inverse is repeatedly re-computed for each sequence pair. Thus, we can cache the pseudo-inverse instead of re-computing it. We call the algorithm that caches the pseudo-inverse the SYMEX+ algorithm. The pseudo-inverse cache is populated by storing the pseudo-inverse with key p, before the calls to the `SolveInsert` function (Line 15 and Line 18). Then, the pseudo-inverse is only computed if the cache lookup is unsuccessful. We show in Section VI that the SYMEX+ algorithm is a factor of *4 times* faster than the SYMEX algorithm.

A. Measure Computation Query

We discuss processing of the MEC query (Query 1) using affine relationships. Assume that the MEC query has requested to compute the covariance matrix of the series identifiers ψ. Let us denote the sequence pairs formed by the series identifiers ψ as $e_\psi \in \mathcal{P}$.

The query processing starts by inserting the values of the covariance matrix $\Sigma(\mathbf{O}_p)$, for each pivot pair p contained in the hash map pivotHash (returned by the SYMEX+ algorithm). Then, for each $e_\psi \in \mathcal{P}$, $\Sigma_{e_\psi}(\mathbf{S})$ is computed using the following equation:

$$\Sigma_{e_\psi}(\mathbf{S}) = \Sigma_{12}(\mathbf{S}_{e_\psi}) = \boldsymbol{a}_1^\top \Sigma(\mathbf{O}_{p_\psi}) \boldsymbol{a}_2, \tag{14}$$

where $\boldsymbol{a}_1 = (a_{11}, a_{21})^\top$ and $\boldsymbol{a}_2 = (a_{12}, a_{22})^\top$ are the first and second columns of the matrix $\mathbf{A}_{e_\psi}$, and $\Sigma(\mathbf{O}_{p_\psi})$ and $\mathbf{A}_{e_\psi}$ are obtained by searching pivotHash and affHash respectively. This procedure is followed for all the sequence pairs e_ψ, and the resulting $|\psi|$-by-$|\psi|$ covariance matrix is returned.

Similarly, $\mathcal{L}$-measures, dot product, or $\mathcal{D}$-measures can be processed using their corresponding properties in Eqs. (5), (7) and (8) along with the output of the SYMEX+ algorithm. For the $\mathcal{D}$-measures, the separable normalizers are computed in the pre-processing step.

Cost Analysis: The total computational cost of the MEC query can be divided into three parts: (a) a one-time cost of order $\mathcal{O}(nk)$ for computing and storing the covariance matrices of all the nk pivot pairs, (b) the average run-time cost of finding an affine relationship from affHash is of order $\mathcal{O}(1)$, and (c) a small cost for computing the requested measure using Eq. (6). As it can be seen, the one-time cost $\mathcal{O}(nk)$ of (a) dominates the Big-$\mathcal{O}$ complexity. In contrast, the naïve approach is of the order $\mathcal{O}(n^2)$. Since $k << n$, in practice this dominating one-time cost becomes nearly linear in the number of time series n, leading to significantly large performance improvements.

Accuracy Measurement: Suppose $\hat{\Sigma}_e(\mathbf{S})$ and $\Sigma_e(\mathbf{S})$ respectively are the true value (computed from scratch) and the approximated value (computed using affine relationships) of the covariance for the sequence pair e. We, then, measure the accuracy by computing the percentage RMSE (root-mean-square error) as follows:

$$\%\ \text{RMSE} = \sqrt{\frac{\sum_{e \in \mathcal{P}} \left(\hat{\Sigma}_e^n(\mathbf{S}) - \Sigma_e^n(\mathbf{S}) \right)^2}{|\mathcal{P}|}} \times 100, \tag{15}$$

where $\hat{\Sigma}_e^n(\mathbf{S})$ and $\Sigma_e^n(\mathbf{S})$ are normalized by dividing $\hat{\Sigma}_e(\mathbf{S})$ and $\Sigma_e(\mathbf{S})$ with $(\max(\hat{\Sigma}_e(\mathbf{S})) - \min(\hat{\Sigma}_e(\mathbf{S})))$.

V. Indexing Affine Relationships

In this section we propose efficient methods for processing the MET and MER queries described in Section II-B. A straightforward way of processing these queries is to either use the naïve approach or the affine relationships approach to first compute the value of the queried statistical measure and then trivially evaluate the MET and MER queries.

A major drawback of the naïve approach is that we have to re-compute the queried statistical measure for every query and for all sequence pairs, which makes this approach inefficient, especially when large number of queries are processed. In contrast, the Scalar Projection or SCAPE index is designed in such a way that: (a) queries over statistical measures can be processed without re-computing the measure for every query, and (b) one index structure can be used for indexing all the $\mathcal{L}$-, $\mathcal{T}$-, and $\mathcal{D}$-measures. Furthermore, the SCAPE index improves the efficiency of processing the MET and MER queries by orders of magnitude.

The SCAPE index is constructed for a $\mathcal{T}$-measure. It consists of one sorted container, like a B-tree, for each pivot pair. Each sorted container, associated to a pivot pair, stores the affine relationships assigned to that pivot pair. The key used for sorting is the most crucial and novel component of the SCAPE index. The key chosen for the SCAPE index should be such that it can be used for querying the $\mathcal{T}$-measure and *all* the $\mathcal{D}$-measures derived from it. For example, the same key can be used for querying the covariance ($\mathcal{T}$-measure) and the correlation coefficient ($\mathcal{D}$-measure). Additionally, the key should be such that a query (MET or MER) over a $\mathcal{T}$-measure – or its $\mathcal{D}$-measure – could be converted into a query (MET or MER) over the keys stored by the SCAPE index, guaranteeing the same results from the converted and the original query.

For choosing a key with the above properties, we propose using an interesting property of the scalar product between two vectors. Let us briefly understand this property through an example. Suppose we have a vector $\boldsymbol{\alpha}$ and vectors $\boldsymbol{\beta}_l$, where l is a positive integer, and our objective is to order the scalar product $\boldsymbol{\alpha}^\top \boldsymbol{\beta}_l \in \mathbb{R}$. Then, it is interesting to note the following observation.

Observation 1: The scalar projections $\xi_l = \|\boldsymbol{\beta}_l\| \cos(\theta_l)$ can be used as a key for ordering the scalar products $\boldsymbol{\alpha}^\top \boldsymbol{\beta}_l = \|\boldsymbol{\alpha}\| \cdot \|\boldsymbol{\beta}_l\| \cos(\theta_l)$, since $\|\boldsymbol{\alpha}\|$ is a common factor and does not affect the ordering of the scalar products (refer Fig. 6).

Now let us discuss the application of Observation 1 for indexing affine relationships.

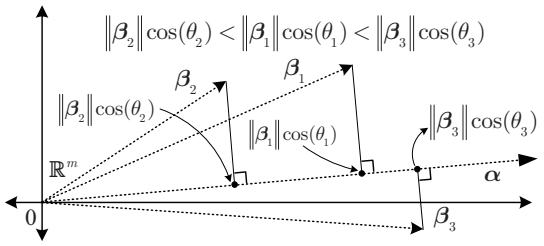


Fig. 6. Toy example demonstrating Observation 1.

A. Scalar Projection (SCAPE) Index

Assume that we obtained $\mathcal{Q}$ pivot pairs by executing the SYMEX algorithm described in Section IV. Let us denote them as p_q where $q = (1, 2, \ldots, \mathcal{Q})$. Also, assume that each pivot pair p_q has $\mathcal{D}_q$ sequence pairs associated with it. Let us denote these sequence pairs as e_{qd} where $d = (1, 2, \ldots, \mathcal{D}_q)$. Suppose we are interested in processing the MET and MER queries for the covariance. Recall that given the affine relationship $(\mathbf{A}, \boldsymbol{b})_{e_{qd}}$ for a sequence pair e_{qd} and the covariance matrix of the pivot matrix $\Sigma(\mathbf{O}_{p_q})$, the covariance $\Sigma_{e_{qd}}(\mathbf{S})$ can be estimated as follows (refer Eq. (14)):

$$\Sigma_{e_{qd}}(\mathbf{S}) = \boldsymbol{a}_2^\top \Sigma(\mathbf{O}_{p_q}) \boldsymbol{a}_1, \tag{16}$$

where $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ are first and second columns of the transformation matrix $\mathbf{A}_{e_{qd}}$. Since from Definition 2, we have a common time series, namely $\boldsymbol{s}_u$, between the sequence pair e_{qd} and the pivot pair p_q, it simplifies the structure of $\boldsymbol{a_1}$ to $(1, 0)^\top$. Thus, Eq. (16) becomes:

$$\Sigma_{e_{qd}}(\mathbf{S}) = (\Sigma_{11}(\mathbf{O}_{p_q}), \Sigma_{21}(\mathbf{O}_{p_q})) \begin{pmatrix} a_{12} \\ a_{22} \end{pmatrix}. \tag{17}$$

We then define $\boldsymbol{\alpha}_q = (\Sigma_{11}(\mathbf{O}_{p_q}), \Sigma_{21}(\mathbf{O}_{p_q}))^\top$, $\boldsymbol{\beta}_{qd} = (a_{12}, a_{22})^\top$, and thus $\Sigma_{e_{qd}}(\mathbf{S}) = \boldsymbol{\alpha}_q^\top \boldsymbol{\beta}_{qd}$. Now, similar to Observation 1, the scalar products $\boldsymbol{\alpha}_q^\top \boldsymbol{\beta}_{qd}$ can be ordered by ordering the scalar projections $\xi_{qd} = \|\boldsymbol{\beta}_{qd}\| \cos(\theta_{qd})$, where θ_{qd} is the angle between $\boldsymbol{\alpha}_q$ and $\boldsymbol{\beta}_{qd}$. Notice that $\boldsymbol{\beta}_{qd}$ is derived only using the affine relationships, and does not change even if $\boldsymbol{\alpha}_q$ changes. Thus, we have essentially decoupled the affine relationship (captured by $\boldsymbol{\beta}_{qd}$) from the statistical measure (captured by $\boldsymbol{\alpha}_q$). This decoupling allows us to define a $\boldsymbol{\alpha}_q$ for other measures without affecting the structure of the SCAPE index. Table II summarizes the values of $\boldsymbol{\alpha}_q$ and $\boldsymbol{\beta}_{qd}$ for all the $\mathcal{L}$- and $\mathcal{T}$-measures.

TABLE II

CHOICES OF $\boldsymbol{\alpha}_q$ AND $\boldsymbol{\beta}_{qd}$ FOR THE SCAPE INDEX.

	$\boldsymbol{\alpha}_q$	$\boldsymbol{\beta}_{qd}$
Location $\mathcal{L}_2(\mathbf{S}_{e_{qd}})$	$(\mathcal{L}_1(\mathbf{O}_{p_q}), \mathcal{L}_2(\mathbf{O}_{p_q}), 1)^\top$	$(a_{12}, a_{22}, b_2)^\top$
Covariance $\Sigma_{12}(\mathbf{S}_{e_{qd}})$	$(\Sigma_{12}(\mathbf{O}_{p_q}), \Sigma_{22}(\mathbf{O}_{p_q}), 0)^\top$	$(a_{12}, a_{22}, b_2)^\top$
Dot product $\Pi_{12}(\mathbf{S}_{e_{qd}})$	$(\Pi_{12}(\mathbf{O}_{p_q}), \Pi_{11}(\mathbf{O}_{p_q}), h_1(\mathbf{O}_{p_q}))$	$(a_{12}, a_{22}, b_2)^\top$

*The third column refers to entries in $(\mathbf{A}, \boldsymbol{b})_e$.

Index Structure: The structure of the SCAPE index for a $\mathcal{T}$-measure is shown in Fig. 7. It contains two types of nodes: (a) *sequence node* that includes the scalar projection $\xi_{qd} = \|\boldsymbol{\beta}_{qd}\| \cos(\theta_{qd})$ and the sequence pair e_{qd}, and (b) *pivot node* that includes the pivot pair p_q, $\|\boldsymbol{\alpha}_q\|$ for the $\mathcal{T}$-measure that is indexed by the SCAPE index, and a pointer to a sorted container, that stores the sequence nodes associated with the pivot pair p_q. The key for sorting the sorted containers is the scalar projection ξ_{qd}.

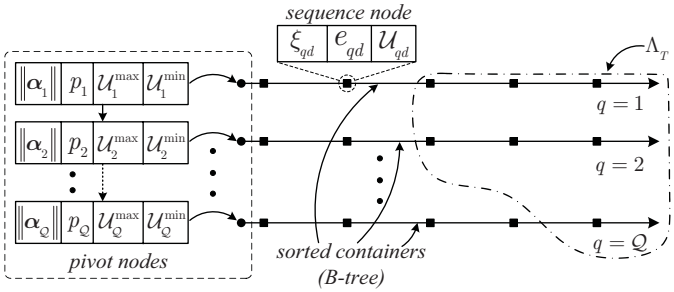


Fig. 7. Example of the SCAPE index for indexing $\mathcal{T}$- and $\mathcal{D}$- measures.

Indexing $\mathcal{D}$-Measures: To add a derived measure ($\mathcal{D}$-measure) to an existing SCAPE index, we additionally store the normalizer $\mathcal{U}_{qd}$ of the indexed $\mathcal{D}$-measure in each sequence node (refer Fig. 7). For example, if the SCAPE index has indexed covariance then $\sqrt{\Sigma(\boldsymbol{s}_u)\Sigma(\boldsymbol{s}_v)}$ is stored for the correlation coefficient. Also, in each pivot node we store the maximum and minimum values of the normalizer ($\mathcal{U}_q^{\max}$ and $\mathcal{U}_q^{\min}$) found in the B-tree associated with the pivot pair p_q.

In Section V-C, we show that the above two quantities,

$\mathcal{U}_q^{\max}$ and $\mathcal{U}_q^{\min}$, are sufficient to prune the SCAPE index and efficiently process the MET and MER queries on the $\mathcal{D}$-measures. Similarly, other $\mathcal{D}$-measures, which are not included in this paper, can also be indexed by the SCAPE index, once their $\mathcal{T}$-measures are indexed.

B. Processing Threshold and Range Queries

Consider the MET query that is requesting the sequence pairs such that the covariance is greater than a user-defined threshold τ. We start the processing by considering one pivot node at a time. Since τ is a threshold for $\|\boldsymbol{\alpha}_q\| \cdot \xi_{qd}$, we have to divide τ by $\|\boldsymbol{\alpha}_q\|$ to be able to compare it with ξ_{qd}. Thus, for a given pivot node, we compute the modified threshold $\tau' = \frac{\tau}{\|\boldsymbol{\alpha}_q\|}$. Next, we scan the B-tree associated with that pivot node, using a binary search algorithm and collect e_{qd}, such that $\tau' > \xi_{qd}$. We repeat this process for all the pivot nodes. The collected set of e_{qd} is the result set Λ_T of the MET query. Fig. 7 shows an example of Λ_T.

Procedure for processing the MER query is similar to the MET query, the only difference is that in the MER query all the e_{qd} such that $\tau'_l < \xi_{qd} < \tau'_u$ are collected, where $\tau'_l = \frac{\tau_l}{\|\boldsymbol{\alpha}_q\|}$ and $\tau'_u = \frac{\tau_u}{\|\boldsymbol{\alpha}_q\|}$.

C. Index-based Pruning for $\mathcal{D}$-Measures

Processing the MET and MER queries over the $\mathcal{D}$-measures is a challenging problem. The primary challenge is that normalization destroys the ordering of the scalar projections ξ_{qd}, which is needed for processing queries over the $\mathcal{T}$-measure. Therefore, the idea here is to prune the sequence pairs using the values $\mathcal{U}_q^{\max}$ and $\mathcal{U}_q^{\min}$, stored in each pivot node. Our pruning technique quickly eliminates a large number of sequence pairs that do not satisfy the query condition(s).

Suppose we have a SCAPE index and a MET query that is requesting all sequence pairs such that the correlation coefficient (a $\mathcal{D}$-measure) is greater than τ. We start by considering one pivot node at a time. For a given pivot node, we compute two modified thresholds: $\tau'_{\min} = \frac{\tau \cdot \mathcal{U}_q^{\min}}{\|\boldsymbol{\alpha}_q\|}$ and $\tau'_{\max} = \frac{\tau \cdot \mathcal{U}_q^{\max}}{\|\boldsymbol{\alpha}_q\|}$, where $\boldsymbol{\alpha}_q$ corresponding to covariance in Table II is used. Observe that the sequence nodes, where $\xi_{pd} > \tau'_{\max}$, are definitely in the result set Λ_T, and are directly added to Λ_T without further processing. This situation is depicted in Fig. 8(*a*) and holds because of the following:

$$\xi_{pd} > \tau'_{\max} \Leftrightarrow \frac{\|\boldsymbol{\alpha}_q\| \cdot \xi_{qd}}{\mathcal{U}_q^{\max}} > \tau \Leftrightarrow \rho_{e_{qd}}(\mathbf{S}) > \tau. \qquad (18)$$

Likewise, the correlation coefficient for all the sequence nodes where $\xi_{pd} < \tau'_{\min}$ can only be less than τ, and can be excluded from the result set Λ_T. The sequence nodes where $\tau'_{\min} < \xi_{qd} < \tau'_{\max}$ cannot be pruned. Thus, for these sequence nodes, we compute the correlation coefficient and check whether it is greater than τ and update the result set Λ_T. We repeat this process for all the pivot nodes, and finally return the result set Λ_T.

Similarly, consider a MER query that is requesting all the sequence pairs such that their correlation coefficient is between τ_l and τ_u. We compute four modified thresholds: $\tau'_{l\min} = \frac{\tau_l \cdot \mathcal{U}_q^{\min}}{\|\boldsymbol{\alpha}_q\|}$, $\tau'_{l\max} = \frac{\tau_l \cdot \mathcal{U}_q^{\max}}{\|\boldsymbol{\alpha}_q\|}$, $\tau'_{u\min} = \frac{\tau_u \cdot \mathcal{U}_q^{\min}}{\|\boldsymbol{\alpha}_q\|}$, and $\tau'_{u\max} = \frac{\tau_u \cdot \mathcal{U}_q^{\max}}{\|\boldsymbol{\alpha}_q\|}$. Again, following a similar reasoning as the MET query, the sequence nodes where $\xi_{pd} > \tau'_{u\max}$ and $\xi_{pd} < \tau'_{l\min}$ cannot be in the result set Λ_R.

For the sequence nodes where $\tau'_{l\max} < \xi_{qd} < \tau'_{u\min}$ there could be two cases: (1) *case I*: $\tau'_{l\max} < \tau'_{u\min}$, and (2) *case II*: $\tau'_{l\max} > \tau'_{u\min}$. These cases are depicted in Fig. 8(*b*). For *case I*, the sequence nodes where $\tau'_{l\max} < \xi_{qd} < \tau'_{u\min}$ can be directly included in the result set Λ_R without further processing. In *case II*, pruning like *case I* is not possible. In both the cases, for the unpruned sequence nodes we compute the correlation coefficient and check whether it is in between τ_l and τ_u and update the result set Λ_R.

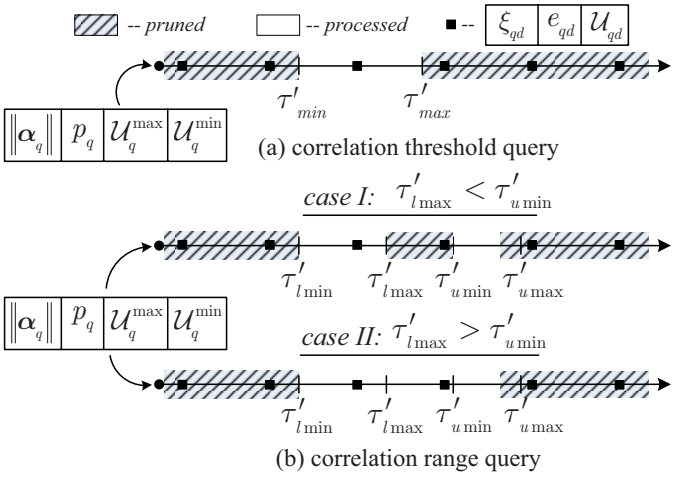


Fig. 8. Index-based pruning for processing MET and MER queries.

Note that the same index-based pruning techniques can be utilized for other $\mathcal{D}$-measures. In Section VI, we compare the query processing methods using the SCAPE index, the naïve method, and other state-of-art methods to demonstrate the dramatic improvement in performance as compared to the other methods.

VI. Experimental Evaluation

In this section we perform extensive experimental evaluation on real datasets to establish the efficacy of our approaches. In Section VI-A, we analyze the trade-off between accuracy and efficiency for computing statistical measures. Performance improvements in query processing using synthetic – but realistic – workloads are discussed in Section VI-B. Scalability of the SYMEX algorithm is established in Section VI-C, followed by experiments using the SCAPE index in Section VI-D.

Since we have more than one method for computing and querying the statistical measures, as a shorthand we use the following notations:

- $\mathcal{W}_N$: a naïve method that computes a given statistical measure from scratch,
- $\mathcal{W}_A$: an approach that uses the affine relationships for computing a statistical measure (refer Section IV-A),
- $\mathcal{W}_F$: an approach that uses the five largest DFT (Discrete Fourier Transform) coefficients for approximating the correlation coefficient; introduced in [1].

In this paper we use two real datasets. The first dataset contains 670 daily time series obtained from 134 sensors monitoring ambient temperature, relative humidity, and surface temperature on the EPFL campus. We refer to this dataset as *sensor-data*. The second dataset consists of weekly, intra-day stock quotes from 996 stocks from the S&P 500 index and ETFs (exchange traded funds). We refer to this dataset as *stock-data*. The most important characteristics of the datasets are summarized in Table III.

TABLE III
SUMMARY OF THE DATASETS.

	sensor-data	*stock-data*
sampling interval	2 min.	1 min.
#time series (n)	670	996
#samples per time series (m)	720	1,950
max. affine relationships	224,115	495,510

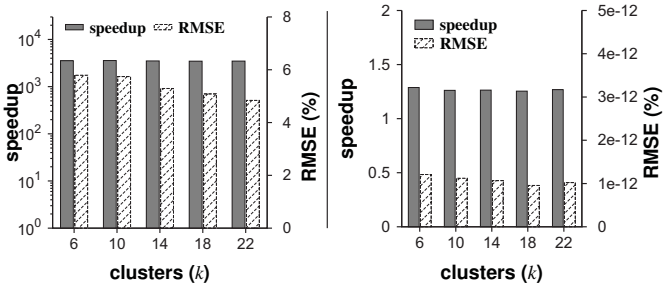
A. Analyzing Trade-Off

We analyze the trade-off between efficiency and accuracy by considering a measure computation query that computes a statistical measure ($\mathcal{L}$, $\mathcal{T}$, or $\mathcal{D}$) over *all* the time series in a dataset. Fig. 9 and Fig. 10 show the speedup and the percentage RMSE (defined in Eq. (15)) obtained for various statistical measures as a function of the number of affine clusters k. The speedup is computed as the ratio of time taken by the $\mathcal{W}_N$ method as compared to the $\mathcal{W}_A$ method.

Since other measures exhibit similar trends to the measures included in Fig. 9 and Fig. 10, we do not show them due to limited space, but are included in [13]. Also, the comparison of absolute time can be found in [13]. In particular, for computing statistical measures, the main focus of our work, the errors are negligible. This shows that various real datasets contain affine relationships, and the AFCLST algorithm is highly effective in finding those complex affine dependencies.

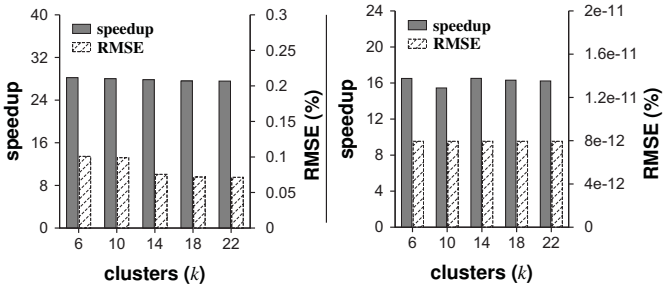
The speedup obtained over all the statistical measures varies largely from a *factor 1.3 to 3500*. The maximum speedup of approximately *3500 times* is obtained for mode. The speedup obtained for dot product (*1.3x*) is low due to the inherent simplicity of computing it using the $\mathcal{W}_N$ method.

Observe that the percentage RMSE does not decrease rapidly as the number of clusters are increased. This is the case since the AFCLST algorithm minimizes the LSFD, and it may not always lead to a dramatic decrease in the percentage RMSE. Nonetheless, the accuracy measured by the percentage RMSE is significantly high. Moreover, since a small numbers of clusters are sufficient to obtain high accuracy, we obtain a nearly linear cost of processing the MEC query. Thus, overall the $\mathcal{W}_A$ method exhibits significant improvements in efficiency and accuracy.



(a) mode (b) dot product

Fig. 9. Trade-off between efficiency and accuracy for *sensor-data*. Note the logarithmic scale for the speedup in (a).

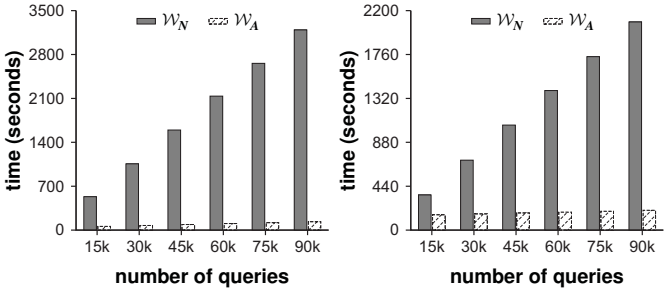


(a) median (b) covariance

Fig. 10. Trade-off between efficiency and accuracy for *stock-data*.

B. Impact of Online Environments

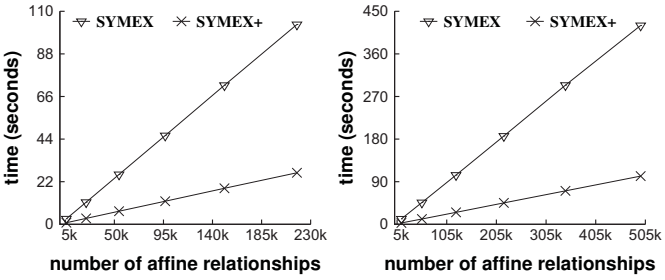
Typically, in online environments, users frequently request for computation of a particular statistical measure over a few entities (stocks or sensors). To simulate this behavior, we generate realistic query workloads as follows: each MEC query chooses uniformly at random a $\mathcal{L}$-, $\mathcal{T}$-, or $\mathcal{D}$-measure and uses a powerlaw distribution for choosing 10 different series identifiers to form the set ψ. The powerlaw distribution is used since it is a well-known model for popularity.



(a) *sensor-data* (b) *stock-data*

Fig. 11. Comparing query processing efficiency.

Fig. 11 compares query processing performance as the number of queries increase. Here the parameters of the SYMEX+ algorithm are chosen as: $k = 6$, $\gamma_{max} = 10$, and $\delta_{min} = 10$. The gains obtained by using the $\mathcal{W}_A$ method are many-fold as compared to the $\mathcal{W}_N$ method. For example, the $\mathcal{W}_A$ method is *10 to 23 times faster* as compared to the $\mathcal{W}_N$ method when 90k queries are processed. Note that the time for the $\mathcal{W}_A$ method shown in Fig. 11 also includes the time taken by the SYMEX+ algorithm for computing the affine relationships. Thus, the $\mathcal{W}_A$ method is far superior than the $\mathcal{W}_N$ method, and is suitable for deployment in online environments. Here we cannot compare with the $\mathcal{W}_F$ method, since unlike $\mathcal{W}_A$, the $\mathcal{W}_F$ method is unable to compute all the statistical measures.



(a) *sensor-data* (b) *stock-data*

Fig. 12. Scalability of the SYMEX algorithm.

C. Scalability of the SYMEX Algorithm

Fig. 12 shows the scaling behavior of the SYMEX and the SYMEX+ algorithms as the number of affine relationships handled by these algorithms increase. For experiments in Fig. 12, we set $k = 6$, $\gamma_{max} = 10$, and $\delta_{min} = 10$ as the parameters of the AFCLST algorithm. The SYMEX and the SYMEX+ algorithms scale linearly. Particularly, the SYMEX+ algorithm is a factor *3.5 to 4 times* faster as compared to the simple SYMEX algorithm. Thus, the pseudo-inverse cache, used in the SYMEX+ algorithm, results in attractive performance improvements.

D. Impact of using the SCAPE Index

Next, we discuss the performance improvements obtained by using the SCAPE index. We build the SCAPE index by using the affine relationships that are returned by the SYMEX+ algorithm for the covariance and the mean of *sensor-data*. For processing the MET and MER queries on the correlation coefficient the index-based pruning methods discussed in Section V-C are utilized. We first analyze the

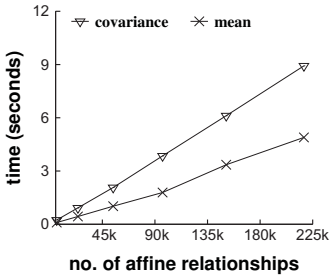
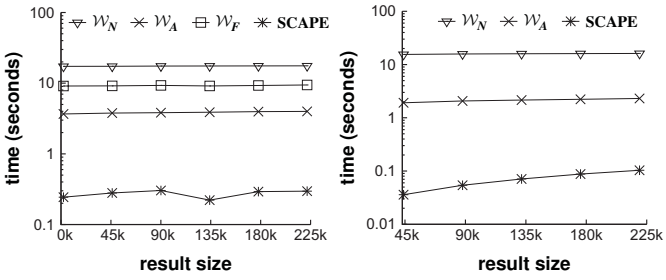


Fig. 13. Scalability of the SCAPE index construction on *sensor-data*.

scalability of constructing the SCAPE index as the number of indexed affine relationships increase. Fig. 13 shows the scaling behavior of the SCAPE index construction when it indexes the affine relationships for a $\mathcal{T}$-measure (covariance) and a $\mathcal{L}$-measure (mean). Admittedly, the SCAPE index exhibits linear scaling, which makes it a viable, practical solution for query processing.

Next, in Fig. 14 we compare the results for processing the MET and MER queries using the SCAPE index. Here, the other methods ($\mathcal{W}_N$, $\mathcal{W}_A$, and $\mathcal{W}_F$) first compute the required statistical measure and then trivially evaluate the MET or MER query. Note that since $\mathcal{W}_F$ only computes the correlation coefficient, it is excluded from Fig. 14(*b*).



(a) correlation coefficient (b) covariance

Fig. 14. Comparing efficiency of the SCAPE index. (a) MET query and (b) MER query.

TABLE IV
QUERY PROCESSING PERFORMANCE FOR THE MAXIMUM RESULT SIZE.

Query type	Measure	Speedup		
		$\mathcal{W}_N$	$\mathcal{W}_A$	$\mathcal{W}_F$
MET	correlation coefficient	59x	13.4x	32x
	covariance	160x	21x	×
	dot product	41x	35x	×
	median	5x	1.1x	×
MER	correlation coefficient	27x	6.4x	14x
	covariance	155x	22x	×

Fig. 14 depicts the orders of magnitude improvement (shown using logarithmic scale) in efficiency for processing the MET and MER queries using the SCAPE index. Table IV shows a snapshot of the tremendous performance improvement for all the statistical measures, and also in particular when comparing to the best known methods from the literature ($\mathcal{W}_F$) for the computation of the correlation coefficient. More experiments that demonstrate the enhancements in query processing performance for other measures can be found in [13].

VII. Extensions and Future Work

There are many issues that remain to be researched. We briefly discuss some future directions in which we are planning to extend this research.

Pruning Affine Relationships: It is not mandatory that we process and store all the affine relationships. We can, if required, prune the unnecessary affine relationships on the basis of domain knowledge, query requirements, low correlation between a sequence pair, etc. Such pruning techniques will be considered in subsequent works. On the contrary, here we consider all the affine relationships returned by the SYMEX algorithm, for clearly demonstrating performance and scalability results.

Dynamic Affine Relationships: Affine relationships can change dynamically, especially as new data is streamed into the system. Handling dynamic affine relationships requires: (i) a sequentially updating version of the AFCLST algorithm [14], and (ii) updating the changed affine relationships in the SCAPE index. Task (ii) is similar to a standard index update operation in a DBMS. Supporting dynamic affine relationships is an interesting direction that we plan to explore in our subsequent works.

Distributed Query Processing: Many datasets are large and

cannot be stored on a single computing device. Therefore, researching techniques for distributing the SCAPE index, and for performing affine clustering in a distributed setting becomes important. Thus, extending the proposed techniques to a distributed environment is an open problem.

VIII. Related Work

Many prior works transform data from time domain to frequency domain using the DFT and then use the equivalence of norms (Parseval's theorem) property of the DFT for approximating the correlation coefficient [1]–[3]. DFT-based techniques provide inaccurate results when the time series contain white noise. Cole *et al.* [4] call such time series *uncooperative* and propose methods for discovering correlation among such signals. All these studies, however, only consider the correlation coefficient and are not applicable for computing and querying a wide variety of statistical measures.

In addition to computing the correlation coefficient using the DFT, there has been a large body of related prior research where the DFT is used for: (a) exact or approximate sequence matching where the sequences could have undergone a similarity transformation [15]–[17], (b) retrieving similar shapes [18], [19], (c) predicting future values and answering similarity queries [20], and (d) reducing the dimensionality of time-series data [21], [22]. Our work, on the contrary, considers affine transformations, which are a more generalized form of similarity transformations. Secondly, these techniques do not notice that affine transformations can be used for efficiently computing statistical measures.

TAPER [5] defines an all-strong-pairs correlation query that returns pairs of highly positively correlated items given a user-specified threshold. SPRIT [23], on the other hand, uses PCA (Principal Component Analysis) for summarizing large collections of streams and discovering correlations. Our work differs from those mainly due to the fact that those techniques are tightly coupled to a particular type of query or statistical measure, most often the correlation coefficient. In that sense our work is unique.

Processing aggregate or related queries over time-series data is another area related to our work [24]. The Cypress framework [6] uses Fourier transform for segmenting the data into various form of trickles, which are then used for query processing. Similarly, GAMPS [25] uses ratio signals for compressing time-series data and processing queries over it. More recently, there has been research conducted on indexing and querying correlated uncertain information using probabilistic databases [26], [27]. Lastly, Ke *et al.* [28] propose approaches for searching graphs correlated to a given query graph.

IX. Conclusion

In this paper, for the first time, we defined and proposed the notion of affine relationships for computing and querying several statistical measures using an unified approach. We proposed the affine clustering algorithm for finding high-quality affine relationships in time-series data. We proposed the SYMEX algorithm that is capable of computing affine relationships in linear time. We showed that the SCAPE index structure can easily index all the statistical measures, and can achieve orders of magnitude performance enhancement in query processing, as compared to the naïve methods and methods proposed in the literature for this problem. Lastly, we performed comprehensive experiments highlighting the effectiveness of our approaches.

References

[1] Y. Zhu and D. Shasha, "Statstream: Statistical monitoring of thousands of data streams in real time," in *VLDB*, 2002, pp. 358–369.

[2] C.-S. Li, P. S. Yu, and V. Castelli, "Hierarchyscan: A hierarchical similarity search algorithm for databases of long sequences," in *ICDE*, 1996, pp. 546–553.

[3] A. Mueen, S. Nath, and J. Liu, "Fast approximate correlation for massive time-series data," in *SIGMOD*, 2010, pp. 171–182.

[4] R. Cole, D. Shasha, and X. Zhao, "Fast window correlations over uncooperative time series," in *SIGKDD*, 2005, pp. 743–749.

[5] H. Xiong, S. Shekhar, P. Tan, and V. Kumar, "TAPER: A two-step approach for all-strong-pairs correlation query in large databases," *TKDE*, pp. 493–508, 2006.

[6] G. Reeves, J. Liu, S. Nath, and F. Zhao, "Managing massive time series streams with multi-scale compressed trickles," in *VLDB*, 2009, pp. 97–108.

[7] J. Hull, *Options, futures and other derivatives*. Prentice Hall, 2009.

[8] M. J. Bommarito II, "Intraday Correlation Patterns between the S&P 500 and Sector Indices," *SSRN*, 2010.

[9] J. Campbell, S. Grossman, and J. Wang, "Trading volume and serial correlation in stock returns," *The Quarterly Journal of Economics*, vol. 108, no. 4, p. 905, 1993.

[10] W. Sharpe, "Capital asset prices: A theory of market equilibrium under conditions of risk," *Journal of Finance*, vol. 19, no. 3, pp. 425–442, 1964.

[11] R. Maronna, R. Martin, and V. Yohai, *Robust statistics*. Wiley Series in Probability and Statistics, 2006.

[12] G. Golub and C. Van Loan, *Matrix computations*. The Johns Hopkins University Press, 1996.

[13] S. Sathe and K. Aberer, "AFFINITY: Efficiently querying statistical measures on time-series data," EPFL, Tech. Rep., 2012, http://infoscience.epfl.ch/record/180121.

[14] C. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[15] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient similarity search in sequence databases," in *FODO*, 1993, pp. 69–84.

[16] R. Agrawal, K. Lin, H. Sawhney, and K. Shim, "Fast similarity search in the presence of noise, scaling and translation in time-series databases," in *VLDB*, 1995.

[17] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast subsequence matching in time-series databases," in *SIGMOD*, 1994, pp. 419–429.

[18] D. Rafiei and A. Mendelzon, "Similarity-based queries for time series data," in *SIGMOD*, 1997, pp. 13–25.

[19] H. Jagadish, A. Mendelzon, and T. Milo, "Similarity-based queries," in *PODS*, 1995, pp. 36–45.

[20] X. Lian and L. Chen, "Efficient similarity search over future stream time series," *TKDE*, vol. 20, no. 1, pp. 40–54, 2008.

[21] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Locally adaptive dimensionality reduction for indexing large time series databases," in *SIGMOD*, 2001, pp. 151–162.

[22] ——, "Dimensionality reduction for fast similarity search in large time series databases," *KAIS*, vol. 3, no. 3, pp. 263–286, 2001.

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

[24] J. Gehrke, F. Korn, and D. Srivastava, "On computing correlated aggregates over continual data streams," in *SIGMOD*, 2001, pp. 13–24.

[25] S. Gandhi, S. Nath, S. Suri, and J. Liu, "GAMPS: Compressing multi sensor data by grouping and amplitude scaling," in *SIGMOD*, 2009, pp. 771–784.

[26] C. Ré, J. Letchner, M. Balazinksa, and D. Suciu, "Event queries on correlated probabilistic streams," in *SIGMOD*, 2008, pp. 715–728.

[27] B. Kanagal and A. Deshpande, "Indexing correlated probabilistic databases," in *SIGMOD*, 2009, pp. 455–468.

[28] Y. Ke, J. Cheng, and W. Ng, "Correlation search in graph databases," in *SIGKDD*, 2007, pp. 390–399.