AFFINITY: Efficiently Querying Statistical Measures on Time-Series Data

Saket Sathe and Karl Aberer

Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
{saket.sathe, 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 directly computing them; thus, reducing the overall computation cost significantly. The resulting methods show 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 on time-series data.

Our approach includes three key components, which exploit affine relationships. First, the AFCLST algorithm that clusters the time-series data such that high-quality affine relationships could be easily found. Second, the SYMEX algorithm that uses the clustered time series and efficiently computes the desired affine relationships. Third, the SCAPE index structure that 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 using real datasets.

1 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, the correlation coefficient is a frequently used statistical measure for financial data. It is well-known that stock traders and investors are interested to find the correlation coefficient among pairs of stocks. Specifically, traders are interested in solving the following problem [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 naive approach for solving Problem 1 is to compute the correlation coefficients for all the pairs of stocks for the day specified by the problem. Clearly, for high values of n this method does not scale well, since it computes the correlation coefficient for all the $\frac{n(n-1)}{2}$ pairs from scratch.

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

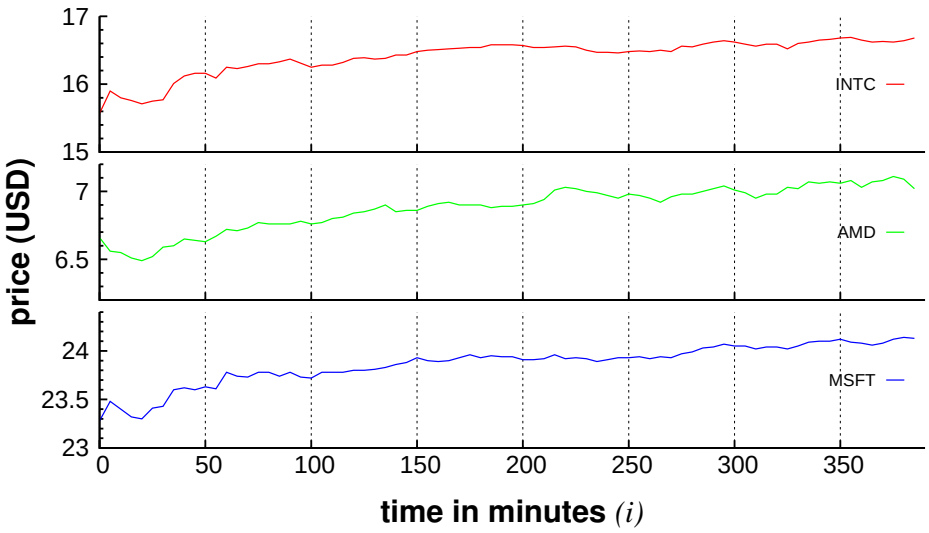


Fig. 1. Stock prices for symbols INTC, AMD and MSFT on 2^{nd} January 2003.

The first idea that this paper proposes in order to enhance the naive 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 $(2,3)$ can be defined by using an affine transformation:

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

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

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 6, we experimentally demonstrate that these advantages manifest a many-fold increase in performance.

Similarly, many other measures of correlation and similarity, beyond the commonly used Pearson's correlation coefficient, can be computed using affine relationships. Thus, by utilizing affine relationships, our approach provides an elegant solution for computing a wide range of statistical measures. As a consequence, our proposal to use affine relationships not only bears the potential

of increasing the efficiency of computing the correlation coefficient, but, at the same time, 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, such relationships may 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 can take as input, values from stocks $\boldsymbol{s}_1$, $\boldsymbol{s}_2$, and $\boldsymbol{s}_3$ in a specific time window and could quantitatively judge the quality of affine relationships. Additionally, we also propose the AFCLST clustering algorithm that uses the LSFD metric for clustering the time series, such that good-quality affine relationships could be found between cluster members.

We have found that although, in practice, it is almost impossible to find an exact affine relationship between time series, interestingly, a large number of high-quality approximate affine relationships exist in real datasets over longer time intervals. Thus, queries could leverage from such relationships for computing many statistical measures on-the-fly, while bounding the approximation error in the computation of these statistical measures.

Indexing affine relationships.
Consider a slightly modified version of Problem 1, where a trader is interested to find all pairs of stock that have the 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 increasing value of n.

A way of circumventing the computation of all the pairwise correlation coefficients is to index the affine relationships. We call this index the SCAPE index. Prior to indexing, the SCAPE index establishes a way of ordering 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 is agnostic to the underlying statistical measure. As a result, the SCAPE index can be used for simultaneously indexing all the statistical measures.

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 as for 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) that is capable of clustering the given data such that high-quality (low LSFD) affine relationships could be found within the cluster members.
- We introduce an efficient algorithm (i.e., SYMEX), that generates high-quality affine relationships on-the-fly, by utilizing the output of the AFCLST clustering algorithm.
- We show that indexing affine relationships with the SCAPE index, 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 2. In Section 3, we propose the LSFD metric and the AFCLST clustering algorithm for finding high-quality affine relationships in time series data. In Section 4, we introduce the SYMEX algorithm for generating high-quality affine relationships, while, in Section 5, we propose the SCAPE index for indexing affine relationships. Lastly, comprehensive experimental evaluations are presented in Section 6, followed by the review of related studies in Section 7.

2 Foundation

In this section we define the basic concepts and establish the notation used in the rest of the paper. A summary of the frequently used notations is presented in Table 1. We, then, define the queries that are processed by the AFFINITY framework. Most importantly, we discuss the notion of affine transformations and examine their properties. Affine relationships are, in fact, enhanced affine transformations designed for facilitating efficient computation and querying of several statistical measures.

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. AFFINITY consists of two key components: the affine relationships and the SCAPE index structure. The affine relationships are inferred using the data_matrix table, and are indexed for processing statistical queries 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 can 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*.

2.1 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.

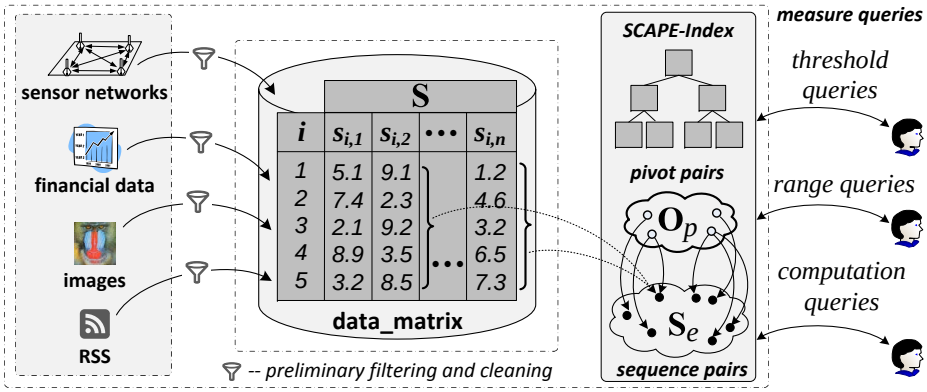


Fig. 2. Architecture of the AFFINITY framework.

Often the statistical measures considered in this paper are required to be computed on pairs of time series. A good example are the covariance and the correlation coefficient. 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 series identifiers $(1, 2, \ldots, n)$ 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 the *series identifier*. Similarly, let $\mathcal{P} = \{(u, v) | u < v \text{ and } (u, v) \in \mathcal{I} \times \mathcal{I}\}$ 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 the *sequence pair*.

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

matrices of size $n \times n$. In the matrices $\mathcal{T}(\mathbf{S})$ and $\mathcal{D}(\mathbf{S})$, the entry found at row u and column v is respectively the dispersion and the derived measure between the time series u and v of the matrix $\mathbf{S}$. The entry found at position (u, v) of $\mathcal{T}(\mathbf{S})$ and $\mathcal{D}(\mathbf{S})$ is denoted as $\mathcal{T}_{uv}(\mathbf{S})$ and $\mathcal{D}_{uv}(\mathbf{S})$ respectively. The $\mathcal{T}$- and $\mathcal{D}$-measures are symmetric, that is $\mathcal{T}_{uv}(\mathbf{S}) = \mathcal{T}_{vu}(\mathbf{S})$ and $\mathcal{D}_{uv}(\mathbf{S}) = \mathcal{D}_{vu}(\mathbf{S})$. Secondly, the entry at the position $e = (u, v)$ of the matrix $\mathcal{T}(\mathbf{S})$ denoted as $\mathcal{T}_e(\mathbf{S})$ is equal to $\mathcal{T}_{12}(\mathbf{S}_e)$, which is the entry at position (1,2) of the matrix $\mathcal{T}(\mathbf{S}_e)$. In short, $\mathcal{T}_e(\mathbf{S}) = \mathcal{T}_{12}(\mathbf{S}_e)$ and $\mathcal{D}_e(\mathbf{S}) = \mathcal{D}_{12}(\mathbf{S}_e)$.

In this paper, we consider three $\mathcal{L}$-measures: mean, mode, and median. In addition, 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 $\Pi(\mathbf{S})$. We also consider one $\mathcal{D}$-measure, namely, the 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 u and v and $\mathcal{L}_u(\mathbf{S})$ denotes a location measure of the time series u.

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

Table 1. 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

2.2 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 statistical measure ($\mathcal{L}$, $\mathcal{T}$, or $\mathcal{D}$) the* measure computation query *returns the value of the given statistical measure for the time series ψ.*

For the $\mathcal{T}$- and $\mathcal{D}$-measures, the response is a matrix of size $|\psi|$-by-$|\psi|$, and for $\mathcal{L}$-measures the response is a vector of size $|\psi|$. For example, the measure computation query could request the mean or the covariance matrix for a subset of the series identifiers ψ.

The second query returns all the series identifiers (sequence pairs) where the location measure (dispersion or derived measure) is greater or lesser than a user-defined threshold.

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) for which the given statistical measure $\mathcal{L}_u(\mathbf{S})$ ($\mathcal{T}_e(\mathbf{S})$ 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 consisting of the series identifiers u (sequence pairs e) for which the given statistical measure $\mathcal{L}_u(\mathbf{S})$ ($\mathcal{T}_e(\mathbf{S})$ 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 for which the covariance is in between τ_l and τ_u.

2.3 Affine Transformations

Consider any two 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, thus $\mathbf{X}$ and $\mathbf{Y}$ are of size m-by-2. 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.

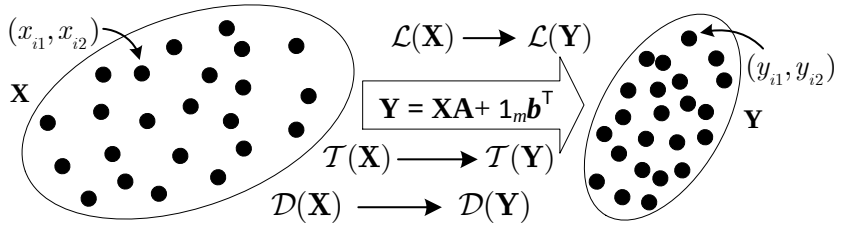


Fig. 3. Illustration of an affine transformation.

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

Additionally, the $\mathcal{D}$-measures are derived by normalizing one of the $\mathcal{T}$-measures. 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 separately computed. 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 1, it is rarely possible to find a perfect affine transformation. To rectify this problem, in the next section, we propose techniques to improve the quality of affine transformations.

3 Affine Clustering

Consider the problem of computing a statistical measure, say covariance, for all the sequence pairs. The naive approach of solving this problem is to compute the covariance for all the sequence pairs from scratch. But, computing covariances from scratch is inefficient because it requires scanning of the sequence pair matrices $\mathbf{S}_e$, for all the sequence pairs e. Since the number of sequence pairs are $\mathcal{O}(n^2)$, where n is the number of time series n, it leads to an overall inefficient operation.

We reduce the $\mathcal{O}(n^2)$ complexity by selecting a small (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 pairs and determine the affine transformations between each sequence pair and one of the pivot pairs. Next, with the help of Eq. (6) and the affine transformations, we approximate the covariance for all the sequence pairs from the covariance of the pivot pairs. Similarly, other measures can also be only computed for the pivot pairs; and then approximated for the sequence pairs. Note that the affine transformations need to be computed only once.

Next, we describe the selection procedure for the pivot pairs. It should satisfy two requirements: (1) the number of selected pivot pairs should be small, and (2) the affine transformations, when used for approximating a statistical measure, should produce low error. In this section we propose techniques for meeting both these requirements.

3.1 Computing the Dot Product

For the dot product, as a special case, we can show that the approximation error can be completely eliminated by having a common time series between the source and target pair matrices. 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, for accurately computing the dot product $\boldsymbol{y}_2^\top \boldsymbol{y}_1$ using affine transformations, we observe that it is sufficient to have one common time series between $\mathbf{X}$ and $\mathbf{Y}$, because of the following lemma:

Lemma 1. *The dot product $\boldsymbol{y}_2^\top \boldsymbol{y}_1$ is preserved under the action of an affine transformation $(\mathbf{A}, \boldsymbol{b})$ that is computed using the least-squares method, if $\boldsymbol{y}_1$ is transformed with zero error.*

Proof. Let the hyperplane spanned by vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ be denoted as $\mathcal{H}$. 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 $\mathcal{H}$. 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 $\mathcal{H}$, $\epsilon_p^\top \boldsymbol{y}_1 = 0$. 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 to both the source pair and target pair matrices. In this case, we

can guarantee that the dot product $\boldsymbol{y}_2^\top \boldsymbol{y}_1$ is accurately computed if the affine transformations are computed using the least-squares method. In addition, as we shall show in Section 4, having a common time series reduces the number of pivot pairs, which are generated using the SYMEX algorithm.

3.2 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 minimizing this error.

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 source pair matrix $\mathbf{X}$ and the target pair matrix $\mathbf{Y}$ 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 Least Significant Frobenius Distance (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 $\hat{\mathbf{Y}}$.

The number of non-zero singular values of a matrix is equal to the number of linearly independent vectors in that matrix. 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, an exact affine transformation between $\mathbf{X}$ and $\mathbf{Y}$ can be computed. Intuitively, the magnitude of the third and the fourth singular value of the matrix $[\hat{\mathbf{X}}, \hat{\mathbf{Y}}]$ quantifies the effort required for making $\boldsymbol{y}_1$ or $\boldsymbol{y}_2$ linearly dependent on $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.

The LSFD obeys the triangular inequality, and therefore is also a metric. Since LSFD is a metric, it can be used as a distance metric for affine clustering. A formal proof of the triangular inequality is presented in the following theorem:

Theorem 1. *$\mathfrak{D}_F(\mathbf{X}, \mathbf{Y})$ is a metric; thus $\mathfrak{D}_F$ 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,

$$\begin{aligned} \|\mathbf{I}_{\hat{X}\hat{Y}} - \tilde{\mathbf{I}}_{\hat{X}\hat{Y}}\|_F \leq \|\mathbf{I}_{\hat{X}\hat{Z}} - \tilde{\mathbf{I}}_{\hat{X}\hat{Z}}\|_F + \|\mathbf{I}_{\hat{Z}\hat{Y}} - \tilde{\mathbf{I}}_{\hat{Z}\hat{Y}}\|_F, \\ \mathfrak{D}_F(\mathbf{X}, \mathbf{Y}) \leq \mathfrak{D}_F(\mathbf{X}, \mathbf{Z}) + \mathfrak{D}_F(\mathbf{Z}, \mathbf{Y}). \end{aligned} \tag{12}$$

3.3 The AFCLST Clustering Algorithm

The affine clustering algorithm clusters the time series in the data matrix $\mathbf{S}$ into k clusters, such that it becomes easier to identify a high-quality affine transformation between a sequence pair and a pivot pair. We have one common time series between the sequence pair matrix and the pivot matrix for computing the dot product accurately. As a result the common time series is transformed with zero LSFD error. Next, for the other (different) time series in the sequence pair matrix, the affine clustering algorithm finds the closest match such that the LSFD between the sequence pair matrix and the pivot pair matrix becomes as low as possible.

The closest match for the different time series is its cluster center, which is returned by the affine clustering algorithm. We show that by following this procedure 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. Thus, the pivot pair matrix – like the source pair matrix $\mathbf{X}$ – can be utilized for accurately computing the statistical measures over the sequence pair matrix.

The affine clustering algorithm clusters the time series in $\mathbf{S}$ into k clusters (refer Algorithm 1). It starts by initializing the cluster centers $\boldsymbol{r}_\ell$, where $\ell = (1, \ldots, k)$ (Lines 2 and 3). In the assignment step, the AFCLST algorithm computes the orthogonal projection of each time series $\boldsymbol{s}_v$, where $1 \leq v \leq n$, onto the cluster centers $\boldsymbol{r}_\ell$, and assigns $\boldsymbol{s}_v$ to the cluster that produces the least projection error $proj_\epsilon$ (Lines 10, 15, and Fig. 4(b)). The lesser the orthogonal projection error $proj_\epsilon$, the more accurately a time series can be represented by a linear combination of its cluster center; leading to a lower LSFD between the sequence pair matrix and the pivot matrix.

In the update phase, the cluster centers $\boldsymbol{r}_\ell$ are re-computed. This is done by forming a matrix $\mathbf{R}_\ell$ for each cluster ℓ, by column-wise concatenation of the time series assigned to cluster ℓ. The updated cluster center is equal to the left singular vector associated with the largest singular value of $\mathbf{R}_\ell$. Intuitively,

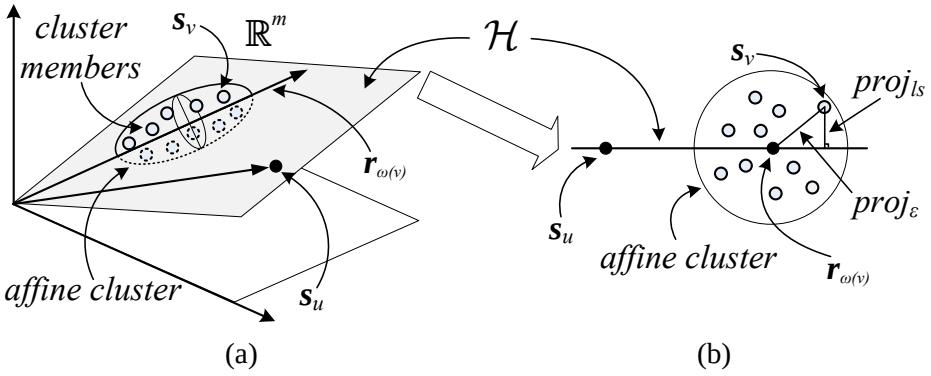


Fig. 4. (a) the 2-D hyperplane $\mathcal{H}$, and (b) directional view of the hyperplane $\mathcal{H}$.

the left singular vector associated with the largest singular value is the one that minimizes the sum of the orthogonal projection errors that are computed between the cluster center of cluster ℓ (i.e., $\boldsymbol{r}_\ell$) and each of its members.

The AFCLST algorithm terminates when the number of cluster membership changes is less than δ_{min} or the maximum number of iterations γ_{max} are completed. The AFCLST algorithm returns two quantities: (a) cluster centers $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_k$ and (b) a cluster assignment function $\omega(v) : v \mapsto \ell$, which returns the cluster identifier ℓ for a given series identifier v.

For a given sequence pair $e = (u, v)$, we now form a pivot pair matrix $[\boldsymbol{s}_u, \boldsymbol{r}_{\omega(v)}]$, obtained by concatenating the time series $\boldsymbol{s}_u$ and the cluster center of the time series $\boldsymbol{s}_v$. Furthermore, let $\mathcal{H}$ be the 2-D hyperplane spanned by the vectors $\boldsymbol{s}_u$ and $\boldsymbol{r}_{\omega(v)}$ (refer Fig. 4). Then there exists a high-quality affine transformation between the pivot pair matrix $[\boldsymbol{s}_u, \boldsymbol{r}_{\omega(v)}]$ and the sequence pair matrix $\mathbf{S}_e = [\boldsymbol{s}_u, \boldsymbol{s}_v]$. This is true since the projection error $proj_{ls}$, obtained from orthogonally projecting $\boldsymbol{s}_v$ onto $\mathcal{H}$, can only be less than $proj_\epsilon$. From Fig. 4(b), $proj_{ls}$ is one side of the right-angled triangle where $proj_\epsilon$ is an hypotenuse. Thus, approximating $\boldsymbol{s}_v$ using $[\boldsymbol{s}_u, \boldsymbol{r}_{\omega(v)}]$ further reduces the LSFD.

Now, we present formal, crisp definitions of the pivot pair and the pivot pair matrix, associated to a sequence pair $e = (u, v) \in \mathcal{P}$:

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

$$\mathbf{O}_p \triangleq [\boldsymbol{s}_u, \boldsymbol{r}_{\omega(v)}]. \tag{13}$$

Observe that $(\omega(u), v)$ 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 relationships*:

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 $\omega(v)$.

for $\ell = 1$ to k **do** ▷ Initialization phase
  $\boldsymbol{r}_\ell \leftarrow$ `randcol`$(\mathbf{S})$ ▷ choose a random column
  $\boldsymbol{r}_\ell \leftarrow \boldsymbol{r}_\ell / \|\boldsymbol{r}_\ell\|$ ▷ normalize
$nChg \leftarrow -1$
for $iter = 1$ to γ_{max} **do**
  $minProj_\epsilon \leftarrow \infty$, $clustID \leftarrow 0$
  for $j = 1$ to m **do** ▷ Assignment phase
    for $\ell = 1$ to k **do**
      $proj_{\boldsymbol{r}_\ell} \leftarrow (\boldsymbol{r}_\ell \boldsymbol{r}_\ell^\top)\boldsymbol{s}_j$
      $proj_\epsilon \leftarrow ||proj_{\boldsymbol{r}_\ell} - \boldsymbol{s}_j||$
      if $proj_\epsilon < minProj_\epsilon$ **then**
        $clustID \leftarrow \ell$
    if $\omega(j)$!= $clustID$ **then**
      $currNChg \leftarrow currNChg + 1$
    $\omega(j) \leftarrow clustID$
  if $|nChg - currNChg| \leq \delta_{min}$ **then**
    break ▷ Converged
  for $\ell = 1$ to k **do** ▷ Update phase
    $\mathbf{R}_\ell \leftarrow \emptyset$
    for $j = 1$ to m **do**
      if $\omega(j) == \ell$ **then**
        $\mathbf{R}_\ell \leftarrow [\mathbf{R}_\ell, \boldsymbol{s}_j]$
    $\boldsymbol{r}_\ell \leftarrow$ `SVDLV`$(\mathbf{R}_\ell)$ ▷ Largest left singular vector
return $\boldsymbol{r}_\ell, \omega(u)$

Definition 3. ***Affine relationship.*** *An affine relationship characterizes a relationship between the sequence pair e and its pivot pair p. Precisely, it is defined as an affine transformation between the sequence pair matrix $\mathbf{S}_e$ and the pivot pair matrix $\mathbf{O}_p$,*

$$\mathbf{S}_e \triangleq \mathbf{O}_p \mathbf{A}_e + \mathbb{1}_m \boldsymbol{b}_e^\top, \tag{14}$$

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

To summarize, we use the following procedure for selecting a pivot pair p for a given sequence pair e. First, we keep a common time series, namely $\boldsymbol{s}_u$, between the sequence pair matrix and the pivot pair matrix. Second, the other (uncommon) time series in the pivot pair matrix is the affine cluster center of the time series $\boldsymbol{s}_v$. By this procedure, the pivot pair of the sequence pair $e = (u, v)$ is $p = (u, \omega(v))$. In the following section we propose an algorithm that systematically follows this procedure for generating pivot pairs p that correspond to sequence pairs e

4 Computing Affine Relationships

In this section we propose an algorithm for generating the pivot pairs p for the given sequence pairs e using the procedure described in Section 3. Secondly, we propose a method for efficiently computing the affine relationships between the selected pivot and sequence pairs.

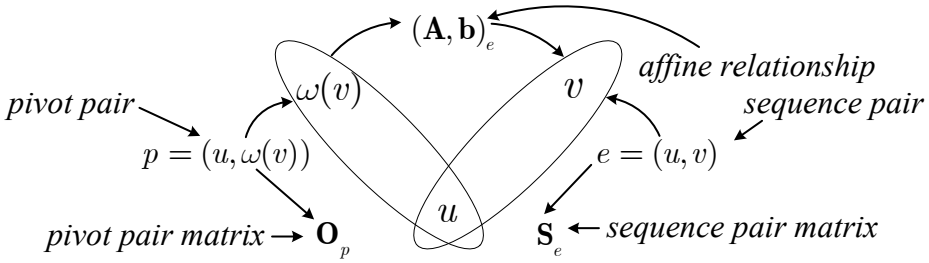


Fig. 5. Procedure for generating the pivot pairs. (*remove breve*)

The proposed algorithm follows the following steps (refer Fig. 5): (i) select any sequence pair $e = (u, v) \in \mathcal{P}$, (ii) generate both the possible pivot pairs for e: $(u, \omega(v))$ and $(\omega(u), v)$, (iii) associate the pivot pair $(u, \omega(v))$ to the sequence pair e, and then form a new sequence pair by changing the second component of e to another member of cluster $\omega(v)$, (iv) repeat Step (iii) with the new sequence pair, until all the members of the cluster $\omega(v)$ have been associated the pivot pair $(u, \omega(v))$, (v) use the other pivot pair $(\omega(u), v)$ and repeat Step (iii), now changing the first component, and (vi) jump to Step (i) if there are more sequence pairs that have not been associated a pivot pair.

A formal algorithm of the Steps (i)-(vi) is presented in Algorithm 2. The only difference is that, instead of selecting any sequence pair, as per Step (i), Algorithm 2 selects them systematically. The algorithm starts processing the sequence pair set $\mathcal{P}$ using two fixed sequence pairs: $e_e = (1, n)$ and $e_w = (\frac{n-1}{2}, \frac{n-1}{2} + 1)$. Then, it generates new sequence pairs by alternatively adding $(1, -1)$ and $(-1, 1)$ to e_e and e_w respectively (Line 6 and Line 9). On Line 14, it scans each component of the new sequence pair, until the boundary of the set $\mathcal{P}$ is reached.

During each step of the scan it associates a sequence pair e to a pivot pair p, only if the sequence pair e has not been associated with a pivot pair earlier (Line 20). On Line 21, e and p are used for computing the affine relationship $(\mathbf{A}, \boldsymbol{b})_e$. These affine relationships are stored in the hash map affHash. affHash is returned by the algorithm along with another hash map pivotHash, which stores the pivot pairs generated by the algorithm. The algorithm stops when both the sequence pairs e_e and e_w are equal. Since Algorithm 2 systematically selects the sequence pairs, we refer to it as the SYMEX (Systematic Exploration of $\mathcal{P}$) algorithm.

The SYMEX algorithm produces maximum nk number of 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 the number of time

Algorithm 2 The SYMEX algorithm.

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

$(\boldsymbol{r}_\ell, \omega(u)) \leftarrow$ `AFCLST`$(\mathbf{S}, k, \gamma_{max}, \delta_{min})$
$e_e \leftarrow (0, n)$, $e_w \leftarrow (\frac{n-1}{2}, \frac{n-1}{2} + 1)$ ▷ sequence pairs
$flip \leftarrow 0$
while e_e != e_w **do**
    if $flip == 0$ **then**
        $e_e \leftarrow e_e + (1, -1)$, $flip \leftarrow 1$ ▷ move towards e_w
        `CreatePivots`$(e_e$, affHash$)$
    else if $flip == 1$ **then**
        $e_w \leftarrow e_w + (-1, 1)$, $flip \leftarrow 0$ ▷ move towards e_e
        `CreatePivots`$(e_w$, affHash$)$
return affHash
function `CreatePivots`$(e_z = (u_z, v_z)$, affHash$)$
    for $v = u_z + 1$ to n **do** ▷ Scan second component
        $e \leftarrow (u_z, v)$, $p \leftarrow (u_z, \omega(v))$
        `SolveInsert`$(\mathbf{O}_p, \mathbf{S}_e$, affHash$)$
    for $u = 0$ to v_z **do** ▷ Scan first component
        $e \leftarrow (u, v_z)$, $p \leftarrow (\omega(u), v_z)$
        `SolveInsert`$(\mathbf{O}_p, \mathbf{S}_e$, affHash$)$
function `SolveInsert`$(\mathbf{O}_p, \mathbf{S}_e$, affHash$)$
    if affHash.*lookup*$(e) == \emptyset$ **then**
        $(\mathbf{A}, \boldsymbol{b}) \leftarrow$ `LeastSquares`$(\mathbf{O}_p, \mathbf{S}_e)$
        affHash.*insert*$(e, (\mathbf{A}, \boldsymbol{b}))$ ▷ $insert(key, value)$
    if pivotHash.*lookup*$(p) == \emptyset$ **then**
        pivotHash.*insert*$(p, \emptyset)$ ▷ null hash values
function `LeastSquares`$(\mathbf{O}_p, \mathbf{S}_e)$
    $pinv \leftarrow$ `PseudoInv`$([\mathbf{O}_p, \mathbb{1}_m])$ ▷ Pseudo-inverse
    $(\mathbf{A}, \boldsymbol{b}) \leftarrow pinv \cdot \mathbf{S}_e$
    return $(\mathbf{A}, \boldsymbol{b})$

series n. Moreover, the complexity of the SYMEX algorithm is $\mathcal{O}(g)$, where g is the number of affine relationships produced by the algorithm; thus it is linear in the number of affine relationships g. In Section 6, we perform experiments for demonstrating the linear scalability of the SYMEX algorithm.

Lastly, we stress the fact that in the SYMEX algorithm it is not necessary to store and track all the affine relationships. We can, if required, prune the unnecessary affine relationships on the basis of domain knowledge, query requirements etc. This, however, is not the main focus of this paper, and would 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.

Pseudo-inverse cache.
Notice that, on Line 26, the SYMEX algorithm computes the pseudo-inverse of the matrix $[\mathbf{O}_p, \mathbb{1}_m]$. This is necessary for solving the system of equations for finding $\mathbf{A}$ and $\boldsymbol{b}$ by the least-squares method. Since there are many sequence pairs e associated to a single pivot pair p, the same pseudo-inverse of $[\mathbf{O}_p, \mathbb{1}_m]$ is repeatedly re-computed for each pivot pair.

Thus, we propose another variant of the SYMEX algorithm that caches, instead of re-computing, the pseudo-inverse. We call this variant the SYMEX+ algorithm. The proposed pseudo-inverse cache is populated by inserting the pseudo-inverse of $[\mathbf{O}_p, \mathbb{1}_m]$ 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. As we shall demonstrate in Section 6, the SYMEX+ algorithm is a factor of *4 times* faster as compared to the SYMEX algorithm.

4.1 Measure Computation Query

We discuss the processing of Query 1, or the MEC query, 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 first step is the pre-processing step. This step fills the values in the empty hash map pivotHash, which is returned by the SYMEX+ algorithm. For each pivot pair p, contained in the pivotHash hash map, the value is set to the covariance matrix of the pivot pair matrix $\mathbf{O}_p$. Our task is to compute $\Sigma_{e_\psi}(\mathbf{S})$ for each sequence pair $e_\psi \in \mathcal{P}$. For performing this task we search for two things: (a) covariance of the pivot pair p_ψ in the pivotHash hash map, denoted as $\Sigma(\mathbf{O}_{p_\psi})$, and (b) affine relationship $(\mathbf{A}, \boldsymbol{b})_{e_\psi}$ for the sequence pair e_ψ in the affHash hash map. Using these inputs and Eq. (6) we compute $\Sigma_{e_\psi}(\mathbf{S}_\psi)$ as:

$$\Sigma_{e_\psi}(\mathbf{S}) = \Sigma_{12}(\mathbf{S}_{e_\psi}) = \boldsymbol{a}_1^\top \Sigma(\mathbf{O}_{p_\psi}) \boldsymbol{a}_2, \tag{15}$$

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 transformation matrix $\mathbf{A}_{e_\psi}$. This procedure is followed for all the other sequence pairs e_ψ.

Similarly, a MEC query requesting computation of a $\mathcal{L}$-measure, dot product, or $\mathcal{D}$-measure 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 and then utilized for normalization.

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 naive approach always computes all the covariance matrices, which are of order $\mathcal{O}(n^2)$. Moreover, as we shall see in Section 6, 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.

Error Measurement: Another important issue is the error measure used for characterizing the approximation error. 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, compute the normalized values $\hat{\Sigma}^n_e(\mathbf{S})$ and $\Sigma^n_e(\mathbf{S})$, by dividing $\hat{\Sigma}_e(\mathbf{S})$ and $\Sigma_e(\mathbf{S})$ with a normalizer $(\max(\hat{\Sigma}_e(\mathbf{S})) - \min(\hat{\Sigma}_e(\mathbf{S})))$, where the maximum and the minimum are computed over all the sequence pairs in $\mathcal{P}$. Next, we compute the RMSE (root-mean-square error) between the normalized values as follows:

$$\% \text{ RMSE} = \sqrt{\frac{\sum_{e \in \mathcal{P}} \left(\hat{\Sigma}^n_e(\mathbf{S}) - \Sigma^n_e(\mathbf{S}) \right)^2}{|\mathcal{P}|}} \times 100 \tag{16}$$

5 Indexing Affine Relationships

In this section we propose efficient methods for processing the MET and MER queries described in Section 2.2. A straight forward way of processing these queries is to either use the naive 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 this 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 all the statistical measures can be processed without re-computing the measure for every query, and (b) a single index can process queries for all the $\mathcal{L}$-, $\mathcal{T}$-, and $\mathcal{D}$-measures. Furthermore, using the SCAPE index we can improve the efficiency of processing the MET and MER queries by orders of magnitude.

The SCAPE index consists of a 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 measure-independent, only then we can index all the statistical measures using the same index. Additionally, the key should be such that a query (MET or MER) over any statistical measure could be converted into a query (MET or MER) over the keys stored by the SCAPE index, guaranteeing that the results from the converted and the original query are the same.

For choosing a key with the above properties, the SCAPE index uses 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, the scalar product can be defined as $\boldsymbol{\alpha}^\top \boldsymbol{\beta}_l = \|\boldsymbol{\alpha}\| \cdot \|\boldsymbol{\beta}_l\| \cos(\theta_l)$, where θ_l is the angle between $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}_l$. Notice, $\|\boldsymbol{\alpha}\|$ is common to all the $\|\boldsymbol{\alpha}\| \cdot \|\boldsymbol{\beta}_l\| \cos(\theta_l)$, therefore it is sufficient to use $\|\boldsymbol{\beta}_l\| \cos(\theta_l)$ as a key for ordering the scalar product (refer Fig. 6). $\|\boldsymbol{\beta}_l\| \cos(\theta_l)$ is known as the *scalar projection* of $\boldsymbol{\beta}_l$ on $\boldsymbol{\alpha}$, and is denoted as ξ_l. The above example encourages us to formulate the following observation:

Observation 1 *Given a vector $\boldsymbol{\alpha}$ and vectors $\boldsymbol{\beta}_l$, where l is a positive integer. The scalar projections $\xi_l = \|\boldsymbol{\beta}_l\| \cos(\theta_l)$ can be used as a key for ordering the scalar products $\boldsymbol{\alpha}^\top \boldsymbol{\beta}_l$.*

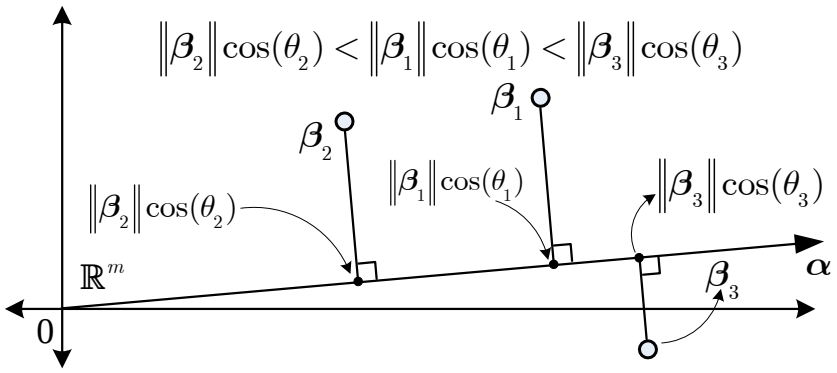


Fig. 6. Toy example demonstrating Observation 1.

5.1 Scalar Projection (SCAPE) Index

Now let us discuss the application of Observation 1 for indexing affine relationships. Assume that we obtained $\mathcal{Q}$ pivot pairs by executing the SYMEX+ algorithm described in Section 4. 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, 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:

$$\Sigma_{e_{qd}}(\mathbf{S}) = \boldsymbol{a}_2^\top \Sigma(\mathbf{O}_{p_q}) \boldsymbol{a}_1, \tag{17}$$

where $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ are first and second column of the transformation matrix $\mathbf{A}_{e_{qd}}$. Since from Definition 2, we have a common time series, namely 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. (17) becomes:

$$\Sigma_{e_{qd}}(\mathbf{S}) = (a_{12}, a_{22}) \begin{pmatrix} \Sigma_{11}(\mathbf{O}_{p_q}) \\ \Sigma_{21}(\mathbf{O}_{p_q}) \end{pmatrix}. \tag{18}$$

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, for ordering the scalar products $\boldsymbol{\alpha}_q^\top \boldsymbol{\beta}_{qd}$ it is sufficient to order only 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 an $\boldsymbol{\alpha}_q$ for other measures without affecting the ordering of the key ξ_{qd}. Thus, like covariance, we can find an $\boldsymbol{\alpha}_q$ for the other $\mathcal{L}$-measures and the dot product. Table 2 summarizes the values of $\boldsymbol{\alpha}_q$ and $\boldsymbol{\beta}_{qd}$ for all the $\mathcal{L}$- and $\mathcal{T}$-measures. In summary, by using the same ordering of ξ_{qd} we can index all the $\mathcal{L}$- and $\mathcal{T}$-measures considered in this paper.

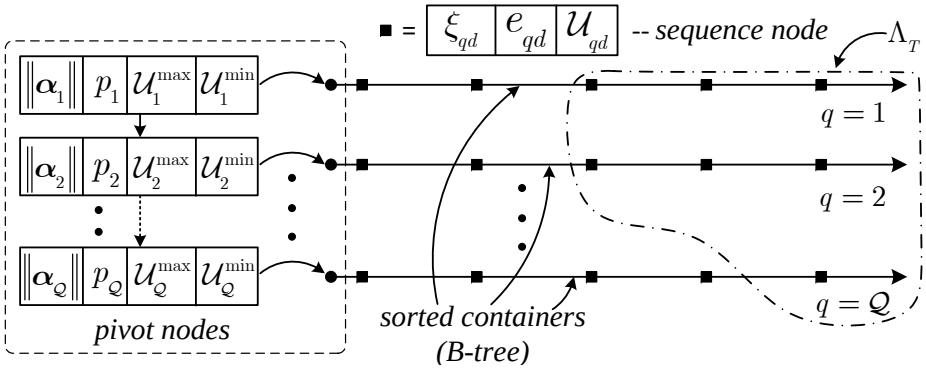


Fig. 7. Example of the SCAPE index for indexing a $\mathcal{T}$-measure and a $\mathcal{D}$- measure.

Moreover, the SCAPE index contains two types of nodes: (a) *pivot node* that includes the pivot pair p_q and $\|\boldsymbol{\alpha}_q\|$ for all the statistical measures that are indexed by the SCAPE index, and (b) *sequence node* that includes the sequence pair e_{qd} and the scalar projection $\xi_{qd} = \|\boldsymbol{\beta}_{qd}\| \cos(\theta_{qd})$. Furthermore, all the sequence nodes, associated with a pivot node, are stored in a sorted container, like a B-tree. The key for sorting is the scalar projection ξ_{qd}, which is found in each sequence node. In addition, each pivot node also stores the reference to the sorted container that stores its sequence nodes. A schematic depicting the arrangement between the pivot nodes and the sequence nodes is shown in Fig. 7. Thus, in short, using the SCAPE index we have essentially indexed all the $\mathcal{L}$- and $\mathcal{T}$-measures at once.

Table 2. Choices of $\boldsymbol{\alpha}_q$ and $\boldsymbol{\beta}_{qd}$. The third column refers to the affine relationship $(\mathbf{A}, \boldsymbol{b})$ between p_q and e_{qd}.

	$\boldsymbol{\alpha}_q$	$\boldsymbol{\beta}_{qd}$
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 $\Pi_{12}(\mathbf{S}_{e_{qd}})$	$(\Pi_{12}(\mathbf{O}_{p_q}), \Pi_{11}(\mathbf{O}_{p_q}), h_1(\mathbf{O}_{p_q}))$	$(a_{12}, a_{22}, b_2)^\top$

Indexing $\mathcal{D}$-Measures: For indexing a $\mathcal{D}$-measure, we should additionally store the following two values with the existing SCAPE index structure. First, in each sequence node, the normalizer $\mathcal{U}_{qd}$ of the indexed $\mathcal{D}$ measure, for e.g., $\sqrt{\Sigma(\boldsymbol{s}_u)\Sigma(\boldsymbol{s}_v)}$ for the correlation coefficient. Second, in each pivot node, the maximum and the minimum value 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 5.3, we show that the above two quantities 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 with the SCAPE index.

5.2 Processing Threshold and Range Queries

Consider the MET query requesting the sequence pairs such that the covariance is greater than a user-defined threshold τ. We obtain the converted query by dividing τ by $\|\boldsymbol{\alpha}_q\|$. We call this the modified threshold $\tau' = \frac{\tau}{\|\boldsymbol{\alpha}_q\|}$. For computing the modified threshold τ' the value of $\boldsymbol{\alpha}_q$ corresponding to covariance in Table 2 is used. Note that the this conversion gaurantees that the result set of the original and the converted query are the same. Next, we scan all the B-trees associated with all the pivot nodes, using a binary search algorithm and collect e_{qd}, such that $\tau' > \xi_{qd}$. Fig. 7 shows an example of this process. The collected set of e_{qd} is the result set Λ_T of the MET query.

Secondly, consider the measure range query requesting all the sequence pairs, such that their covariance is in between thresholds τ_l and τ_u. Similar to the MET query, we compute the modified thresholds: $\tau_l' = \frac{\tau_l}{\|\boldsymbol{\alpha}_q\|}$ and $\tau_u' = \frac{\tau_u}{\|\boldsymbol{\alpha}_q\|}$. We, then, collect all the e_{qd} from all the B-trees using a binary search such that $\tau_l' < \xi_{qd} < \tau_u'$. The collected set of e_{qd} is the result set Λ_R of the measure range query.

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

Processing the MET and MER queries over the $\mathcal{D}$-measures is a challenging problem. Recall that a $\mathcal{D}$-measure is derived by normalizing a $\mathcal{T}$-measure. The

primary challenge is that normalization destroys the ordering of the scalar projections ξ_{qd}, which is established for processing queries for the $\mathcal{L}$- and $\mathcal{T}$-measures. Now, the idea here is to prune the sequence pairs stored in a sorted container 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, which is a $\mathcal{D}$-measure, is greater than τ. We start the processing by considering each pivot node at a time. For a given pivot node, we compute the two modified thresholds: $\tau'_{\min} = \frac{\tau \cdot \mathcal{U}_q^{\min}}{\|\boldsymbol{\alpha}_q\|}$ and $\tau'_{\max} = \frac{\tau \cdot \mathcal{U}_q^{\max}}{\|\boldsymbol{\alpha}_q\|}$. Observe that the sequence nodes, associated to a pivot node, where $\xi_{pd} > \tau'_{\max}$, are definitely in the result set Λ_T, and do not require further processing. This situation in depicted in Fig. 8(a) and holds because of the following:

$$\xi_{pd} > \tau'_{\max} \Leftrightarrow \frac{\|\boldsymbol{\alpha}_q\| \cdot \xi_{qd}}{\mathcal{U}_q^{\max}} > \tau \Leftrightarrow \rho_{e_{qd}}(\mathbf{S}) > \tau. \tag{19}$$

Thus for all the sequence nodes where $\xi_{pd} > \tau'_{\max}$ the correlation coefficient can only be greater than τ.

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.

Similarly, consider a measure range query that is requesting all the sequence pairs such that their correlation coefficient is between τ_l and τ_u. As before, 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.

Furthermore, 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 *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.

Note that the same index pruning techniques can be utilized for other $\mathcal{D}$-measures. In Section 6, we compare the query processing methods using the SCAPE index with: (a) a method that uses affine relationships to compute the statistical measure and then process the MET or MER, and (b) a method that first computes the statistical measure from scratch and then processes the MET or MER query. Our experiments show that by using the SCAPE index structure

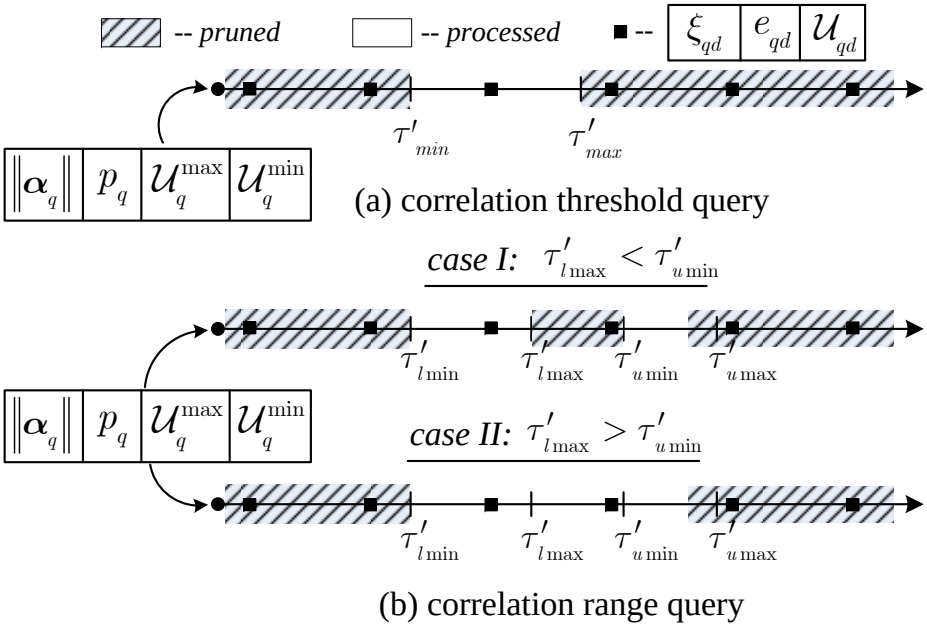


Fig. 8. Index-based pruning for processing MET and MER queries on $\mathcal{D}$-measures.

we obtain a dramatic improvement in performance as compared to the other methods.

6 Experimental Evaluation

In this section we perform extensive experimental evaluation on real datasets to establish the efficacy of our approaches. In Section 6.1, we analyze the trade-off between accuracy and efficiency for computing statistical measures using affine relationships. We emphasize the performance improvements in query processing using synthetic – but realistic – workloads in Section 6.2. The scalability of the SYMEX algorithm is established in Section 6.3, and the performance gains by using the SCAPE index are demonstrated in Section 6.4. Since we have more than one method for computing and querying the statistical measures, as a shorthand we use the following notations:

- $\mathcal{W}_N$: the naive method that computes a given statistical measure from scratch,
- $\mathcal{W}_A$: the affine relationships method that uses affine relationships for computing a statistical measure (refer Section 4.1),
- $\mathcal{W}_F$: an approach that uses the five largest DFT (Discrete Fourier Transform) coefficients for approximating the correlation coefficient, and has been introduced in [1–3].

In this paper we use two real datasets. The first dataset contains 670 daily time series obtained from 134 sensors monitoring environmental parameters on a university 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 3.

Table 3. 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

6.1 Analyzing Trade-Off

For analyzing the tradeoff between efficiency and accuracy we consider a MEC query that computes a statistical measure ($\mathcal{L}$, $\mathcal{T}$, or $\mathcal{D}$) over *all* the time series present in a dataset. Fig. 9 and Fig. 10 show the speedup and the percentage RMSE error (refer Section 4.1) obtained for all the 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. To give a sense of the absolute times, in Fig. 11 we show the absolute time comparison for *stock-data*.

In particular, for computing statistical measures, the focus of our work, the errors are negligible. 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 and the minimum speedups of *1.3x* and *4x* are obtained for dot product and mean respectively. The speedup obtained for mean and dot product is low due to the inherent simplicity of computing them using the $\mathcal{W}_N$ method. Thus, in summary, the $\mathcal{W}_A$ method exhibits significant improvements in efficiency and accuracy.

Since the *stock-data* is larger than the *sensor-data*, the efficiency improvement for *stock-data* is more prominent than *sensor-data*. This demonstrates that our approaches are capable of effectively handling large datasets. Moreover, for all the statistical measures a small number of clusters (6) are sufficient to obtain high accuracy; thus resulting in a nearly linear cost of processing the MEC query.

6.2 Impact of Online Environments

Our task here is to analyze how the AFFINITY framework handles MEC queries posed in online environments. Typically, in online environments, users frequently

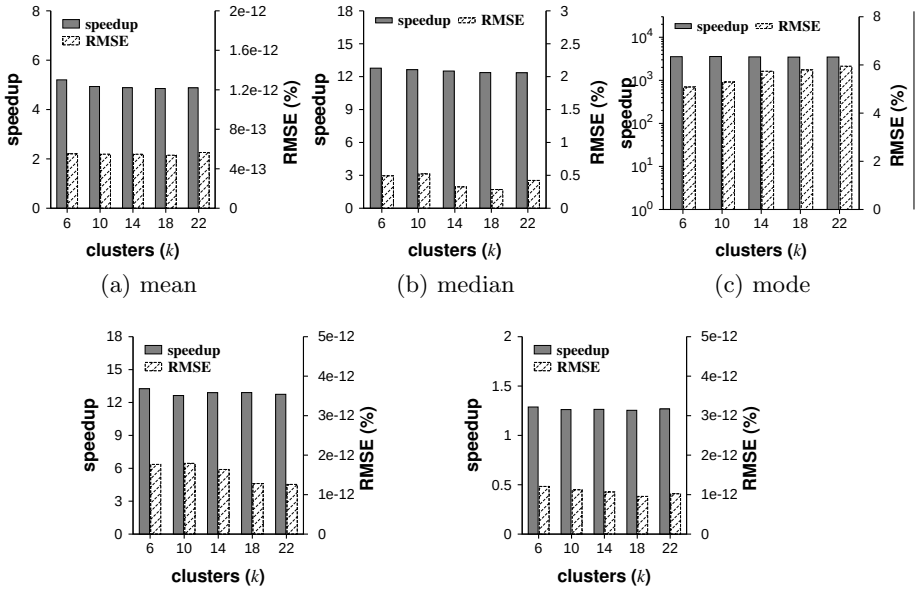


Fig. 9. Efficiency and accuracy tradeoff for *sensor-data*. Note the logarithmic scale for the speedup in (c).

request for computation of a particular statistical measure for only few entities (stocks or sensors). To simulate this behavior, we generate realistic query workloads as follows: each 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 intuition behind the powerlaw distribution is that since some entities (stocks or sensors) are popular as compared to others, thus we model their popularity with a powerlaw distribution.

Fig. 12 compares query processing performance as the number of queries increase for the *sensor-data* and the *stock-data*. 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, and it is *2.5 to 9 times* faster when 15k queries are processed. Note that the time for the $\mathcal{W}_A$ method shown in Fig. 12 also includes the initial time taken by the SYMEX+ algorithm for computing the affine relationships.

Thus, the proposed $\mathcal{W}_A$ method is far superior than the $\mathcal{W}_N$ method of query processing and is suitable for deployment in online environments. Here we cannot compare with the $\mathcal{W}_F$ method, since the $\mathcal{W}_F$ method only computes the correlation coefficient and does not work for all the other statistical measures.

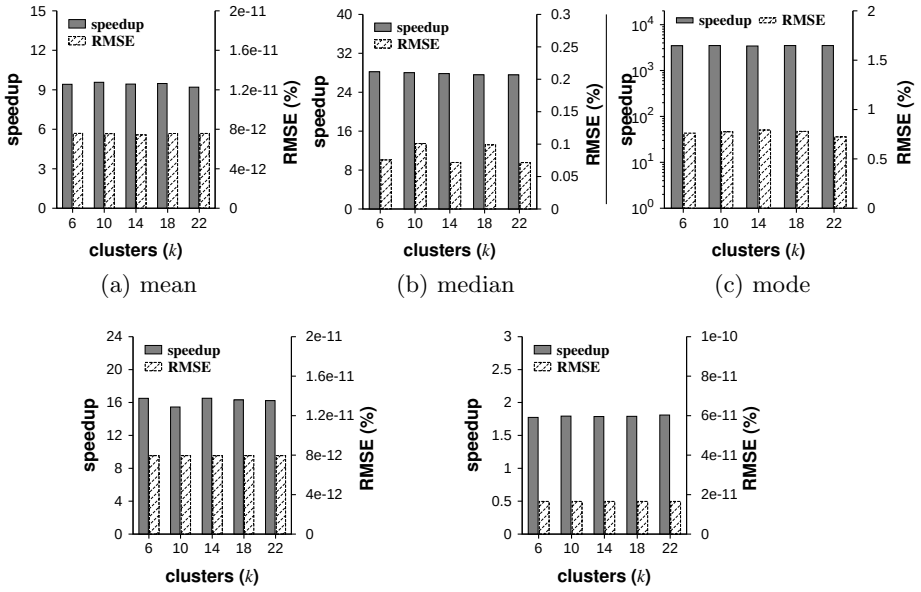


Fig. 10. Efficiency and accuracy tradeoff for *stock-data*. Note the logarithmic scale for the speedup in (c).

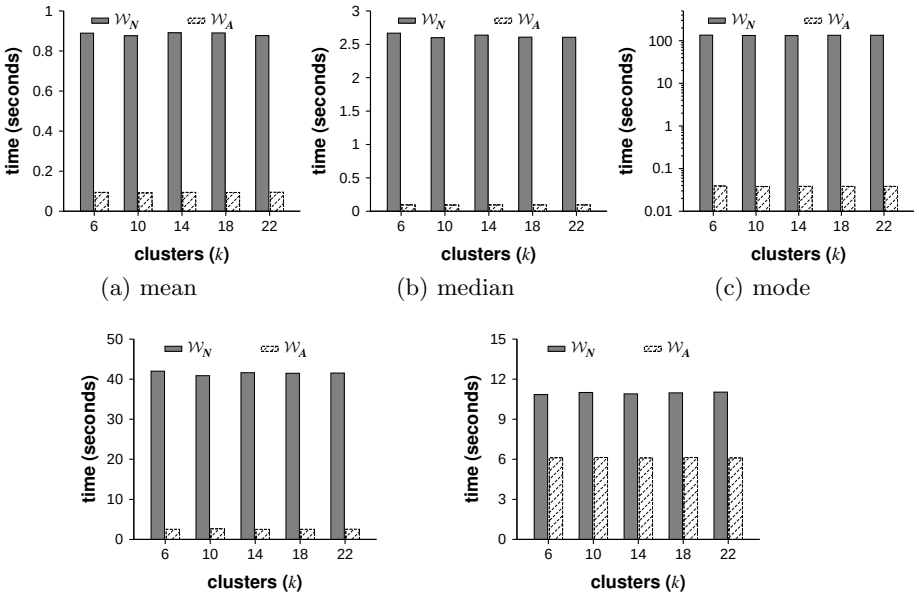
6.3 Scalability of the SYMEX Algorithm

Now we compare scalability of the SYMEX and SYMEX+ algorithms when the number of affine relationships generated by them increases. Fig. 13 shows the scaling behavior of the SYMEX and the SYMEX+ algorithms as the number of affine relationships handled by these algorithms increase. Clearly, the SYMEX and the SYMEX+ algorithms scale linearly as the number of affine relationships increase. Particularly, the SYMEX+ algorithm is a factor *3.5 to 4 times* faster as compared to the simple SYMEX algorithm. For experiments in Fig. 13, we set $k = 6$, $\gamma_{max} = 10$, and $\delta_{min} = 10$ as the parameters of the AFCLST algorithm. Thus, the SYMEX+ algorithm exhibits attractive improvements as compared to the SYMEX algorithm.

6.4 Impact of using the SCAPE Index

We discuss the performance improvements obtained by using the SCAPE index. Here, all the experiments are performed on the *sensor-data*. Recall, the SCAPE index uses the affine relationships returned by the SYMEX+ algorithm. For processing the MET and MER queries on the correlation coefficient the index pruning methods discussed in Section 5.3 are utilized.

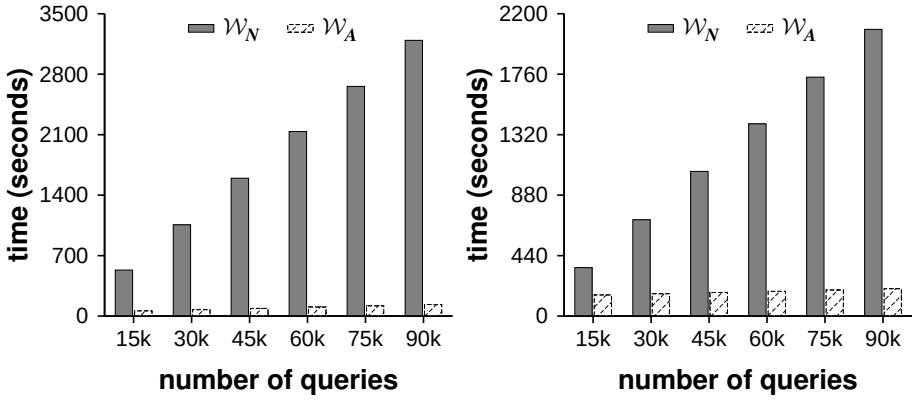
We first analyze the scalability of constructing the SCAPE index as the number of indexed affine relationships increase. Fig. 14 shows the scaling behavior



(a) mean (b) median (c) mode

(d) covariance (e) dot product

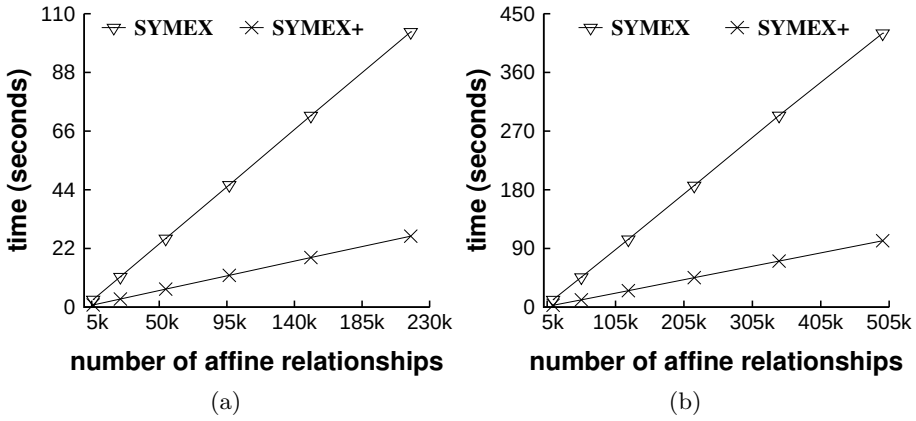
Fig. 11. Absolute time comparison for *stock-data*. Note the logarithmic scale for the speedup in (c).



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

Fig. 12. Comparing query processing efficiency.

of the SCAPE index when it indexes the affine relationships for a $\mathcal{T}$-measure (covariance) and a $\mathcal{L}$-measure (mean). Clearly, the SCAPE index exhibits linear scaling, which makes it a viable practical alternative for query processing.



(a) (b)

Fig. 13. Scalability of the SYMEX algorithm. (a) *sensor-data* and (b) *stock-data*.

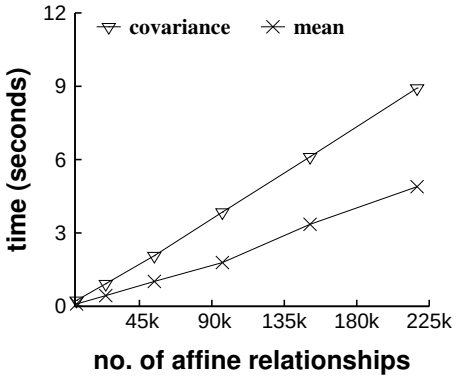
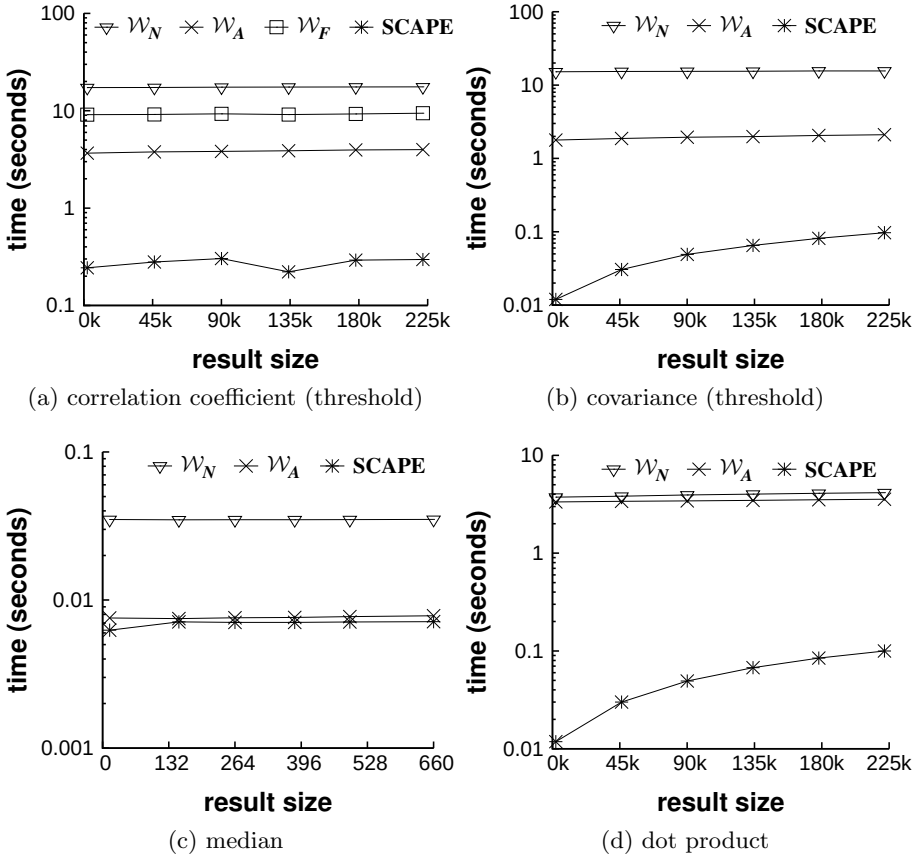


Fig. 14. Scalability of the index construction on *sensor-data*.

Next, we compare the performance improvement obtained by using the SCAPE index for processing the MET and MER queries for the covariance and the correlation coefficient. Here, all the affine relationships that are returned by the SYMEX+ algorithm are used for creating the SCAPE index. Fig. 15 and Fig. 16 compares the results for query processing obtained using the SCAPE index with the other methods. 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, therefore it is only include in Fig. 15(a) and Fig. 16(a).

Fig. 15 and Fig. 16 depict the orders of magnitude improvement (shown using logarithmic scale) in efficiency while processing the MET and MER queries using the SCAPE index. Table 4 shows a snapshot of the orders of magnitude 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.



(a) correlation coefficient (threshold) (b) covariance (threshold)

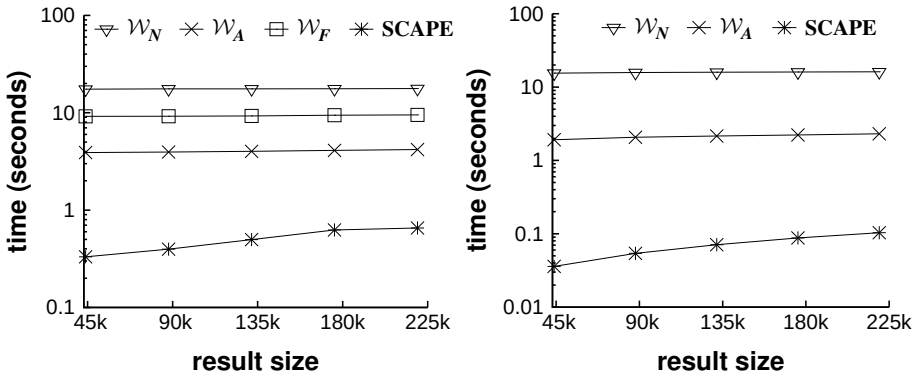
(c) median (d) dot product

Fig. 15. Comparing efficiency of the SCAPE index for the MET query.

It is clearly evident from Table 4, that by using the SCAPE index we obtain orders of magnitude performance improvement. There is, however, one exception – median. Since median is a $\mathcal{L}$-measure, the maximum possible number of affine relationships for it are low (linear in n). These affine relationships are insufficient for demonstrating the efficacy of the SCAPE index. In summary, the proposed indexing methods exhibit tremendous improvement in the efficiency of processing MET and MER queries.

7 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 using the largest DFT coefficients [1–3]. Computing the Pearson's correlation coefficient using DFT-based techniques provides inaccurate results when the time series contain white



(a) correlation coefficient (b) covariance

Fig. 16. Comparing efficiency of the SCAPE index for the MER query.

Table 4. Query processing speedup computed when the query returns the maximum size of the result set Λ_T or Λ_R.

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	×

noise. Cole *et al.* [4] call such time series *uncooperative* and propose methods for discovering correlation amongst such signals. All these studies, however, typically only consider the correlation coefficient and do not propose an unified approach for computing and querying a wide variety of statistical measures, which includes the correlation coefficient.

In addition to computing the correlation coefficient, there has been a large body of related prior research using the DFT for: (a) exact or approximate sequence matching where the sequences could have undergone a similarity transformation [13–15], (b) retrieving similar shapes [16, 17], (c) predicting future values and answering similarity queries [18], and (d) reducing the dimensionality of the time-series data [19, 20]. 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 [21], on the other hand, uses PCA (Principal Component Analysis) for summarizing a 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. A method of computing correlated aggregates is proposed in [22]. The Cypress framework [6] uses Fourier transform and random projection based multi-scale analysis for segmenting the data into various form of trickles, which are then used for query processing. Similarly, GAMPS [23] uses ratio signals for compressing time-series data and proposes approaches for query processing over such compressed data. More recently, there has been research conducted on indexing and querying correlated uncertain information using probabilistic databases [24, 25]. Lastly, Ke *et al.* [26] propose approaches for searching graphs correlated to a given query graph.

8 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 clustering the time-series data, such that high-quality affine relationships could be found. We proposed the SYMEX and SYMEX+ algorithms that are capable of computing affine relationships in linear time. We demonstrated that the SCAPE index structure can easily index all the statistical measures and produce orders of magnitude improvement in efficiency for processing measure threshold and range queries, as compared to naive methods and methods proposed in the literature for this problem. Comprehensive experiments highlight 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. R. Agrawal, C. Faloutsos, and A. Swami, "Efficient similarity search in sequence databases," in *FODO*, 1993, pp. 69–84.
14. 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.
15. C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast subsequence matching in time-series databases," in *SIGMOD*, 1994, pp. 419–429.
16. D. Rafiei and A. Mendelzon, "Similarity-based queries for time series data," in *SIGMOD*, 1997, pp. 13–25.
17. H. Jagadish, A. Mendelzon, and T. Milo, "Similarity-based queries," in *PODS*, 1995, pp. 36–45.
18. X. Lian and L. Chen, "Efficient similarity search over future stream time series," *TKDE*, vol. 20, no. 1, pp. 40–54, 2008.
19. 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.
20. ——, "Dimensionality reduction for fast similarity search in large time series databases," *KAIS*, vol. 3, no. 3, pp. 263–286, 2001.
21. S. Papadimitriou, J. Sun, and C. Faloutsos, "Streaming pattern discovery in multiple time-series," in *VLDB*, 2005, pp. 697–708.
22. J. Gehrke, F. Korn, and D. Srivastava, "On computing correlated aggregates over continual data streams," in *SIGMOD*, 2001, pp. 13–24.
23. 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.
24. C. Ré, J. Letchner, M. Balazinksa, and D. Suciu, "Event queries on correlated probabilistic streams," in *SIGMOD*, 2008, pp. 715–728.
25. B. Kanagal and A. Deshpande, "Indexing correlated probabilistic databases," in *SIGMOD*, 2009, pp. 455–468.
26. Y. Ke, J. Cheng, and W. Ng, "Correlation search in graph databases," in *SIGKDD*, 2007, pp. 390–399.