Querying Time Series Data Based on Similarity

Davood Rafiei*
Department of Computing Science
University of Alberta
drafiei@cs.alberta.ca

Alberto O. Mendelzon
Department of Computer Science
University of Toronto
mendel@cs.toronto.edu

Abstract

We study similarity queries for time series data where similarity is defined, in a fairly general way, in terms of a distance function and a set of affine transformations on the Fourier series representation of a sequence. We identify a safe set of transformations supporting a wide variety of comparisons and show that this set is rich enough to formulate operations such as moving average and time scaling. We also show that queries expressed using safe transformations can efficiently be computed without prior knowledge of the transformations. We present a query processing algorithm that uses the underlying multidimensional index built over the data set to efficiently answer similarity queries. Our experiments show that the performance of this algorithm is competitive to that of processing ordinary (exact match) queries using the index, and much faster than sequential scanning. We propose a generalization of this algorithm for simultaneously handling multiple transformations at a time, and give experimental results on the performance of the generalized algorithm.

Keywords: *Similarity Queries, Time Series Retrieval, Indexing Time Series Fourier Transform.*

1 Introduction

Time-series data are of growing importance in many new database applications, such as data mining or data warehousing. A time series is a sequence of real numbers, each number representing a value at a time point. For example, the sequence could represent stock or commodity prices, sales, exchange rates, weather data, biomedical measurements, etc. We are often interested in similarity queries on time-series data [APWZ95, ALSS95]. For example, we may want to find stocks that behave in approximately the same way (or approximately the opposite way, for hedging); or products that had similar selling patterns during the last year; or years when the temperature patterns in two regions of the world were similar. In queries of this type, approximate, rather than exact, matching is required.

*This work was done when the author was a graduate student in *Computer Science Department of the University of Toronto.*

A simple approach to determine a possible similarity between two time series is to compute the Euclidean distance (or any other distance, such as the city-block distance) between the two series, and call the two series similar if their distance is less than some user-defined threshold.

However, the notion of similarity is often more complex and cannot be expressed using a distance function. The meaning of a similarity comparison may vary from one data domain to another data domain and different users may have different perceptions of a similarity comparison; for example, one may consider two stocks similar if they have almost the same price fluctuations, even though one stock might sell for twice as much as the other. Consider the following motivating examples.

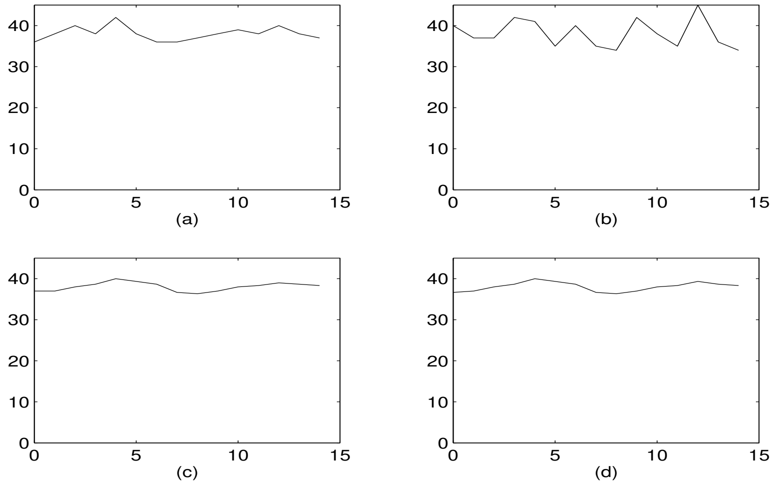


Figure 1: (a) Time series $\vec{s_1}$ = [36,38,40,38,42,38,36, 36,37,38,39,38,40,38,37], (b) time series $\vec{s_2}$ = [40,37,37,42,41,35,40,35,34,42,38,35,45,36,34], (c) the 3-day moving average of $\vec{s_1}$, and (d) the 3-day moving average of $\vec{s_2}$

Example 1.1 Consider time series $\vec{s_1} = [36, 38, 40, 38, 42, 38, 36, 36, 37, 38, 39, 38, 40, 38, 37]$ and $\vec{s_2}$ $=]40, 37, 37, 42, 41, 35, 40, 35, 34, 42, 38, 35, 45, 36, 34]$, for example corresponding to the closing prices of two stocks. Looking at Figure 1(a),(b), the sequences do not appear very similar. This is justified by the high Euclidean distance $D(\vec{s_1}, \vec{s_2}) = 11.92$ between them. However, if we look at the three-day moving averages of the two sequences (Figure 1 (c),(d)), they do look quite similar. The Euclidean distance between the three-day moving averages of the two sequences is 0.47.

Moving averages are widely used in stock data analysis (for example, see [EM69]). Their primary use is to smooth out short term fluctuations and depict the underlying trend of a stock. The computation is simple; the l-day moving average of a sequence $\vec{s} = [v_1, \ldots, v_n]$ is computed as follows: the mean is computed for an l-day-wide window placed over the end of the sequence; this will give the moving average for day $n - \lfloor l/2 \rfloor$; the subsequent values are obtained by stepping the window through the beginning of the sequence, one day at a time. This will produce a moving average of length $n - l + 1$. We use a slightly different version of moving average which is easier to compute in our framework. We circulate the window to the end of the sequence when it reaches the beginning. This gives us a moving average of length n. It turns out that when the length of the window is small enough compared to the length of the sequence, which is usually the case in practice, both averages are almost the same.

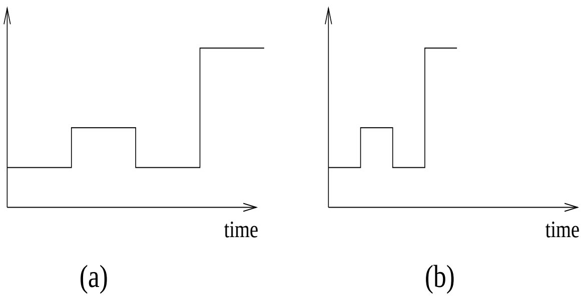


Figure 2: (a)Time series $\vec{s} = [20, 20, 21, 21, 20, 20, 23, 23]$ (b)time series $\vec{p} = [20, 21, 20, 23]$

Example 1.2 Consider two time series $\vec{s} = [20, 20, 21, 21, 20, 20, 23, 23]$ and $\vec{p} = [20, 21, 20, 23]$ that are sampled with different frequencies (Figure 2). For example, $\vec{s}$ could be the closing price of a stock taken every day, and $\vec{p}$ could be the closing price of another stock taken every other day. A typical query is "is $\vec{p}$ similar to $\vec{s}$?". The sequence $\vec{s}$ is twice as long as $\vec{p}$, so they cannot be compared directly. The Euclidean distance between $\vec{p}$ and any subsequence of length four of $\vec{s}$ is more than 1.41. If the time dimension of $\vec{p}$ is scaled by 2, i.e., every value "v_j" is replaced by "v_j, v_j", the resulting sequence will be identical to $\vec{s}$. This operation is a special kind of *time warping* (for example, see [SK83]), often called *time scaling*, which can be expressed within our framework.

We propose a class of transformations that can be used in a query language to express similarity in a fairly general way, handling a wide variety of comparisons including cases like the two examples above. We show that the query evaluation for similarity comparisons can be efficiently implemented on top of a point-based access method such as R-tree [Gut84] without a prior knowledge of transformations. For example, we demonstrate that an index structure for moving average can be constructed on the fly from an existing index (built on the original sequences) and the query evaluation engine can benefit from the new index in the same way as it does from the original index with no extra disk overhead. To the best of our knowledge, this is the first indexing method that can handle moving average and time scaling in the context of similarity queries.

To apply this framework to time series data, we need to make some choices on how we compare two sequences and also how we do indexing. We have chosen the Euclidean distance to compare two sequences mainly because: (a) it easily corresponds to the cross correlation (Eq.13); (b) it is frequently used [AFS93, FRM94]; (c) it remains the same under orthonormal transforms [1]. There are several multidimensional indexes (such as R-tree family

[1] A transformation, denoted with matrix M, is orthonormal if $M^T.M = I$ where M^T denotes the transpose of matrix M and I represents the identity matrix.

[Gut84, BKSS90, SK96], the k-d-B-tree [LS90] or the grid file [NHS84]) that can be used to do indexing on sequences. All these indexes reduce to sequential scanning in higher dimensions due to a phenomenon known as the *curse of dimensionality*. A solution to avoid this problem is to choose a few important features, those that can approximately differentiate one sequence from another, and only keep those features in the index. The approximation introduces a few false hits when sequences are being compared. However, the false hits can be easily removed in a postprocessing step.

For feature extraction, we use the Discrete Fourier Transform (DFT) to map sequences into the frequency domain and only keep the first few DFT coefficients for each sequence. The reason for choosing DFT is mainly because: (a) for a large class of time series (often referred as *colour noise data*), it concentrates the energy in a few lower frequency coefficients, so those coefficients can make a key for indexing purposes; (b) it is an orthonormal transform, so the Euclidean distance is unchanged under the DFT; (c) the class of transformations proposed in this paper (see Section 2) can express an interesting class of operations if applied in the frequency domain. In general, one can use any orthonormal transform such as the Discrete Cosine Transform (DCT), the Harr transform, the wavelet transform, etc. For a broad class of feature selection methods see, for example, the online bibliography maintained at the National Research Council Canada site [NC].

The organization of the rest of the paper is as follows: In the rest of the current section we provide some background material on the discrete Fourier transform and also review the related work. Our definition of similarity queries is discussed in Section 2. In Section 3 we use R-tree indexes and develop algorithms for efficiently evaluating queries expressed within our framework. In Section 4 we briefly describe how our algorithms can be implemented using the grid file and the k-d-B-tree. Section 5 presents experimental performance results. We conclude in Section 6.

1.1 The Discrete Fourier Transform

In this section, we briefly review the Discrete Fourier Transform and its properties. Let a time sequence be a finite duration signal $\vec{x} = [x_t]$ for $t = 0, 1, \cdots, n-1$. The DFT of $\vec{x}$, denoted by $\vec{X}$, is given by

$$X_f = \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t e^{\frac{-i2\pi tf}{n}} \quad f = 0, 1, \cdots, n-1 \tag{1}$$

where $i = \sqrt{-1}$ is the imaginary unit. Throughout this paper, unless it is stated otherwise, we use small letters for sequences in the time domain and capital letters for sequences in the frequency domain. The inverse Fourier transform of $\vec{X}$ gives the original signal, i.e.,

$$x_t = \frac{1}{\sqrt{n}} \sum_{f=0}^{n-1} X_f e^{\frac{i2\pi tf}{n}} \quad t = 0, 1, \cdots, n-1 \tag{2}$$

Some books define the DFT with no constant in front and the inverse DFT with constant $1/n$ in front or vice versa. Following some earlier conventions [AFS93, FRM94], we have $1/\sqrt{n}$

in front of both Equations 1 and 2 for it simplifies the upcoming Parseval relation without changing any properties of the DFT. The energy of signal $\vec{x}$ is given by the expression

$$E(\vec{x}) = \sum_{t=0}^{n-1} \mid x_t \mid^2 . \tag{3}$$

The convolution of two signals $\vec{x}$ and $\vec{y}$ is given by

$$Conv(\vec{x}, \vec{y})_j = \sum_{k=0}^{n-1} x_k y_{j-k} \quad j = 0, 1, \cdots, n-1 \tag{4}$$

where $j - k$ is computed modulo n. This convolution is usually called *circular convolution.* Equations 3 and 4 are unchanged in the frequency domain.

The following properties of DFT can be found in any signal processing textbook (for example, see [OS75]). The symbol $\Leftrightarrow$ denotes a DFT pair.

Linearity if $\vec{x} \Leftrightarrow \vec{X}$ and $\vec{y} \Leftrightarrow \vec{Y}$, then

$$a\vec{x} + b\vec{y} \;\Leftrightarrow a\vec{X} + b\vec{Y} \tag{5}$$

for arbitrary constants a and b,

Convolution-Multiplication if $\vec{x} \Leftrightarrow \vec{X}$ and $\vec{y} \Leftrightarrow \vec{Y}$, then

$$conv(\vec{x}, \vec{y}) \Leftrightarrow \vec{X} * \vec{Y} \tag{6}$$

where $\vec{X} * \vec{Y}$ is the element-to-element multiplication of two vectors $\vec{X}$ and $\vec{Y}$,

Symmetry if $\vec{x} \Leftrightarrow \vec{X}$ for a real-valued sequence $\vec{x}$ of length n, then

$$|X_{n-f}| = |X_f| \quad for\ f = 1, \ldots, n-1, \tag{7}$$

and

Parseval's Relation if $\vec{x} \Leftrightarrow \vec{X}$, then

$$E(\vec{x}) = E(\vec{X}). \tag{8}$$

Using Parseval's relation and the linearity, it is easy to show that the Euclidean distance between two signals in the time domain is the same as their distance in the frequency domain.

$$D^2(\vec{x}, \vec{y}) = E(\vec{x} - \vec{y}) = E(\vec{X} - \vec{Y}) = D^2(\vec{X}, \vec{Y}) \tag{9}$$

1.2 Related Work

An indexing technique for the fast retrieval of similar time sequences is proposed by Agrawal et al. [AFS93]. The idea is to use Discrete Fourier Transform (DFT) to map time sequences (stored in a database) into the frequency domain. Keeping only the first k Fourier coefficients, each sequence becomes a point in a k-dimensional feature space. To allow a fast retrieval, the authors keep the first k Fourier coefficients of a sequence in an R-tree index. There have been several extensions and improvements to this technique. An extension for subsequence matching is proposed by Faloutsos et al. [FRM94]. None of the aforementioned work allows the expression of a query that uses some transformations in its expression of similarity. Goldin et al. [GK95] show that the similarity retrieval will be roughly invariant to simple translations and scales if sequences are normalized before being stored in the index. The authors store in the index both the translation and the scale factors, in addition to normalized sequences, and also allow those factors to be queried using range predicates. In our earlier work [Raf98, RM98], we have shown that the last few Fourier coefficients of a sequence (those corresponding to lower negative frequencies) are as important as the first few coefficients due to the symmetry property of DFT for real-valued sequences. We have also shown that this observation can be used to reduce the size of the search rectangle in the indexing method of Agrawal et al. [AFS93] and its extensions [FRM94, GK95] without really storing the last few Fourier coefficients in the index. This leads to a search time improvement of more than a factor of 2. In this paper, we use this improved version of the index for efficiently evaluating queries that use transformations in their expressions of similarities.

Jagadish et al.[JMM95] develop a domain-independent framework to pose similarity queries on a database. The framework has three components: a *pattern language* P, a *transformation rule language* T, and a *query language* L. An expression in P specifies a set of data objects. An object A is considered similar to an object B, if B can be reduced to it by a sequence of transformations defined in T. The query language proposed in the paper is an extension of relational calculus with predicates that test whether an object A can be transformed into a member of the set of objects described by expression e using the transformation t, at a cost bounded by c. As a specialization of this work to real-valued sequences [FJMM97], the same authors describe how the search can be performed over sequence signatures instead of the original sequences. Our work here can also be seen as a specialized variation of this general framework where transformations are restricted to affine transformations.

There is other work on time series data. Yi et al. [YJF98] use time warping as a distance function and present algorithms for retrieving similar time sequences under this function. Chu et al. [CW99] show how similar sequences can be efficiently searched irrespective of differences in simple translations and scales. A hierarchical sequence matching algorithm based on the correlation between sequences is proposed by Li et al. [LYC96]. Agrawal et al. [APWZ95] describe a pattern language called SDL to encode queries about "shapes" found in time sequences. The language allows a kind of blurry matching where the user specifies the overall shape instead of the specific details, but it does not support any operations or transformations on sequences. A method for approximately representing sequences in terms of some functions and processing queries over such a representation is described by Shatkay and Zdonik [SZ96]. A query language for time series data in the stock market domain is developed by Roth [Rot93]. The language is built on top of CORAL [RS92], and every query is translated into a sequence of CORAL rules. Seshadri et al. [SLR94] develop a data model and a query language for sequences in general but do not mention similarity matching as a query language operator.

2 Similarity Queries

Time series often have differences that need to be removed or reduced before comparing them to each other. One way to remove those differences is to apply some transformations to them. We are interested in a set of transformations which fulfills the following two requirements: (a) it expresses a large class of useful transformations on time series, (b) queries expressed using this set of transformations can be efficiently processed. The set of affine transformations seems to be a good candidate for its members can express any combination of four important transformations in space: *translation*, *scaling*, *rotation* and *shear*. Furthermore, any affine transformation preserves two basic features namely *collinearity* (i.e., all points lying on a line initially still lie on a line after transformation) and *ratios of distances* (e.g., the midpoint of a line segment remains the midpoint after transformation). To fulfill the second requirement, we limit our transformations to *translation* and *scaling* (along all dimensions) only. Specifically, we define a transformation in an n-dimensional space, denoted by $(\vec{a}, \vec{b})$, as a pair of vectors where $\vec{a}$ specifies a stretch and $\vec{b}$ represents a translation. We will show that this class can express a large class of useful transformations on time series; at the same time, queries expressed using this class can be efficiently processed.

Definition 1 The transformation $(\vec{a}, \vec{b})$ applied to a point $\vec{x}$ in an n-dimensional space maps $\vec{x}$ to $\vec{a} * \vec{x} + \vec{b}$.

Transformations may be associated with costs. Given a set of transformations T, and the cost of applying each transformation, a measure of distance (dissimilarity) between two time series can be defined as follows:

$$D(\vec{x}, \vec{y}) = min \begin{cases} D_0(\vec{x}, \vec{y}) \\ min_{t \in T}(cost(t) + D(t(\vec{x}), \vec{y})) \\ min_{t \in T}(cost(t) + D(\vec{x}, t(\vec{y}))) \\ min_{t_1, t_2 \in T}(cost(t_1) + cost(t_2) \\ \qquad +D(t_1(\vec{x}), t_2(\vec{y}))) \end{cases} \tag{10}$$

where $D_0(\vec{x}, \vec{y})$ is a base distance (such as the Euclidean distance) between $\vec{x}$ and $\vec{y}$.

Next we propose an extension of multidimensional similarity queries where similarity is defined on the basis of both a distance function and a set of transformations.

2.1 Queries with Single Transformations

Transformations can be seen as a way to remove certain variations before aligning two sequences. Although many kinds of variations may be present in each sequence, we consider

$$\begin{tikzcd} \vec{x} = [11.3, 11.5, \ldots] \arrow[r, "\mathbf{f : 10dmv, \ldots}"] \arrow[d, "\mathrm{DFT}"'] & 10dmv(\vec{x}) = [11.3, 11.8, \ldots] \arrow[d, "\mathrm{DFT}"] \\ \vec{X} = [107, 3.3\text{-}3i, \ldots] \arrow[r, "\mathbf{t : (\vec{a}\,,\, \vec{b})}"] & T(\vec{X}) = [107, 2.5\text{-}3.5i, \ldots] \end{tikzcd}$$

Figure 3: Applying transformations to time series

only those that can be removed using the limited form of affine transformations on the Fourier series representation of the sequence (Figure 3). We first consider queries that use a single transformation to remove such variations. To gain some insight into this class of queries, we give some examples from real stock data. The data was obtained from the FTP site “ftp.ai.mit.edu/pub/stocks/results/”.

Example 2.1 Figure 4 shows the daily closing price for *The Bombay Co.* (BBA) starting from October 25th, 1994 for 128 days, and that for *Zweig Total Return Fund Inc.* (ZTR) starting from July 20th, 1995 for 128 days. The Euclidean distance between the two series is 16.16. The mean for BBA is 9.51, and the mean for ZTR is 8.64. If we do not care about this variation in our comparison, we can vertically shift the mean of both series to zero, i.e, subtract the mean of each series from everyday closing price. Now the Euclidean distance between the two sequences reduces to 12.78. The closing price of ZTR fluctuates in a smaller range than that of BBA; the standard deviation of ZTR is 0.10 while the standard deviation of BBA is 1.18. Again if we do not care about the scale in our comparison, we can scale both series, for example by the inverse of their standard deviation. The resulting series are often called the *normal forms* of the original series [GK95]. Figure 4 shows that the Euclidean distance between the normal forms of the two series is still 11.10; ZTR is more volatile than BBA. To remove this variation, we need to smooth out short term fluctuations, for example by computing the 20-day moving average of the two series. The Euclidean distance between the two series after applying the 20-day moving average drops to 2.75.

Example 2.2 Figure 5 shows in normal form the daily closing prices of stocks of *Pacific Gas and Electric Co.* (PCG) and *Plum Creek Timber Co.* (PCL) both starting from November 2nd, 1994 for 128 days. The Euclidean distance between the two normalized time series is 11.34. One way to compare the change rates of two stocks is to compare their “momenta”, which are obtained for every stock by subtracting the price at time t from the price at time $t+1$ (or, in general, $t+n$ for some n). The Euclidean distance between the two momenta is 13.01. The series representing the price of PCG has a spike on February 7th while the series of PCL has a spike on February 8th. If we horizontally shift the series of PCL one day to the left, then the spikes will overlap. The Euclidean distance between the two normalized time series after the horizontal shift drops to 8.91. The horizontal shifting reduces the distance between the momenta of the two series to 5.62.

The momentum of a time series describes the rate at which its value (such as the price in the preceding example) is rising or falling and it is seen as a measure of strength behind

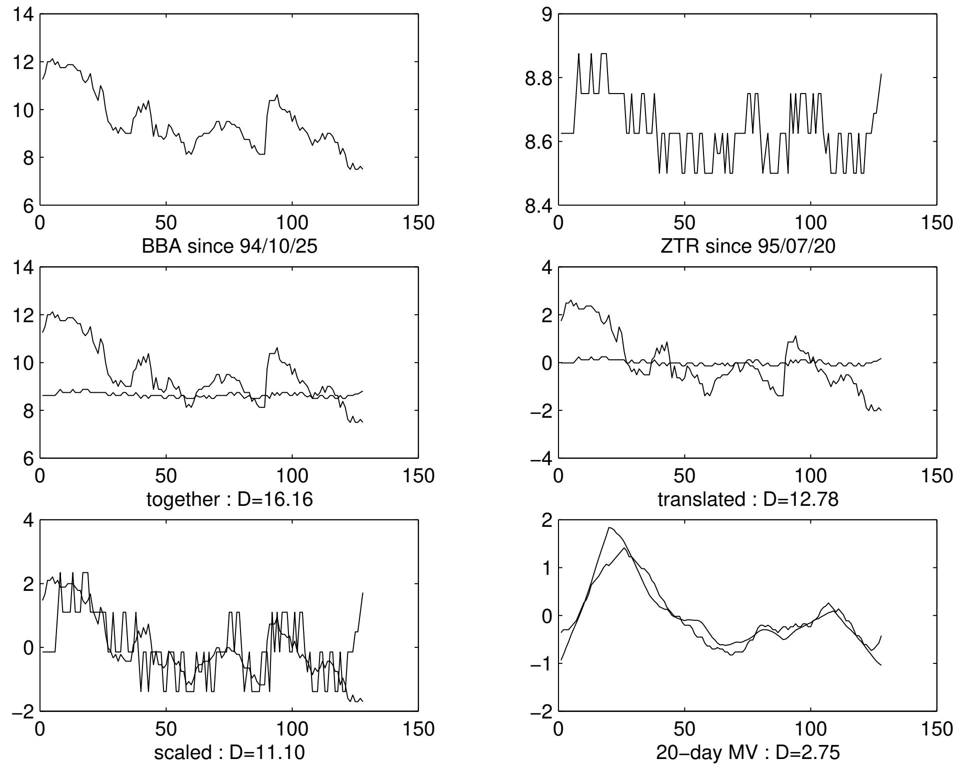


Figure 4: From left to right, top to bottom: the daily closing price of *The Bombay Co. (BBA)* starting from "94/10/25" for 128 days, the daily closing price of *Zweig Total Return Fund Inc. (ZTR)* starting from "95/07/20" for 128 days, the two stocks put together, both vertically translated, both vertically scaled, and both smoothed using the 20-day moving average (D denotes the Euclidean distance).

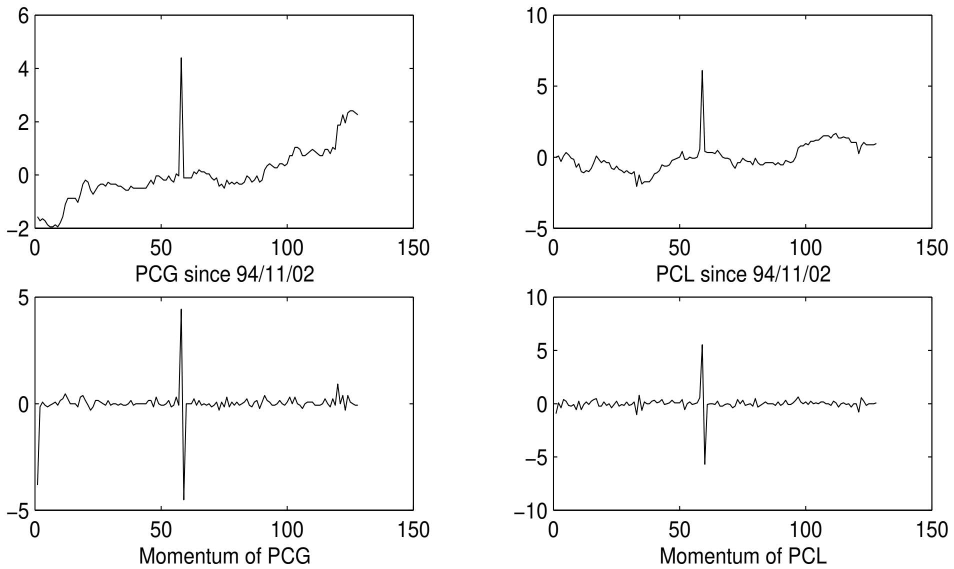


Figure 5: The daily closing price of *Pacific Gas and Electric Co. (PCG)* and that of *Plum Creek Timber Co. (PCL)*, both starting from 94/11/02 for 128 days, represented in normal forms and their momenta.

upward or downward movements. On the other hand, shifting a sequence horizontally before comparing it to another sequence removes any possible delay between the two sequences which can arise, for example in the stock market domain because of different reactions of two stocks to the same piece of news or recording errors.

As in the examples, we often specify a sequence of transformations to be applied to a time series. Given transformations $t_1 = (\vec{a_1}, \vec{b_1})$ and $t_2 = (\vec{a_2}, \vec{b_2})$, for example respectively corresponding to "*1-day shift*" and "*10-day moving average*", suppose we want to apply t_1 followed by t_2, which we denote by $t_2(t_1)$, to sequence $\vec{X}$. We can construct the new transformation as follows:

$$\begin{aligned} t_2(t_1(\vec{X})) &= \vec{a_2} * (\vec{a_1} * \vec{X} + \vec{b_1}) + \vec{b_2} \\ &= \vec{a_2} * \vec{a_1} * \vec{X} + \vec{a_2} * \vec{b_1} + \vec{b_2} \end{aligned} \tag{11}$$

Transformation $t_2(t_1)$ equivalently can be expressed as $t_3 = (\vec{a_3}, \vec{b_3})$ where $\vec{a_3} = \vec{a_2} * \vec{a_1}$ and $\vec{b_3} = \vec{a_2} * \vec{b_1} + \vec{b_2}$.

We now show how we can express some of these transformations in our framework. Suppose we want to express the 3-day moving average in our transformation language. Let us denote the Fourier transform of $\vec{s}$ by $\vec{S}$ and the Fourier transform of $\vec{m_3} = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ by $\vec{M_3}$. Now consider the transformation $t_{mavg3} = (\vec{M_3}, \vec{0})$ where $\vec{0}$ is a zero vector of the same size as $\vec{M_3}$. If we apply the transformation t_{mavg3} to $\vec{S}$, i.e.,

$$t_{mavg3}(\vec{S}) = \vec{S} * \vec{M_3} + \vec{0} = \vec{S} * \vec{M_3}$$

we get the 3-day moving average of $\vec{s}$ in the frequency domain. If we transform the right hand side back to the time domain using the convolution-multiplication relation (Equation 6), we get $conv(\vec{s}, \vec{m}3)$ which is the 3-day moving average of $\vec{s}$ in the time domain.

In general, the m-day moving average of a series of length n can be expressed by $t_{mavg} = (\vec{a}, \vec{0})$ where

$$\vec{a} = \underbrace{[\underbrace{w_1, w_2, \cdots, w_m}_{m}, 0, 0, \cdots, 0]}_{n} \tag{12}$$

and $\vec{0}$ is a zero vector of size n. Transformation t_{mavg} may be applied several times to get successive moving averages. The weights $w_1, \cdots, w_m$ are not necessarily equal. For trend prediction purposes, for example, the weights at the end are usually chosen to be higher than those at the beginning. Whereas for normal smoothing purposes, weights are equal, or those at the center are larger than those at the endpoints. Both momentum and horizontal shifting can also be formulated as affine transformations over the Fourier representation of a sequence. See Appendixes B and C for details.

Another transformation of special interest is normalization, which can be applied to time series before storing them in an index. Given a time series of mean μ and standard deviation σ, the normal form of the series is obtained by first vertically shifting the series by μ and then scaling it by $1/\sigma$. The equivalent transformation can be expressed in the frequency domain as $(\overrightarrow{1/\sigma}, [-\mu/\sigma, 0, 0, \ldots])$.

Although it is not required by the algorithms given in this paper, we assume time sequences are normalized and for every sequence, its normal form along with its mean and standard deviation are stored in a relation. This is mainly because of efficiency, as is noted by Goldin and Kanellakis [GK95], and the following two attractive properties of the normal form which are not mentioned by these authors.

1. It minimizes the Euclidean distance with respect to translation, i.e. $D(\vec{X} - s_x, \vec{Y} - s_y)$ has its minimum when s_x and s_y respectively are chosen to be the means of $\vec{x}$ and $\vec{y}$ [2].

2. The Euclidean distance between two normalized sequences is directly related to their cross-correlation [3]. The cross-correlation is a statistic measure to find out if two random variables (such as the crops harvested and the rain level) are linearly correlated. A value of cross-correlation near 0 often indicates that the variables are independent, while a value near 1 or -1 indicates a strong positive or negative correlation.

$$D^2(\vec{X}, \vec{Y}) = 2(n-1)(1 - \rho(\vec{X}, \vec{Y})) \tag{13}$$

   Equation 13 can be derived by expanding the Euclidean distance formula and replacing the mean and the standard deviation respectively by 0 and 1 in both the Euclidean distance and the cross-correlation formulas.

The second property can be quite useful in formulating similarity queries or translating one query to another. Since the Euclidean distance between two sequences can range from zero to infinity, it is usually difficult to specify a threshold for this distance. Instead, we can specify a threshold for cross-correlation which is between -1 and 1 and plug it into Equation 13 to find a threshold for the Euclidean distance. Using Equation 13, we can

[2] This can be verified by taking the first derivatives of D w.r.t. s_x and s_y and equating them to zero.

[3] $\rho(\vec{X}, \vec{Y}) = \frac{\mu_{\vec{X}*\vec{Y}} - \mu_{\vec{X}} \cdot \mu_{\vec{Y}}}{\sigma_{\vec{X}} \cdot \sigma_{\vec{Y}}}$

also translate any expression that uses the cross-correlation in a query to one that uses the Euclidean distance or vice versa.

Transformations can also be defined to stretch the time dimension (Example 1.2). Details are given elsewhere [RM97].

2.2 Queries with Multiple Transformations

It is often desirable to specify a set of transformations in a query. This is particularly useful if there are different ways of removing variations and one is not sure which one should be used. Given a distance function D and a set of transformations denoted by T, the semantics of $D(T(\vec{x}), \vec{y})$ for arbitrary data points $\vec{x}$ and $\vec{y}$ is defined as follows:

$$D(T(\vec{x}), \vec{y}) = min\{D(t(\vec{x}), \vec{y}) \mid t \in T\} \tag{14}$$

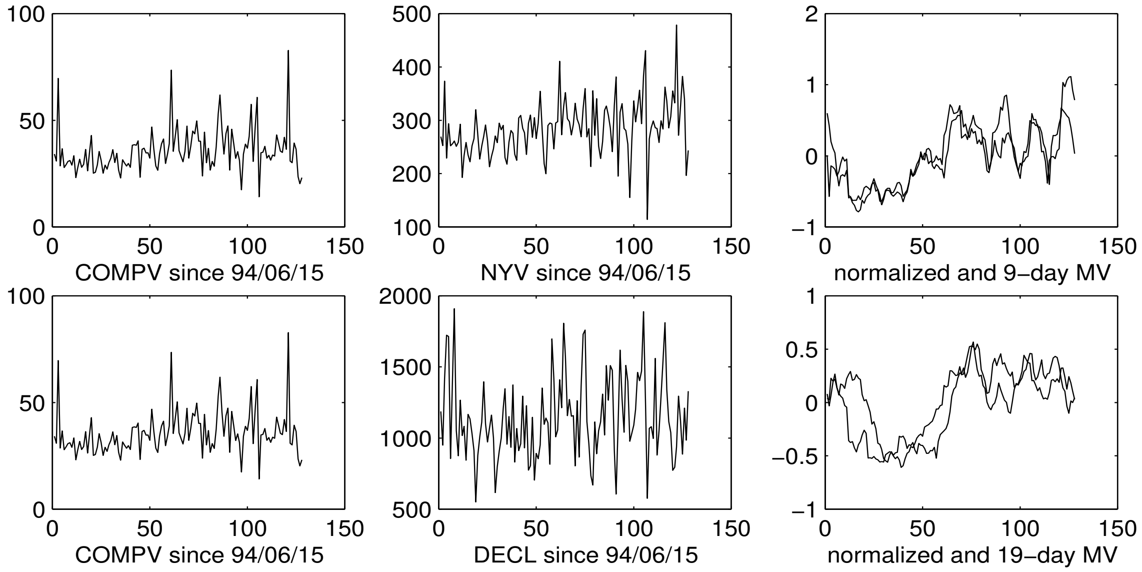


Figure 6: On the top from left to right, daily closings of *Dow Jones 65 Composite Volume (COMPV)* index, *NYSE Volume (NYV)* index and both put together, normalized and smoothed using 9-day moving average. On the bottom from left to right, again daily closings of COMPV index, *NYSE Declining Issues (DECL)* index and both put together, normalized and smoothed using 19-day moving average.

Example 2.3 Figure 6 shows daily closings of three indices: *Dow Jones 65 Composite Volume* (COMPV), *NYSE Volume* (NYV) and *NYSE Declining Issues* (DECL). It is difficult to see any similarity between these sequences. The Euclidean distance between closes of COMPV and NYV is 2873 and that between COMPV and DECL is 12939. On the other hand, if we normalize the closes of COMPV and NYV and compare their 9-day moving averages, they look very similar. The Euclidean distance between the 9-day moving averages of the normalized closes of COMPV and NYV is less than 3. Similarly, if we normalize the closes of COMPV and DECL and compare their 19-day moving averages, they also look quite similar. In fact, '19-day moving average' is the shortest moving average that reduces the Euclidean distance between normalized closes of COMPV and DECL to less than 3. To identify these similarities, a query may specify a set of moving averages to be applied to sequences. Although it may seem that if two sequences are similar w.r.t. n-day moving average, they should be similar w.r.t. $(n+1)$-day moving average, this is not true in general. For a counter example see Lemmas 6 and 7 in Appendix A.

Similar to the way we did for single transformations, we can compose two sets of transformations. Given two transformation sets T_1 and T_2, for example respectively corresponding to "*s-day shift*" for $s = 0, \ldots, 10$ and "*m-day moving average*" for $m = 1, \ldots, 40$, we can construct transformation set $T_3 = T_2(T_1)$, which corresponds to a "*s-day shift*" followed by an "*m-day moving average*" for all possible values of s and m, as follows:

$$T_3 = \{t_3 = t_2(t_1) \mid t_1 \in T_1, t_2 \in T_2\} \tag{15}$$

where $t_2(t_1)$ is defined by Equation 11. Using Equations 11 and 15, we can simplify a query by replacing any expression that uses a sequence of transformations with one that uses only a single or a set of transformations. We can process the resulting query using the techniques proposed in Section 3.2.

2.3 Safety of Transformations

Time series data can be easily stored in a multidimensional index and be efficiently retrieved using simple similarity queries. However, to the best of our knowledge, no prior work has been done on using these access methods for efficiently evaluating queries that use transformations in their expression of similarity. To achieve this goal, we define a notion of safety for transformations. We show in Section 3 that any query that only uses safe transformations in their expression of similarity can be efficiently supported using multidimensional indexes.

Definition 2 *An affine transformation is* safe *if it maps every j-axis-parallel line segment in (R^n, R^n), for $j = 1, \ldots, n$, to a j-axis-parallel line segment.*

Lemma 1 *An affine transformation t is safe iff it can be expressed as $t = (\vec{a}, \vec{b})$ where $\vec{a}, \vec{b} \in R^n$.*

Proof: For the clarity of the presentation, we give the proof in a 2-dimensional space. The extension of the result to an arbitrary n-dimensional space is straightforward.

(*if*) Suppose t can be expressed as $t = ([a_x, a_y], [b_x, b_y])$. Let $([x_1, y_1], [x_2, y_2])$ be an arbitrary axis-parallel line segment. If the line segment is parallel to the x axis, i.e. $y_1 = y_2$, then it will remain parallel to the x axis after the transformation because $a_y.y_1 + b_y = a_y.y_2 + b_y$. Similarly if the line segment is parallel to the y axis, then it will remain parallel to the y axis after the transformation. Thus t is safe.

(*only if*) Suppose t is safe. We show that t can be expressed as $([a_x, a_y], [b_x, b_y])$. Let $([x_1, y_1], [x_2, y_2])$ be an arbitrary axis-parallel line segment and $([x'_1, y'_1], [x'_2, y'_2])$ be its image under t. We can find a_x, a_y, b_x, b_y such that $x'_1 = a_x.x_1 + b_x$, $y'_1 = a_y.y_1 + b_y$, $x'_2 = a_x.x_2 + b_x$, $y'_2 = a_y.y_2 + b_y$. For instance, if the original line segment is parallel to the x axis, then $y_1 = y_2$, $y'_1 = y'_2$ and there will be three equations with four unknown variables to be solved. The rest is straightforward. ∎

Since we use the Fourier series representation of time series data as our features, and a Fourier coefficient, in general, is a complex number, we need to do a mapping before we can show the safety of transformations. If we decompose a complex number into its real and imaginary components, then we can represent a vector of k Fourier coefficients with a point in R^{2k}. We denote this mapping from the complex plane to the real plane by M_{rect}. Alternatively, we can decompose a complex number into its components in the polar coordinate system where a complex number is represented by a magnitude and a phase angle. We denote this mapping from the complex plane to the real plane by M_{pol}. We use $Re(x)$, $Im(x)$, $Abs(x)$, and $Angle(x)$ to denote respectively the real, the imaginary, the magnitude, and the phase angle of a complex number x. We now show that the transformations described for time series data in previous sections are safe.

Definition 3 *Let $\vec{a}$ and $\vec{b}$ be vectors of complex numbers and $\vec{a'}$ and $\vec{b'}$ be respectively their images under a mapping M from the complex plane to the real plane. Transformation $t = (\vec{a}, \vec{b})$ is safe w.r.t. M if $t' = (\vec{a'}, \vec{b'})$ is safe.*

Lemma 2 *Let $\vec{a}$ be a vector of real numbers, and $\vec{b}$ be a vector of complex numbers; the transformation $t = (\vec{a}, \vec{b})$ is safe w.r.t. M_{rect}.*

Proof: Without loss of generality, we assume the real and the imaginary components of the complex coordinate j (for $j = 1, \cdots, k$) are respectively mapped to the coordinates $2j-1$ and $2j$ of the real plane. Suppose $\vec{z}$ is a k-dimensional vector of complex numbers and $\vec{x}$, a $2k$-dimensional vector, is its image under M_{rect}. We have $z_j = x_{2j-1} + x_{2j}i$ for $j = 1, \cdots, k$. If we apply transformation t to $\vec{z}$, we get $\vec{z'} = t(\vec{z}) = \vec{a} * \vec{z} + \vec{b}$. We can rewrite this as follows:

$$\begin{aligned} z'_j &= a_j * (x_{2j-1} + x_{2j}i) + (Re(b_j) + Im(b_j)i) \\ &= (a_j * x_{2j-1} + Re(b_j)) + (a_j * x_{2j} + Im(b_j))i \end{aligned}$$

for $j = 1, \cdots, k$. If we map the resulting vector to a point x' using M_{rect}, we get $x'_{2j-1} = a_j * x_{2j-1} + Re(b_j)$ and $x'_{2j} = a_j * x_{2j} + Im(b_j)$ for $j = 1, \cdots, k$. This transformation can be rewritten as $t' = (\vec{c}, \vec{d})$ where $c_{2j-1} = c_{2j} = a_j$, $d_{2j-1} = Re(b_j)$, and $d_{2j} = Im(b_j)$ for $j = 1, \cdots, k$. Since $\vec{c}$ and $\vec{d}$ are vectors of real numbers, the rest follows from Lemma 1. ∎

On the other hand, Lemma 2 does not hold if $\vec{a}$ is chosen to be a vector of complex numbers. For example consider the axis-parallel line segment made by the two points $p = -5 + 2i$ and $q = -5 - 5i$. If we apply the transformation $t = ((2 - 3i), (0))$ to the line segment, i.e. we multiply the two endpoints by $2 - 3i$, the resulting line segment made by $t(p) = -4 + 19i$ and $t(q) = -25 + 5i$ is not axis-parallel anymore!

Lemma 3 *Let $\vec{a}$ be a vector of complex numbers, and $\vec{b}$ be a zero vector ($\vec{b} = \vec{0}$); the transformation $t = (\vec{a}, \vec{b})$ is safe w.r.t. M_{pol}.*

Proof: Without loss of generality, we assume the magnitude and the phase angle of the complex coordinate j (for $j = 1, \cdots, k$) are respectively mapped to the coordinates $2j - 1$ and $2j$ of the real plane. Suppose $\vec{z}$ is a k-dimensional vector of complex numbers and $\vec{x}$ is its image under M_{pol}. We have $z_j = x_{2j-1}e^{x_{2j}i}$ for $j = 1, \cdots, k$. If we apply transformation t to $\vec{z}$, we get $\vec{z'} = t(\vec{z}) = \vec{a} * \vec{z} + \vec{b}$. We can rewrite this as follows:

$$\begin{aligned} z'_j &= Abs(a_j)e^{Angle(a_j)i} * x_{2j-1}e^{x_{2j}i} + 0 \\ &= (Abs(a_j) * x_{2j-1})e^{(x_{2j}+Angle(a_j))i} \end{aligned}$$

for $j = 1, \cdots, k$. If we map the resulting vector to a point x' using M_{pol}, we get $x'_{2j-1} = Abs(a_j) * x_{2j-1}$ and $x'_{2j} = x_{2j} + Angle(a_j)$ for $j = 1, \cdots, k$. This transformation can be rewritten as $t' = (\vec{c}, \vec{d})$ where $c_{2j-1} = Abs(a_j)$, $d_{2j-1} = 0$, $c_{2j} = 1$, and $d_{2j} = Angle(a_j)$for $j = 1, \cdots, k$. Since $\vec{c}$ and $\vec{d}$ are vectors of real numbers, the rest follows from Lemma 1. ∎

In the next section, we develop efficient algorithms for evaluating queries that only use safe transformations in their expression of similarity.

3 Query Evaluation Using R-tree Indexes

Given a set of time series data, an index can be constructed as follows ([AFS93, RM98]): find the DFT of each sequence and keep the first few DFT coefficients as the sequence features. If we only keep the first k DFT coefficients, sequences will become points in a k-dimensional space. These points can be organized in a multidimensional index such as the R-tree family [Gut84, BKSS90, SK96], the k-d-B-tree [LS90] or the grid file [NHS84]. In this section, we describe the query evaluation for R-tree indexes. The same techniques can be extended to other index structures. We do that for the k-d-B-tree and the grid file in Section 4.

An R-tree can be seen as a natural extension of B-trees for more than one dimension. Non-leaf nodes contain entries of the form *(MBR, ptr)* where *MBR* is a *Minimum Bounding Rectangle* of all the entries in a descendent node, and *ptr* is a pointer to the descendent node. Bounding rectangles at the same tree level may overlap. Leaf nodes for point data sets contain a set of data points or pointers to full data records.

3.1 Evaluation of Queries with Single Transformations

Given an R-tree index I on a data set S, and any safe transformation t, we can construct an R-tree index I' for $t(S) = \{t(\vec{x}) \mid \vec{x} \in S\}$ as follows:

Algorithm 1 : For every node n

$$n = ((MBR_1,\ ptr_1), \cdots, (MBR_m,\ ptr_m))$$

in I, construct node n' in I' such that

$$\begin{aligned} n' &= t(n) \\ &= ((MBR'_1,\ ptr'_1), \cdots, (MBR_m,\ ptr'_m)) \end{aligned}$$

where MBR'_i is the rectangle obtained by applying t to the endpoints of MBR_i and ptr'_i is a pointer to the node $t(n_i)$ if n_i is the node pointed by ptr_i. If n is a leaf, ptr'_i ($= ptr_i$) is a pointer to a data tuple or null. Since MBR_i is an axis-parallel rectangle and t is a safe transformation, MBR'_i is an axis-parallel rectangle, due to the definition of a safe transformation. Thus the new node n' is a valid R-tree node. The construction stops when every node in I is mapped to a node in I'.

The original index I can be constructed in many different ways, each with a different performance, depending on the order of insertions and the way splits and overlaps are handled. Our experiments show that the new index I' has a similar performance to that of the original index. The main observation here is that for a given index I and transformation t, index I' can be built on the fly without having much impact on the performance of the search. This allows us to use one index for many possible transformations.

As our running example, consider the following proximity query:

> **Query 1:** Given a query point $\vec{q}$, a safe transformation t and a threshold ϵ, find all points $\vec{x}$ in the data set such that the Euclidean distance $D(t(\vec{x}), \vec{q}) < \epsilon$.

A naive evaluation of this query requires reading the whole data set, applying t to every data point and choosing every point $\vec{x}$ such that $D(t(\vec{x}), \vec{q}) < \epsilon$. This is a costly process. A better approach is to use Algorithm 1 to construct a new index for transformed data points. This new index can be built on the fly during the search operation as follows:

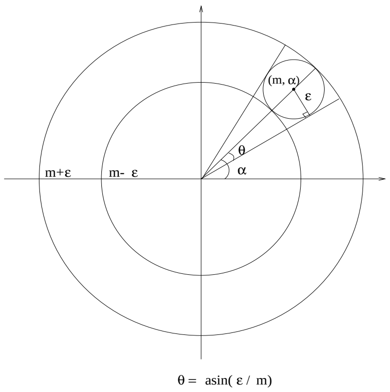


Figure 7: Minimum bounding rectangle in the polar coordinate system

Algorithm 2 : Suppose an R-tree index is constructed on the first k DFT coefficients of sequences, and suppose the root node is denoted by N.

1. **Preprocessing:**

   (a) Transform t and $\vec{q}$ into the frequency domain if they are in the time domain. Let us denote the first k Fourier coefficients of t and $\vec{q}$ by t_k and $\vec{q}_k$ respectively.

(b) Build a search rectangle q_{rect} for $\vec{q}_k$. A search rectangle is the minimum bounding rectangle that contains all points within the Euclidean distance ϵ of $\vec{q}$. Building a search rectangle is straightforward in the rectangular coordinate system; it is simply $(q_i - \epsilon, q_i + \epsilon)$ for $i = 1, \cdots, 2k$. The minimum bounding rectangle for a complex number $me^{\alpha j}$ in the polar coordinate system is demonstrated in Figure 7. The magnitude is in the range from $m - \epsilon$ to $m + \epsilon$, and the angle is in the range from $\alpha - arcsin(\frac{\epsilon}{m})$ to $\alpha + arcsin(\frac{\epsilon}{m})$.

2. **Search:**

   (a) If N is not a leaf, apply t to every (rectangle) entry of N and check if the resulting rectangle overlaps q_{rect}. For all overlapping entries, call **Search** on the index whose root node is pointed to by the overlapping entry.

   (b) If N is a leaf, apply t to every (point) entry of N and check if the resulting point overlaps q_{rect}. If so, the entry is a candidate.

3. **Postprocessing:**

   (a) For every candidate point $\vec{x}$, check its full database record to determine if the Euclidean distance between $t(\vec{x})$ and $\vec{q}$ is at most ϵ. If so, the entry is in the answer set.

The algorithm is guaranteed not to miss any qualifying sequence (see Appendix A for a proof). Similarly all-pairs queries and nearest neighbors queries can be efficiently processed using the index. For an all-pairs query, if we do a spatial join using the index, the only change in the spatial join algorithm will be to replace the join predicate evaluation with one that applies the desired transformation to the objects used in the join predicate. For example, the join predicate evaluation for $a_i \cap b_j \neq \emptyset$ can be easily changed to that for $t(a_i) \cap t(b_j) \neq \emptyset$ where t is a transformation and a_i and b_j are members of two spatial sets (such as two MBRs). For a nearest neighbors query, the search starts from the root and proceeds down the tree. As we go down the tree, we apply t to all entries of the node we visit. This change can be easily encoded in existing nearest neighbors search algorithms such as those proposed by Roussopoulos et al. [RKV95] and Seidl et al. [SK98].

3.2 Evaluation of Queries with Multiple Transformations

Given an R-tree index on a data set S, consider evaluating the following proximity query:

> **Query 2:** "Given a query time series $\vec{q}$ and a set T of safe transformations, find every time series $\vec{s} \in S$ and transformation $t \in T$ such that the Euclidean distance $D(t(\vec{s}), t(\vec{q})) < \epsilon$."

As a specific example, T could be the set of m-day moving averages for $m \in \{1 \ldots 40\}$ and we may want to find all stocks that have an m-day moving average similar to that of IBM. One way to process this query is to scan the data set sequentially and for every sequence

$\vec{s} \in S$ and transformation $t \in T$ find out if the distance predicate evaluates to true. The cost of this algorithm is one scan of the data set and computing the distance predicate $|T| * |S|$ times.

Another approach is for every transformation $t \in T$, use Algorithm 2 to retrieve all sequences within the proximity ϵ of the query sequence. The union of the results retrieved gives the query answer. We call this algorithm *ST-index*, where ST stands for 'a Single Transformation at a time'. The cost of this algorithm is $|T|$ times the cost of traversing the index.

The third approach is to group transformations together and scan the index once for every group. We call this algorithm *MT-index*, where MT stands for 'Multiple Transformations at a time'. There are two issues that need to be resolved here: first, how shall transformations be grouped together; second, how can a group of transformations be efficiently processed. We first address the second issue.

Since a transformation is a pair of n-dimensional vectors, it is a point in a $2n$-dimensional space. We assume both vectors are of real numbers. If not, one can rewrite any safe transformation in terms of a pair of real vectors (due to Lemma 1). Given a group of transformations, we can construct a *minimum bounding rectangle* (MBR) for all transformations in the group. Having an R-tree index built over sequences, we can apply the transformation rectangle to entries of the index and construct a new index on the fly. To apply a transformation rectangle to a data rectangle, we decompose the $2n$-dimensional transformation rectangle into two n-dimensional MBRs, one corresponding to $\vec{a}$ which we denote by *mult-MBR*, and the other corresponding to $\vec{b}$ which we denote by *add-MBR*. Given mult-MBR: $<(M_1l, M_1h), \ldots>$, add-MBR: $<(A_1l, A_1h), \ldots>$ and data rectangle X: $<(X_1l, X_1h), \ldots>$, the result of applying mult-MBR and add-MBR to rectangle X is rectangle Y: $<(Y_1l, Y_1h), \ldots>$ where

$$\begin{aligned} Y_il &= A_il + min(M_il * X_il,\ M_il * X_ih,\ M_ih * X_il,\ M_ih * X_ih) \\ Y_ih &= A_ih + max(M_il * X_il,\ M_il * X_ih,\ M_ih * X_il,\ M_ih * X_ih) \end{aligned}$$

for all dimensions i. As an example, consider the points of m-day moving average for $m = 1, \ldots, 40$ and their MBR. The result of applying the MBR of these transformations to a data rectangle is shown in Figure 8.

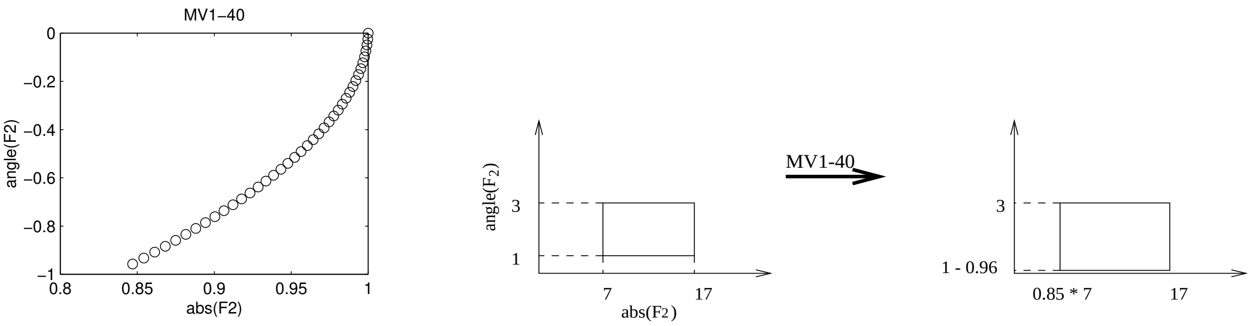


Figure 8: From left to right, the second DFT coefficient of m-day moving averages for $m = 1, \ldots, 40$ and a data rectangle before and after being transformed.

We now address the next issue, i.e. how shall transformations be grouped together. Consider the extreme case where all transformations in T are grouped together. If a few transformations spread all over the space, the minimum bounding rectangle of transformations will cover a large area. This MBR, when applied to a data rectangle, can easily make the data rectangle intersect the query region. This can reduce the filtering power of the index dramatically. On the other hand, if the number of groups and as a result the number of MBRs goes up, the area of each MBR gets smaller. Thus the filtering power of each MBR increases, but the same index needs to be traversed several times. In the worst case, the number of MBRs is the same as the number of transformations, i.e. every MBR includes only one transformation point. In such a case, both *ST-index* and *MT-index* perform exactly the same.

Now the question is how we should optimally choose MBRs for a given set of transformations such that the total cost (in terms of the number of disk accesses) becomes minimum. One solution is to estimate the cost for any possible set of MBRs and choose the set with minimum cost. A first attempt in estimating the cost for a given set of MBRs is to use the total area of MBRs. However, the total area is minimum if every MBR includes only one transformation point, i.e. the *ST-index* algorithm is used. Another approach for estimating the cost of a given set of MBRs is to apply MBRs for a fixed data rectangle, say a unit square, then compute the total area of the resulting data rectangles. Due to this estimation, the best performance should be obtained using only one transformation rectangle.

However, our experiments showed that using one transformation rectangle did not necessarily give the best performance. The worst performance for MT-index, which is close to that of ST-index, is when we pack two clusters of transformations into one rectangle. A solution to avoid this problem is to use a cluster detection algorithm (such as CURE [GRS98]) and avoid packing two clusters into one rectangle.

We can now write the search algorithm for Query 2 as follows:

Algorithm 3 : Given an R-tree index which is built on the first k Fourier coefficients of time series and whose root is N, a safe transformation set T, a threshold ϵ, and a search sequence $\vec{q}$, use the index to find all sequences that become within distance ϵ of $\vec{q}$ after being transformed by a member of T.

1. Decompose T into sets $T_1, T_2, \ldots$ using a clustering algorithm.

2. Do the following steps (steps 3 to 6) for every set T_i:

3. Build an MBR for points in T_i and project it into a mult-MBR and an add-MBR as described earlier.

4. If N is not a leaf, apply the mult-MBR and the add-MBR to every (rectangle) entry of N using Equation 16 and check if the resulting rectangle intersects q_{rect}. For every intersecting entry, go to step 4 and do this step on the index rooted at the node of the intersecting entry.

5. If N is a leaf, apply the mult-MBR and the add-MBR to every (point) entry of N and check if the resulting rectangle intersects q_{rect}. If so, the entry is a candidate.

6. For every candidate entry, retrieve its full database record, apply all transformations in T_i to the sequence, and determine transformations that reduce the Euclidean distance between the data sequence and the query sequence to less than ϵ.

This algorithm is guaranteed not to miss any qualifying sequence. The proof is similar to the one given in Appendix A for Algorithm 2. We can develop similar algorithms for efficiently processing spatial join and nearest neighbors queries. In a spatial join query, we apply the transformation MBR to all data items used in the join predicate before computing the predicate. Similarly in a nearest neighbors query, as we walk down the tree, we apply the transformation MBR to all entries of the node we visit. This change can be easily incorporated into the existing nearest neighbors search algorithms on the R-tree.

So far, we have made no assumption on any possible ordering among transformations. Next we show how one can exploit such an ordering during query evaluation.

3.3 Orderings of Transformations

In this section, we define a notion of ordering between transformations and show that this notion can be quite useful in guiding the search more effectively.

Definition 4 *Let T be a set of transformations on a vector space S (e.g., R^n) and D be a distance function on pairs of points in S; we call $< T, \prec >$ an ordering of T w.r.t. D if $\forall \vec{v_i}, \vec{v_j} \in S, \forall t_k, t_l \in T$,*

$$t_l \prec t_k \Rightarrow D(t_l(\vec{v_i}), t_l(\vec{v_j})) \leq D(t_k(\vec{v_i}), t_k(\vec{v_j}))$$

Once an ordering is established among transformations, we can use this ordering to guide the search more cleverly. To give an example, consider Query 2 with the transformation set $T = \{t_2, \ldots, t_{100}\}$ on R^n, where t_i means "scale by factor i". An ordering of T w.r.t. the *Euclidean distance* is: $t_l \prec t_k$ if $l < k$ (see Appendix A for a proof). To find all transformations that make a data sequence become similar to the query sequence we do not need to apply all transformations to sequences. Instead, we need to find the t_i with the largest i (i.e., the t_i that corresponds to the largest scale factor) that makes the distance predicate true. One way to find t_i is to do a binary search on the ordered list of transformations. Definition 4 implies that the distance predicate is true for every transformation t_j where $j < i$.

Given an ordered set T of transformations on the Fourier series representation of time series w.r.t. the Euclidean distance, we can use the binary search technique in all three algorithms presented in Section 3.2. In the case of the sequential scan method, we still need to scan the whole data set S. However, the number of transformations to be checked or the number of sequence comparisons drops to $|S| * log|T|$. The ordering assumption reduces the number of index traversals for ST-index to $log|T|$. In the case of MT-index, suppose T is decomposed into transformation groups $T_1, T_2, \ldots, T_m$. If the ordering is preserved among transformation groups, then the number of transformation groups to be checked and as a result the number of index traversals reduces to $log(m)$, whereas the number of transformations to be checked inside each group T_i (out of the $log(m)$ groups tested) reduces to $log|T_i|$. However, if the ordering is not preserved among transformation groups, then the number of index traversals will remain the same and the total number of comparisons reduces to $\sum_{i=1}^{m} log|T_i|$.

There are useful transformations on time series that are not ordered w.r.t. the Euclidean distance. For example, no ordering is possible for a set of moving averages on R^n w.r.t. the Euclidean distance. See Appendix A for a proof.

4 Query Evaluation Using Other Index Structures

The algorithms presented in Section 3 can be easily extended to other point-based access methods. To this end, we briefly examine two other index structures, namely the grid file and the k-d-B-tree. Compared to the R-tree which partitions the data points, these two access methods partition the embedding space that contains the data points. In this section, we show how these indexes can be used in efficiently evaluating our similarity queries.

4.1 Use of the Grid File

The grid file imposes an n-dimensional grid decomposition of space. A grid directory maps one or more cells of the grid into a data bucket. The grid, represented by n one-dimensional arrays called *scales*, is kept in main memory while the directory and data buckets are stored on disk.

Given a set of time series data, a grid file can be constructed as follows: find the DFT of each sequence and keep the first k DFT coefficients in a $2k$-dimensional grid file with data buckets keeping the full sequences. If a query requires a safe transformation $t = (\vec{a}, \vec{b})$ to be applied to the data set, a new grid file can be constructed on the fly by replacing every scale array $\vec{X_i} = (X_{i1},\ X_{i2},\ \ldots)$ along dimension i with scale array $\vec{X_i'} = (a_i.X_{i1} + b_i,\ a_i.X_{i2} + b_i,\ \ldots)$ and every data record $\vec{x}$ with $t(\vec{x})$. The result is still guaranteed to give a valid grid file, due to the safety of t.

Transformations may also be grouped together and simultaneously applied to the grid file. This can be done by: (1) building an MBR for each group of transformations, (2) applying each side of the MBR to the scale array associated with that side and (3) applying all transformations inside the MBR to the qualifying sequences.

4.2 Use of the K-D-B-Tree

The k-d-B-tree imposes an irregular (and not unique) decomposition of space. A leaf node contains a set of data records or pointers to data records. An interior node contains a set of pairs *(region, ptr)* with *region* corresponding to the disjoint union of regions represented by the node pointed to by *ptr* if *ptr* points to an interior node; otherwise it is a bounding rectangle for a set of points.

Given a k-d-B-tree on a set of time series data, and a query that requires a safe transformation t to be applied to data sequences, a new k-d-B-tree can be constructed on the fly by applying t to entries of the index (which are either rectangles or points) as the search proceeds. Due to the safety of t, the result of applying t to a k-d-B-tree is guaranteed to give a valid k-d-B-tree.

Similarly transformations may be grouped together and simultaneously applied to the k-d-B-tree. The algorithm is very similar to that of the R-tree.

5 Experimental Results

We implemented our methods on top of Norbert Beckmann's Version 2 implementation of the R*-tree [BKSS90]. We ran experiments on both real stock prices data obtained from the FTP site "ftp.ai.mit.edu/pub/stocks/results/" and synthetic data. The stock prices database consisted of 1068 stocks and for each stock its daily closing prices for 128 days. To test our algorithms on a larger data set and also to simulate the experiments reported in the literature for the purpose of comparison, we also did some experiments on synthetic data. Each synthetic sequence was in the form of $\vec{x} = [x_t]$ where $x_t = x_{t-1} + z_t$ and z_t was a randomly generated number in the range $[-500, 500]$.

For every time series, we first transformed it to the normal form for reasons described in Section 2.1, and then we found its Fourier coefficients. Since the mean of a normal form series is zero by definition, the first Fourier coefficient is always zero, so we can discard it. For every sequence, we stored the magnitudes and the angles of the second and the third DFT coefficients in the index.

All our experiments were conducted on a 168MHZ Ultrasparc station. For our experiments on proximity queries, we ran each experiment at least 100 times. Each time we chose a random query sequence from the data set and searched for all other sequences within distance ϵ of the query sequence. The execution time was averaged over these runs. Our all-pairs queries were spatial self-join queries where we searched the data set for all sequence pairs within distance ϵ of each other.

We report our experiments in two parts:

1. evaluating queries with single transformations (e.g. Query 1),
2. evaluating queries with multiple transformations (e.g. Query 2).

5.1 Evaluating Queries with Single Transformations

In this section, first we show that the overhead of applying single transformations to the index is negligible. We then compare our method to sequential scanning.

5.1.1 Overhead on the Index for Proximity Queries

Our first experiment was on synthetic data. Figure 9 compares the execution time for two kinds of queries: (a) a proximity query using transformations and (b) a proximity query that uses no transformations. We kept the sequence length fixed to 128 while we varied the number of sequences from 500 to 12,000. In order to have a precise comparison, the identity transformation $T_i = (\vec{I}, \vec{0})$ was chosen such that $T_i(\vec{X}) = \vec{X}$ for all sequences $\vec{X}$ ($\vec{I}$ is a vector of 1's and $\vec{0}$ is a vector of 0's). This made the two queries produce the same

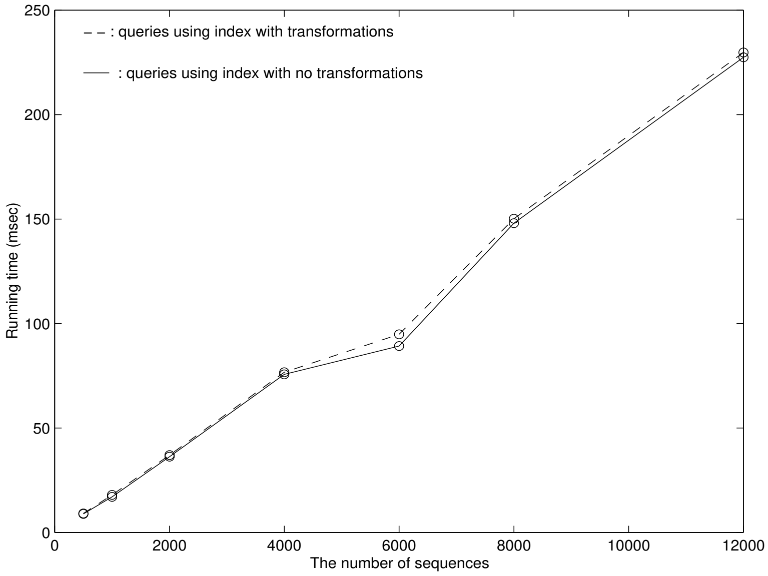


Figure 9: Time per query varying the number of sequences

results. Instead of setting the distance threshold ϵ directly in our proximity queries, we set the correlation to 0.85, plugged it in Equation 13 and found a value for ϵ. This threshold resulted in an average output size ranging from 1 to 69. As Figure 9 shows there is not much difference between the two curves. The minor difference is due to the CPU time spent for vector multiplication which is unavoidable. The number of disk accesses is the same in both cases. For a non-identity transformation, the CPU time spent for vector multiplication is still the same, but the number of disk accesses can be more or less depending on whether the transformation inflates or deflates the index rectangles.

In the next experiment, we kept the number of sequences fixed to 1000 while we varied the length of the sequences from 64 to 1024. The experiment was on proximity queries over synthetic data. The correlation threshold was again set to 0.85; this resulted in an average output size ranging from 1 to 6. We used the identity transformation again for the reason described in the previous experiment. To increase the accuracy of our measurements, we ran each experiment 1000 times. As demonstrated in Figure 10, the result was the same. Thus the index traversal for similarity queries does not deteriorate the performance of the index.

5.1.2 Comparison with Sequential Scanning for Proximity Queries

Figures 11 and 12 compare the execution time of our approach to sequential scanning for proximity queries. In Figure 11 the sequence length was fixed to 128 and the number of sequences varied from 500 to 12000. In Figure 12 the number of sequences was fixed to 1000 and the sequence length varied from 64 to 1024. The experiment was again on synthetic data. The correlation threshold was set to 0.85 and the identity transformation was also used. To have a good implementation of the sequential scanning algorithm, the transformation is applied to both data and query sequences during the distance computation; this means both sequences are scanned at most once when they in the main memory. We also stop

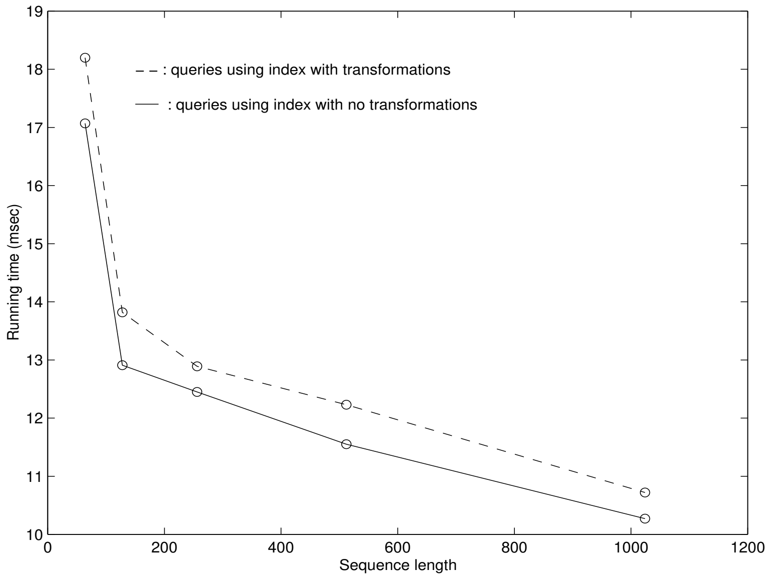


Figure 10: Time per query varying the sequence length

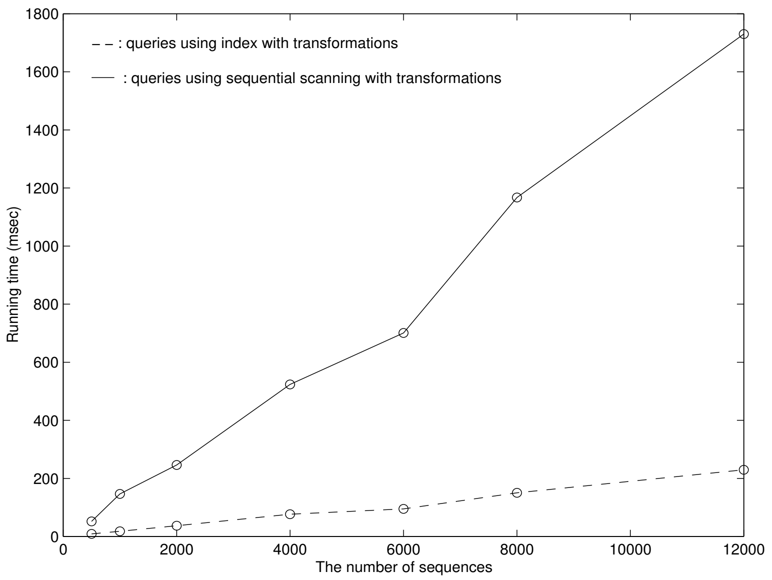


Figure 11: Time per query varying the number of sequences

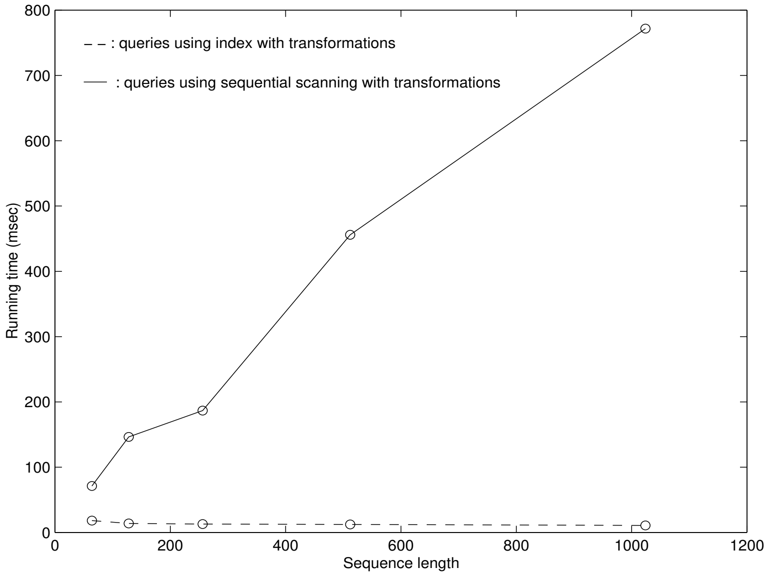


Figure 12: Time per query varying the sequence length

the distance computation process as soon as the distance exceeds ϵ. Both graphs show the superiority of our algorithm.

5.1.3 Comparison for All-Pairs Queries

Our last experiment in this part was the spatial self-join. We used the following methods:

a Scan the relation of time sequences sequentially, and compare every sequence $\vec{s}$ to all other sequences in the relation; the transformation t_{mavg20} is applied to every sequence during the comparison; The distance computation is stopped as soon as the distance exceeds ϵ.

b Scan the relation of Fourier coefficients sequentially, and for every time series build a search rectangle and pose it as a proximity query to the index after applying t_{mavg20} to both the index and the search rectangle.

c Do the spatial self-join as described in b without applying any transformation.

The experiment ran on a relation of stock prices data that had 1068 time sequences, and the length of each sequence was 128. The correlation threshold was set to 0.9895 and the distance threshold was computed accordingly. The result of the test is shown in Table 1. Method b compared to its competitor method a is 6 to 7 times faster. Methods b and c compute different queries, but the comparison between them confirms that the overhead on the index is very small.

5.2 Evaluating Queries with Multiple Transformations

In this section, we first compare the performance of MT-index to that of ST-index and sequential scan. To do the comparison, we made the choice of packing all transformations

algorithm	time (milisec)	size of the answer set
a	33.25	12
b	5.26	12
c	5.25	8

Table 1: The result of the join

into one rectangle though it did not necessarily give us the best possible performance of MT-index. In the subsequent section, we show the effect of grouping transformations on the performance of MT-index. In all our experiments over proximity queries, we set the correlation threshold fixed to 0.96 and used Equation 13 to find a value for the Euclidean distance threshold.

5.2.1 Comparing MT-index to ST-index and Sequential Scan

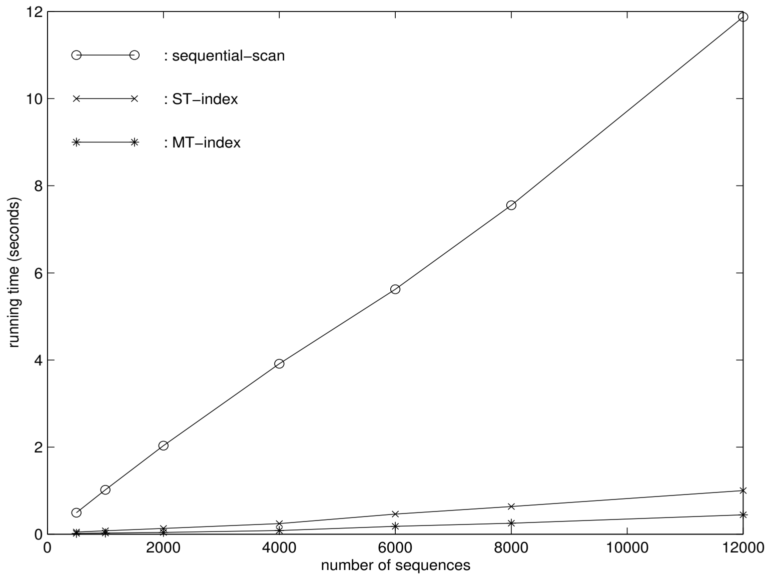


Figure 13: Time per query varying the number of sequences

Figure 13 shows the running time of Query 2 using three algorithms *sequential-scan*, *ST-index*, and *MT-index*. The transformations were a set of moving averages ranging from 10-day moving average to 25-day moving average with their number fixed to 16. The experiment ran on synthetic sequences of length 128 with their number varying from 500 to 12,000. The average output size was 7 or more depending on the number of input sequences. The figure shows that *MT-index* performs better than both *ST-index* and *sequential-scan*.

Figure 14 shows the running time of Query 2 again using three algorithms *sequential-scan*, *ST-index*, and *MT-index*. In the experiment, we set the number of sequences fixed to 1068, but we varied the number of transformations from 1 to 30. The transformations were a set of moving averages ranging from 5-day moving average to 34-day moving average.

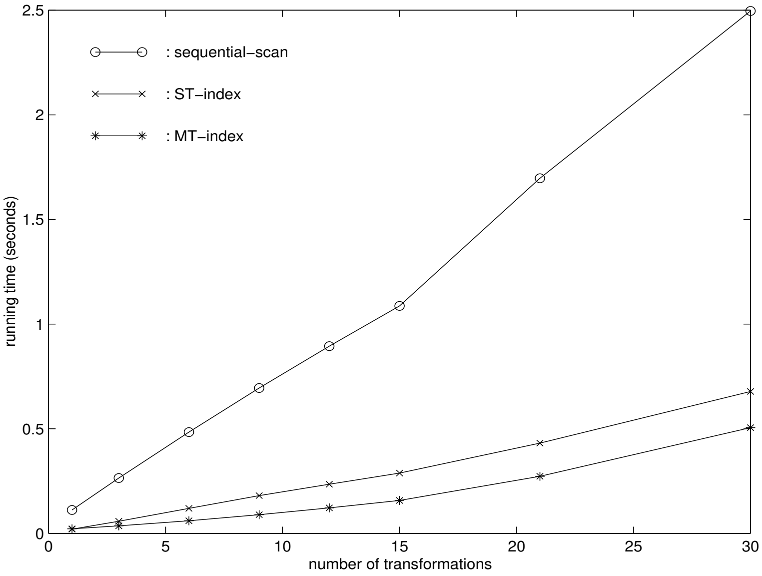


Figure 14: Time per query varying the number of transformations

The experiment ran on real stock price data. The average output size was 11 or more depending on the number of transformations. The figure shows that *MT-index* outperforms both *ST-index* and *sequential-scan*.

The next experiment was on a spatial join query which was expressed as follows:

> **Query 3:** "Given a set of transformations denoted by T, find every pair $\vec{s_1}$ and $\vec{s_2}$ of time series and every $t \in T$ such that the correlation $\rho(t(\vec{s_1}), t(\vec{s_2})) \geq 0.99$."

The transformations were again a set of moving averages ranging from 5-day moving average to 34-day moving average. We varied the number of transformations from 1 to 30. The experiment ran on real stock prices data which consisted of 1068 sequences of length 128. The average output size was at least 7. Figure 15 shows the running time of Query 3 using three algorithms: *sequential-scan, ST-index*, and *MT-index*. Both *ST-index* and *MT-index* perform better than *sequential-scan*. As we increase the number of transformations, the *MT-index* algorithm performs better than *ST-index* until the number of transformations gets 30. At this point the running time for both is the same.

5.2.2 Multiple Transformation Rectangles

In this section, we show that grouping all transformations in one rectangle does not necessarily give us the best possible performance. To show this, we ran Query 2 using the *MT-index* algorithm on real stock price data, but varied the number of transformations per MBR from one to its maximum. The transformation set consisted of m-day moving averages for $m = 6, \ldots, 29$. We equally partitioned subsequent transformations and built an MBR for each partition. As is shown in Figure 16, despite the fact that collecting all transformations in one rectangle resulted in the minimum number of disk accesses, it did not necessarily give us the best running time.

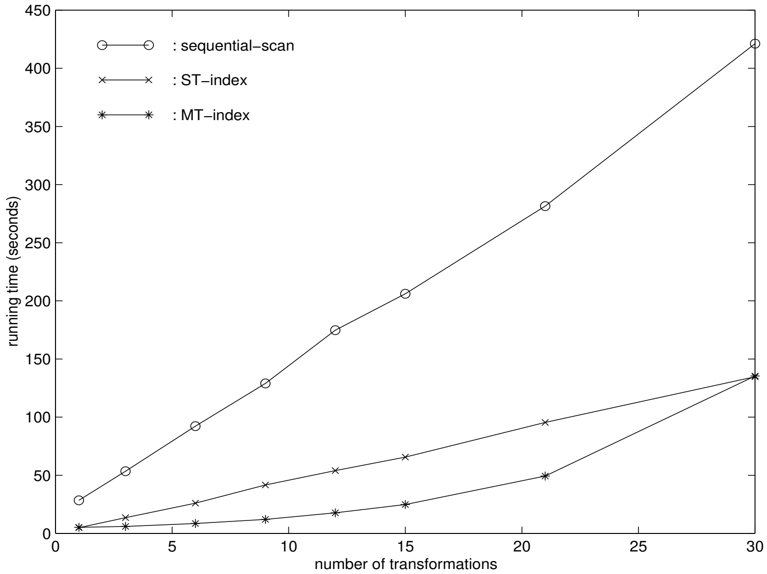


Figure 15: Time per query varying the number of transformations

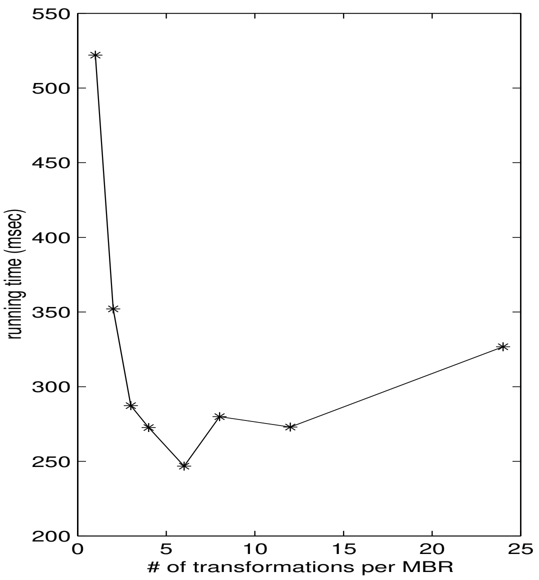
We later added the inverted version of each transformation, which was obtained by multiplying every coefficient by -1, to the transformation set. This created two clusters in a multidimensional space. Again, we equally partitioned subsequent transformations and built an MBR for each partition. We varied the number of transformations per MBR from one to 48 which was the size of the transformation set. As is shown in Figure 17, the running time shows bumps when we pack one third or all of the transformations in a rectangle. The same bumps are also observed in the number of disk accesses. This is due to the fact that in these two cases two separate clusters (or parts of them) are placed in one transformation rectangle.

These experiments show that as we start packing transformations into rectangles, we see a major performance improvement up to a certain point (six to eight transformations per rectangle here). The performance after this point either stays the same or goes down. The worst performance for MT-index, which was even worse than ST-index, was when we packed two clusters of transformations into one rectangle. A solution to avoid this problem is to use a cluster detection algorithm in advance and avoid packing more than one cluster to a rectangle.

6 Conclusions

We have proposed a class of transformations that can be used in a query language to express similarity between time series in a general way. This class allows the expression of several practically important notions of similarity, and queries using this class can be efficiently implemented on top of point-based multidimensional indexes. Our contributions can be summarized as follows:

- Formulation of an interesting class of transformations including the moving average in



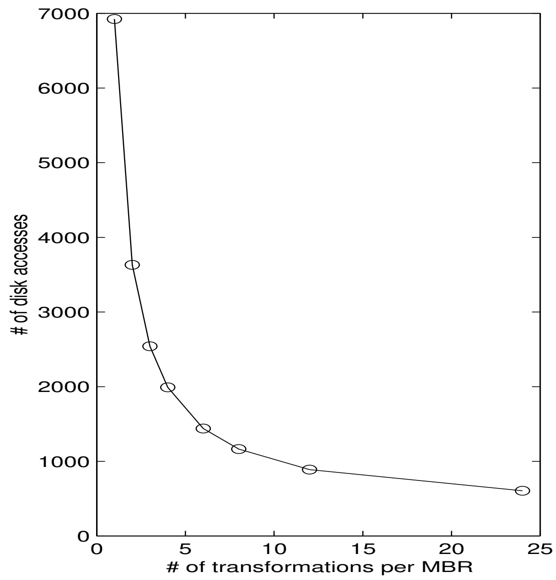


Figure 16: Both the running time and the number of disk accesses varying by the number of transformations per MBRs

our transformation language.

- Development of a safety condition for transformations.
- Implementation of similarity matching under these transformations on top of the R-tree index.
- Development of a new algorithm that applies multiple transformations specified in a query to a set of sequences in one scan of the R-tree index built on those sequences.
- Development of a notion of ordering among transformations and using this notion in efficiently guiding the search.

We evaluated our methods over both real stock prices and synthetic data. In the case of queries with single transformations, the experiments show that the execution time of our method is almost the same as that of accessing the index with no transformations; our method has much better performance than sequential scanning, and the performance gets better as both the number and the length of sequences increase. In the case of queries with multiple transformations, the experiments show that the given algorithm for handling multiple transformations outperforms both sequential scanning and the index traversal using one transformation at a time.

Acknowledgments

This research was supported by Communications and Information Technology Ontario and the Natural Sciences and Engineering Research Council.

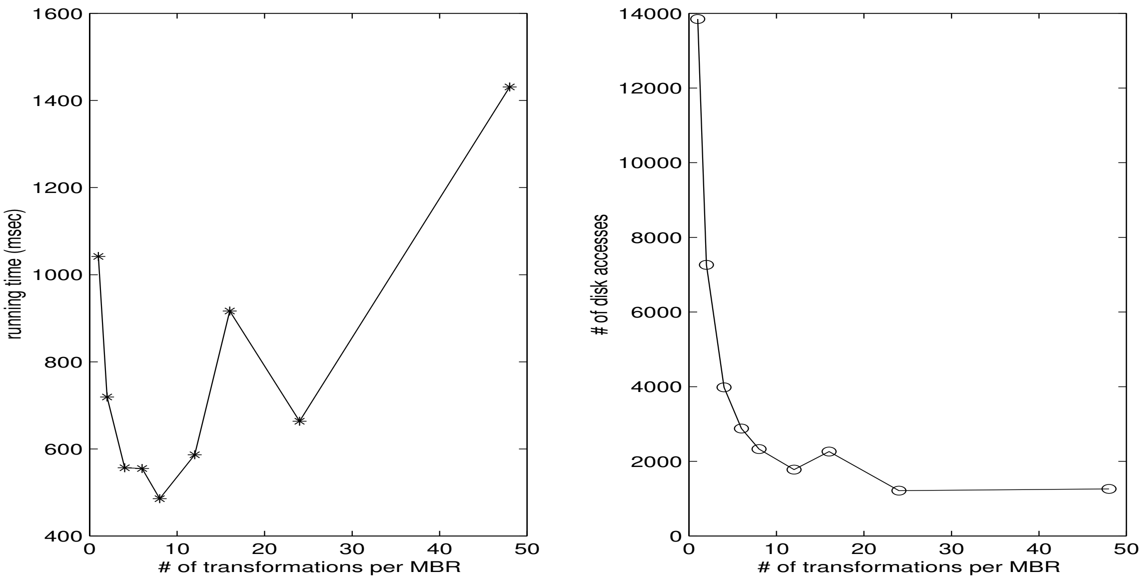


Figure 17: Both the running time and the number of disk accesses varying by the number of transformations per MBRs

7 Vitae

Davood Rafiei completed his undergrad in Computer Engineering at Sharif University of Technology, Tehran, in 1990, received his master in Computer Science from the University of Waterloo in 1995 and his Ph.D. in Computer Science from the University of Toronto in 1999. He was a post-doctoral fellow at the University of Toronto prior to joining to the Department of Computing Science at the University of Alberta as an Assistant Professor in 2000. Davood's research interests include database systems, querying and indexing non-traditional data such as time series and images and information retrieval on the Web.

Alberto Mendelzon did his undergraduate work at the University of Buenos Aires and obtained his Master's and Ph.D. degrees from Princeton University. He was a post-doctoral fellow at the IBM T.J. Watson Research Center and has been since 1980 at the University of Toronto, where he is now Professor of Computer Science and member of the Computer Systems Research Group. He has spent time as a visiting scientist at the NTT Basic Research Laboratories in Japan, the IASI in Rome, the IBM Toronto Lab, and AT&T Bell Labs Research in New Jersey. Alberto's research interests are in database systems, database query languages, Web-based information systems, and information integration.

References

[AFS93] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In *Proceedings of the 4th International Conference on Foundations of Data Organizations and Algorithms (FODO '93)*, pages 69–84, Chicago, October 1993.

[ALSS95] Rakesh Agrawal, King-Ip Lin, Harpreet S. Sawhney, and Kyuseok Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB '95)*, pages 490–501, Zurich, September 1995. Morgan Kaufmann Publishers.

[APWZ95] R. Agrawal, G. Psaila, E. L. Wimmers, and M. Zait. Querying shapes of histories. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB '95)*, pages 502–514, Zurich, September 1995.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R* tree: an efficient and robust index method for points and rectangles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '90)*, pages 322–331, Atlantic City, May 1990.

[CW99] K.K.W. Chu and M.H. Wong. Fast time series searching with scaling and shifting. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS '99)*, pages 237–248, Philadelphia, 1999.

[EM69] R. D. Edwards and J. Magee. *Technical analysis of stock trends.* John Magee, Springfield, Massachusetts, 1969.

[FJMM97] C. Faloutsos, H. V. Jagadish, A. O. Mendelzon, and T. Milo. A signature technique for similarity-based queries. In *Proc. Compression and Complexity of Sequences (SEQUENCES '97)*, Positano, June 1997.

[FRM94] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '94)*, pages 419–429, Minneapolis, May 1994.

[GK95] D. Q. Goldin and P. C. Kanellakis. On similarity queries for time-series data: constraint specification and implementation. In *1st Intl. Conf. on the Principles and Practice of Constraint Programming*, pages 137–153. LNCS 976, Sept. 1995.

[GRS98] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: an efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '98)*, pages 73–84, Seattle, June 1998.

[Gut84] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '84)*, pages 47–57, Boston, June 1984.

[JMM95] H. V. Jagadish, A. O. Mendelzon, and T. Milo. Similarity-based queries. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '95)*, pages 36–45, San Jose, May 1995.

[LS90] David B. Lomet and Betty Salzberg. The hb-tree: a multiattribute indexing method with good guaranteed performance. *ACM Transactions on Database Systems*, 15(4):625–658, December 1990.

[LYC96] C.S. Li, P.S. Yu, and V. Castelli. Hierarchyscan: a hierarchical similarity search algorith for databases of long sequences. In *Proceedings of the 12th Intl. Conf. on Data Engineering (ICDE '96)*, pages 546–553, New Orleans, February 1996.

[NC] NRC-CNRC. Feature selection bibliography. http://ai.iit.nrc.ca/bibliographies/feature-selection.html.

[NHS84] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: an adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, March 1984.

[OS75] A. V. Oppenheim and R. W. Schafer. *Digital Signal Processing*. Prentice-Hall, Englewood Cliffs, N.J., 1975.

[Raf98] Davood Rafiei. *Fourier-transform based techniques in efficient retrieval of similar time sequences*. PhD thesis, University of Toronto, 1998.

[RKV95] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '95)*, pages 71–79, San Jose, May 1995.

[RM97] Davood Rafiei and Alberto Mendelzon. Similarity-based queries for time series data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '97)*, pages 13–24, Tucson, Arizona, May 1997.

[RM98] Davood Rafiei and Alberto Mendelzon. Efficient retrieval of similar time sequences using DFT. In *Proceedings of the 5th International Conference on Foundations of Data Organizations and Algorithms (FODO '98)*, pages 249–257, Kobe, Japan, November 1998.

[Rot93] William G. Roth. MIMSY: A system for analyzing time series data in the stock market domain. University of Wisconsin, Madison, 1993. Master Thesis.

[RS92] Raghu Ramakrishnan and Divesh Srivastava. CORAL: Control, relations and logic. In *Proceedings of 18th International Conference on Very Large Data Bases (VLDB '92)*, pages 238–250, Vancouver, August 1992. Morgan Kaufmann.

[SK83] David Sankoff and Joseph B. Kruskal. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley Publishing Company, 1983.

[SK96] Kenneth C. Sevcik and Nikos Koudas. Filter trees for managing spatial data over a range of size granularities. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB '96)*, pages 16–27, Mumbai(Bombay), India, September 1996.

[SK98] Thomas Seidl and H.-P. Kriegel. Optimal multi-step nearest neighbour search. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '98)*, pages 154–165, Seattle, 1998.

[SLR94] P. Seshadri, M. Livny, and R. Ramakrishnan. Sequence query processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '94)*, pages 430–441, Minneapolis, May 1994.

[SZ96] H. Shatkay and S. Zdonik. Approximate queries and representations for large data sequences. In *Proceedings of the 12th International Conference on Data Engineering (ICDE '96)*, pages 536–545, New Orleans, March 1996.

[YJF98] Byoung-Kee Yi, H. V. Jagadish, and Christos Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings of the 14th International Conference on Data Engineering (ICDE '98)*, pages 201–208, Orlando, February 1998.

A Some Proofs

Lemma 4 *Algorithm 2 is guaranteed not to miss any qualifying sequence.*

Proof: Suppose $\vec{x}$ is a qualifying data sequence, i.e. $D(t(\vec{x}), \vec{q}) < \epsilon$, but $\vec{x}$ is not returned by Algorithm 2. If we represent t by $(\vec{a}, \vec{b})$, we can write

$$D(\vec{a} * \vec{x} + \vec{b}, \vec{q}) = (\sum_{f=0}^{n-1} |a_f x_f + b_f - q_f|^2)^{\frac{1}{2}} \leq \epsilon.$$

However,

$$(\sum_{f=0}^{k-1} |a_f x_f + b_f - q_f|^2)^{\frac{1}{2}} \leq (\sum_{f=0}^{n-1} |a_f x_f + b_f - q_f|^2)^{\frac{1}{2}} \leq \epsilon \tag{16}$$

for $k \leq n$. Thus sequence $\vec{x}$ must be returned by the proximity query on the index. This is a contradiction. ∎

Lemma 5 *Let $T = \{t_1, t_2, \ldots, t_m\}$ be a set of transformations on R^n where t_i means "scale by factor i"; an ordering of T w.r.t. the Euclidean distance is defined as $t_l \prec t_k$ if $l < k$.*

Proof: Let t_i and t_j be two arbitrary transformations in T and suppose, without loss of generality, $i < j$. Let $\vec{x}$ and $\vec{y}$ be two arbitrary points in R^n and $D(\vec{x}, \vec{y})$ denotes their Euclidean distance. Since $D(\vec{x}, \vec{y})$ is a positive number, we can multiply it to both sides of inequality $i < j$. This will give us

$$i.D(\vec{x}, \vec{y}) < j.D(\vec{x}, \vec{y}), \tag{17}$$

but we have

$$\begin{aligned} i.D(\vec{x}, \vec{y}) &= i.(\sum_{k=0}^{n-1} (x_k - y_k)^2)^{0.5} \\ &= (\sum_{k=0}^{n-1} (i.x_k - i.y_k)^2)^{0.5} = D(i.\vec{x}, i.\vec{y}). \end{aligned} \tag{18}$$

Equations 17,18 imply $D(i.\vec{x}, i.\vec{y}) < D(j.\vec{x}, j.\vec{y})$. ∎

Lemma 6 *Let T denotes a set of (circular) moving averages on R^n and D denotes the Euclidean distance; no ordering is possible for transformation set T w.r.t. D.*

Proof: We prove this lemma by contradiction. Suppose there is an ordering among members of T w.r.t. D. Consider the following sequences:

$\vec{s_1} =$ [10, 12, 10, 12]
$\vec{s_2} =$ [10, 11, 12, 11]
$\vec{s_3} =$ [11, 11, 11, 11]

If we denote the circular 2-day moving average by *mv2* and the circular 3-day moving average by *mv3*, we can write

$$\begin{array}{llllllllllll} mv2(\vec{s_1}) = & [\,11, & 11, & 11, & 11\,], & mv3(\vec{s_1}) = & [\,10.67, & 11.33, & 10.67, & 11.33\,], \\ mv2(\vec{s_2}) = & [\,10.5, & 10.5, & 11.5, & 11.5\,], & mv3(\vec{s_2}) = & [\,11, & 10.67, & 11, & 11.33\,], \\ mv2(\vec{s_3}) = & [\,11, & 11, & 11, & 11\,], & mv3(\vec{s_3}) = & [\,11, & 11, & 11, & 11\,]. \end{array}$$

There are two possible orderings between *mv2* and *mv3*:

- Case 1: $mv2 \prec mv3$
  By Definition 4, $D(mv2(\vec{s_i}), mv2(\vec{s_j})) \leq D(mv3(\vec{s_i}), mv3(\vec{s_j}))$ for all pairs $\vec{s_i}$ and $\vec{s_j}$. However, this does not hold for $\vec{s_2}$ and $\vec{s_3}$;

$$D(mv2(\vec{s_2}), mv2(\vec{s_3})) = 1 > D(mv3(\vec{s_2}), mv3(\vec{s_3})) = 0.75$$

- Case 2: $mv3 \prec mv2$
  By Definition 4, $D(mv3(\vec{s_i}), mv3(\vec{s_j})) \leq D(mv2(\vec{s_i}), mv2(\vec{s_j}))$ for all pairs $\vec{s_i}$ and $\vec{s_j}$. However, this does not hold for $\vec{s_1}$ and $\vec{s_3}$;

$$D(mv3(\vec{s_1}), mv3(\vec{s_3})) = 0.66 > D(mv2(\vec{s_1}), mv2(\vec{s_3})) = 0$$

There are no other cases, so the proof is complete. ∎

Lemma 7 *Let T denotes a set of non-circular moving averages on R^n and D denotes the Euclidean distance; no ordering is possible for transformation set T w.r.t. D.*

Proof: The proof is similar to that of Lemma 6. Suppose there is an ordering among members of T w.r.t. D. If we denote the non-circular 2-day moving averages by *mv2* and the non-circular 3-day moving averages by *mv3*, we can write

$$\begin{array}{llllllll} mv2(\vec{s_1}) = & [\,11, & 11, & 11\,], & mv3(\vec{s_1}) = & [\,10.67, & 11.33\,], \\ mv2(\vec{s_2}) = & [\,10.5, & 11.5, & 11.5\,], & mv3(\vec{s_2}) = & [\,11, & 11.33\,], \\ mv2(\vec{s_3}) = & [\,11, & 11, & 11\,], & mv3(\vec{s_3}) = & [\,11, & 11\,] \end{array}$$

where sequences $\vec{s_1}$, $\vec{s_2}$ and $\vec{s_3}$ are those given in Lemma 6. There are two possible orderings between *mv2* and *mv3*:

- Case 1: $mv2 \prec mv3$
  By Definition 4, $D(mv2(\vec{s_i}), mv2(\vec{s_j})) \leq D(mv3(\vec{s_i}), mv3(\vec{s_j}))$ for all pairs $\vec{s_i}$ and $\vec{s_j}$. However, this does not hold for $\vec{s_2}$ and $\vec{s_3}$;

$$D(mv2(\vec{s_2}), mv2(\vec{s_3})) = 0.87 > D(mv3(\vec{s_2}), mv3(\vec{s_3})) = 0.33$$

- Case 2: $mv3 \prec mv2$
  By Definition 4, $D(mv3(\vec{s_i}), mv3(\vec{s_j})) \leq D(mv2(\vec{s_i}), mv2(\vec{s_j}))$ for all pairs $\vec{s_i}$ and $\vec{s_j}$. However, this does not hold for $\vec{s_1}$ and $\vec{s_3}$;

$$D(mv3(\vec{s_1}), mv3(\vec{s_3})) = 0.47 > D(mv2(\vec{s_1}), mv2(\vec{s_3})) = 0$$

There are no other cases, so the proof is complete. ∎

B Expressing Momentum as a Transformation

Let $\vec{m} = [1,\ -1,\ 0,\ \ldots,\ 0]$ be a vector of length n and $\vec{x}$ be a time series of the same length. Let us denote the DFT of $\vec{m}$ by $\vec{M}$ and the DFT of $\vec{x}$ by $\vec{X}$. The convolution of $\vec{x}$ and $\vec{m}$, $conv(\vec{x}, \vec{m})$, gives the momentum of $\vec{x}$. Since a convolution in the time domain corresponds to a multiplication in the frequency domain, the product of $\vec{M}$ and $\vec{X}$ gives the momentum in the frequency domain. If we use the polar representation for complex numbers and map $\vec{X}$ and $\vec{M}$ respectively to real vectors $\vec{X}'$ and $\vec{M}'$ such that $M_j = M'_{2j}e^{iM'_{2j+1}}$ and $X_j = X'_{2j}e^{iX'_{2j+1}}$, we will have $M_j.X_j = (M'_{2j}.X'_{2j})e^{i(X'_{2j+1}+M'_{2j+1})}$ for $j = 0, \ldots, n-1$. Thus, we can express the momentum operation as an affine transformation of the form $(\vec{a}, \vec{b})$ where $a_{2j} = M'_{2j}$, $b_{2j} = 0$, $a_{2j+1} = 1$ and $b_{2j+1} = M'_{2j+1}$.

C Expressing Time Shift as a Transformation

Suppose we want to shift sequence $\vec{x} = [x_0, x_1, \ldots, x_{n-1}]$ one day to the right. If we inserted a zero at the beginning, the result after the shift would be $\vec{x}' = [0, x_0, x_1, \ldots, x_{n-1}]$ which is a sequence of length $n+1$. Using Equation 1, we can write the DFT of $\vec{x}'$ as follows:

$$X'_f = \frac{1}{\sqrt{n+1}} \sum_{t=0}^{n-1} x_t e^{\frac{-i2\pi(t+1)f}{n+1}} = e^{\frac{-i2\pi f}{n+1}} \left(\frac{1}{\sqrt{n+1}} \sum_{t=0}^{n-1} x_t e^{\frac{-i2\pi tf}{n+1}}\right)$$

where $f = 0, \ldots, n$. Time sequences are usually long, i.e. n is a large number, so we can easily replace $n+1$ inside the parentheses by n without much affecting the equation. Now the expression inside the parentheses becomes X_f, the fth DFT coefficient of $\vec{x}$, and we can write

$$X'_f \approx e^{\frac{-i2\pi f}{n+1}} X_f$$

This gives the first n Fourier coefficients of $\vec{x}'$. If we use the polar representation for complex numbers, we can express the shift operation as an affine transformation of the form $(\vec{1},\ [0,\ \frac{-2\pi(0)}{n+1},\ 0,\ \frac{-2\pi(1)}{n+1},\ \ldots])$. We can still do time shift even if $\vec{x}$ is not a long sequence. The trick is to pad at least as many zeros as the amount of the shift at the end of the sequence. Now we can forget the overflow zeros generated by the shift and consider the shifted sequence the same size as the original sequence.