On Similarity-Based Queries for Time Series Data

Davood Rafiei

Department of Computer Science, University of Toronto
E-mail: drafiei@db.toronto.edu

Abstract

We study similarity queries for time series data where similarity is defined in terms of a set of linear transformations on the Fourier series representation of a sequence. We have shown in an earlier work that this set of transformations is rich enough to formulate operations such as moving average and time scaling.

In this paper, we present a new algorithm for processing queries that define similarity in terms of multiple transformations instead of a single one. The idea is, instead of searching the index multiple times and each time applying a single transformation, to search the index only once and apply a collection of transformations simultaneously to the index. Our experimental results on both synthetic and real data show that the new algorithm for simultaneously processing multiple transformations is much faster than sequential scanning or index traversal using one transformation at a time. We also examine the possibility of composing transformations in a query or of rewriting a query expression such that the resulting query can be efficiently evaluated.

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 [3, 2]. 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.

A simple approach to determine a possible similarity between two time sequences is to compute the Euclidean distance (or any other distance, such as the city-block distance) between the two sequences, and call the two sequences similar if their distance is less than some user-defined threshold. However, there are many similarity queries that such a simple notion of similarity fails to capture; 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.

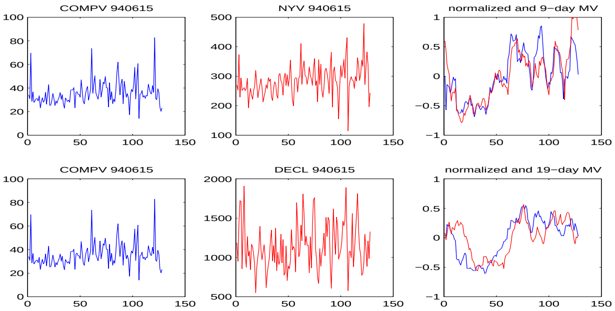

Figure 1. On the top from left to right, daily closings of *Dow Jones 65 Composite Volume (COMPV)* **index,** *NYSE Volume (NYV)* **index and both put together, normalized and smoothed using 9-day moving average. On the bottom from left to right, again daily closings of COMPV index,** *NYSE Declining Issues (DECL)* **index and both put together, normalized and smoothed using 19-day moving average.**

Example 1.1 Figure 1 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 [1] closes of COMPV and NYV

[1] This operation is described in Section 3.

and compare their 9-day moving averages, they look similar. The Euclidean distance between 9-day moving averages of 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 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.

Moving averages are widely used in stock data analysis (for example, see [5]). Their primary use is to smooth out short term fluctuations and depict the underlying trend of a stock. Given two sequences to be compared, we usually do not know what moving average can make them similar. There can be several moving averages that reduce the distance between two sequences to less than a threshold. We are often interested in the shortest moving average mainly because it leaves more details to the distance computation process. In addition, if two sequences happens to be similar w.r.t. n-day moving average, there is a high chance for them to be similar w.r.t. $(n+1)$-day moving average [2]. Moving averages can be formulated as linear transformations over the Fourier representation of a time sequence (see [13] for details).

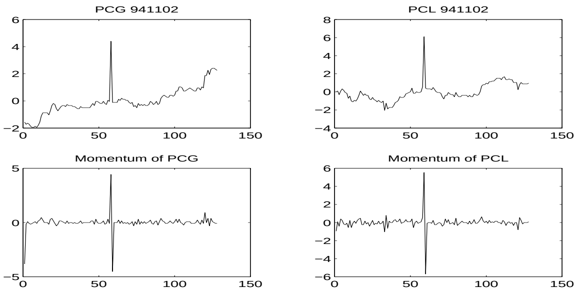

Figure 2. The daily closing price of *Pacific Gas and Electric Co. (PCG)* **and that of** *Plum Creek Timber Co. (PCL)***, both starting from 94/11/02 for 128 days, represented in normal forms and their momentums. Data was taken from the ftp site "ftp.ai.mit.edu/pub/stocks/results".**

Example 1.2 Figure 2 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. 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 3rd while the series of PCL has a spike on February 8th. No value is recorded for February 4th, 5th and 6th. If we shift the momentum of PCG two days to the right, the spikes will overlap and the Euclidean distance will reduce to 5.65.

The momentum of a sequence 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 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. Both momentum and shifting can be formulated as linear transformations over the Fourier representation of a sequence (see Section 3.1 for details). In general, there can be several possible linear transformations (or time shifts, as an example) to be applied to sequences and each transformation can either reduce or increase the distance between sequences. However, for every pair of sequences we are usually interested in finding transformations that reduce the distance between them to a minimum.

In this paper, we propose a fast algorithm to process queries that specify more than one transformation as the basis for similarity. The idea is, instead of processing a single transformation at a time, to process a collection of them at once. To achieve this goal, we construct a minimum bounding rectangle (MBR) for transformations. We show that the minimum bounding rectangle for transformations can be applied to a multidimensional index constructed on sequences, thus reducing the number of searches over the index to one. Our experiments show that this algorithm performs much better than both sequentially scanning all sequences and also the index traversal using one transformation at a time. We discuss the relationship between two similarity measures, the Euclidean distance and the cross-correlation, and show that queries expressed in terms of one over normalized sequences can be easily expressed in terms of the other. We also examine the possibility of composing transformations in a query or of rewriting a query expression such that the resulting query can be efficiently evaluated.

The organization of the rest of the paper is as follows. In the next section we review some background material, including past related work and the Discrete Fourier Transform (DFT). The benefits of using transformations for expressing similarity queries is discussed in Section 3. In Section 4 we propose algorithms for fast processing queries that express similarity in terms of multiple transformations. Section 5 contains experiments that show the effectiveness of our algorithms. Section 6 is the conclusion.

[2] This is not true in general; a counter example is given in the Appendix

2 Background

In this section, we discuss some background material, including past related work and the Discrete Fourier Transform (DFT).

2.1 Related Work

An indexing technique for the fast retrieval of similar time sequences is proposed by Agrawal et al. [1]. 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 a R-tree index. In an upcoming thesis [12], we discuss a major improvement of this indexing technique. We show that the last few Fourier coefficients of a sequence are as important as the first few coefficients due to the symmetry property of DFT. We also show that using the symmetry property improves the search time of the index by more than a factor of 2 without increasing its dimensionality.

In an earlier work ([13]), we use this indexing method and propose techniques for retrieving similar time sequences whose differences can be removed by a linear transformation such as moving average, time scaling and inverting. In this paper, we generalize our earlier work and allow queries that express similarity in terms of multiple transformations. Our work here can be seen as an efficient implementation of a special case of the query language described by Jagadish et al.[10] for time-series data.

There are other related works on time series data. An extension of the indexing technique of Agrawal et al. [1] for subsequence matching is proposed by Faloutsos et al. [7]. Goldin et al. [8] show that the similarity retrieval will be invariant to simple shifts and scales if sequences are normalized before being stored in the index. Yi et al. [20] use time warping as a distance function and present algorithms for retrieving similar time sequences under this function. Agrawal et al. [3] 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 [18]. A query language for time series data in the stock market domain is developed by Roth [15]. The language is built on top of CORAL [14], and every query is translated into a sequence of CORAL rules. Seshadri et al. [17] develop a data model and a query language for sequences in general but do not mention similarity matching as a query language operator.

2.2 Discrete Fourier Transform

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

$$X_f = \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t e^{\frac{-j2\pi tf}{n}} \quad f = 0, 1, \cdots, n-1 \tag{1}$$

where $j = \sqrt{-1}$ 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 energy of signal $\vec{x}$ is given by the expression

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

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

$$Conv(\vec{x}, \vec{y})_i = \sum_{k=0}^{n-1} x_k y_{i-k} \quad i = 0, 1, \cdots, n-1 \tag{3}$$

where $i - k$ is computed modulo n.

We use the following properties of DFT throughout this paper; they can be found in any signal processing textbook (for example, see [11]). The symbol $\Leftrightarrow$ denotes a DFT pair.

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

$$a\vec{x} + b\vec{y} \Leftrightarrow a\vec{X} + b\vec{Y} \tag{4}$$

for arbitrary constants a and b,

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

$$conv(\vec{x}, \vec{y}) \Leftrightarrow \vec{X} * \vec{Y} \tag{5}$$

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

$$|X_{n-f}| = |X_f| \;\; for \; f = 1, \ldots, n-1, \tag{6}$$

and

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

$$E(\vec{x}) = E(\vec{X}). \tag{7}$$

Using Parseval's relation, 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.

$$D^2(\vec{x}, \vec{y}) = E(\vec{x} - \vec{y}) = E(\vec{X} - \vec{Y}) = D^2(\vec{X}, \vec{Y}) \tag{8}$$

3 Similarity Queries and Transformations

A transformation 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 only those that can be removed using linear transformations on the Fourier series representation of the sequence. This class of transformations accounts for differences due to scaling, shifting, and the relative scaling (or weighting) and shifting between different coefficients.

3.1 Transformations - Examples

A transformation, denoted by $t = (\vec{a}, \vec{b})$, is a pair of real [3] vectors. The transformation t applied to a point $\vec{x}$ maps $\vec{x}$ to $\vec{a} * \vec{x} + \vec{b}$. To gain some insight into transformations, let us formalize the two operations, momentum and time shift, described in Example 1.2. More examples of transformations including those for moving averages can be found somewhere else [13, 12].

3.1.1 Momentum

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 $M_i = M'_{2i} e^{jM'_{2i+1}}$ and $X_i = X'_{2i} e^{jX'_{2i+1}}$, we will have $M_i.X_i = (M'_{2i}.X'_{2i}) e^{j(X'_{2i+1} + M'_{2i+1})}$ for $i = 0, \ldots, n-1$. Thus, we can express the momentum operation as a linear transformation of the form $(\vec{a}, \vec{b})$ where $a_{2i} = M'_{2i}$, $b_{2i} = 0$, $a_{2i+1} = 1$ and $b_{2i+1} = M'_{2i+1}$.

3.1.2 Time Shift

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:

$$
\begin{aligned}
X'_f &= \frac{1}{\sqrt{n+1}} \sum_{t=0}^{n-1} x_t e^{\frac{-j2\pi(t+1)f}{n+1}} \\
&= e^{\frac{-j2\pi f}{n+1}} \left(\frac{1}{\sqrt{n+1}} \sum_{t=0}^{n-1} x_t e^{\frac{-j2\pi t f}{n+1}} \right)
\end{aligned}
$$

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

$$
X'_f = e^{\frac{-j2\pi f}{n+1}} X_f
$$

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 the linear transformation $shift = (\vec{1}, [0, \frac{-2\pi(1)}{n+1}, 0, \frac{-2\pi(2)}{n+1}, \ldots])$. 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. Next we study one particular transformation which is quite useful in removing variations due to scalar shift and scale.

3.2 Transformations - Normal Form

An efficient way to compare two time sequences is to compare their normal forms. Given a time sequence $\vec{x}$ of mean μ and standard deviation σ, the transformation $(\overrightarrow{1/\sigma}, \overrightarrow{-\mu/\sigma})$ applied to $\vec{x}$ gives its normal form. Due to the linearity property of DFT, the same transformation is applicable to the Fourier representation of a sequence.

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 et al. [8]) and the following two attractive properties of the normal form sequences which are not mentioned by Goldin et al. [8].

1. It minimizes the Euclidean distance with respect to the scalar shift, 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 $\vec{y}$ [4].

2. The Euclidean distance between two normalized sequences is directly related to their cross-correlation [5].

$$
D^2(\vec{X}, \vec{Y}) = 2(n - 1 - n\rho(\vec{X}, \vec{Y})) \qquad (9)
$$

This 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.

[3] We have shown earlier ([13]) how a transformation described in terms of two complex vectors can be mapped into one expressed in terms of two real vectors under a safety constraint.

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

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

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 0 and 1 and plug it into Equation 9 to find a threshold for the Euclidean distance. Using Equation 9, we can also translate any expression that uses the cross-correlation in a query to one that uses the Euclidean distance or vice versa.

3.3 Composing Transformations in a Query

In a query, we may specify a sequence of transformations to be applied to a time sequence. For example, we may want to apply a *"s-day shift"* followed by an *"m-day moving average"*, for $s = 0, \ldots, 10$ and $m = 1, \ldots, 40$, to a sequence. We claim the queries expressed in terms of such a sequence of transformations also benefit from the algorithms given in this paper. We show this by giving a method to translate any query expression that uses a sequence of transformations into one that uses only a set of transformations. The resulting query can then be processed using the same technique that we present for multiple transformations.

Given transformations $t_1 = (\vec{a_1}, \vec{b_1})$ and $t_2 = (\vec{a_2}, \vec{b_2})$, for example respectively corresponding to "*2-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:

$$\begin{aligned} t_2(t_1(\vec{X})) &= \vec{a_2} * (\vec{a_1} * \vec{X} + \vec{b_1}) + \vec{b_2} \\ &= \vec{a_2} * \vec{a_1} * \vec{X} + \vec{a_2} * \vec{b_1} + \vec{b_2} \end{aligned} \tag{10}$$

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 can use this result to 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:

$$T_3 = \{t_3 = t_2(t_1) \mid t_1 \in T_1, t_2 \in T_2\} \tag{11}$$

where $t_2(t_1)$ is defined by Equation 10. Using Equations 10 and 11, 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 described in the next section.

4 Processing Similarity Queries

We consider spatial queries, namely range queries, spatial join queries and nearest neighbor queries and allow our transformations to be used in those queries. We discuss the issue of processing range queries in more detail and the two others very briefly. We start with the following range query:

> **Query 1:** "Given the closing price of a stock *q* and a set of transformations denoted by *T*, find every stock $s \in stocks$ and transformation $t \in T$ such that the Euclidean distance $D(t(\overrightarrow{s.close}), t(\overrightarrow{q.close})) < \epsilon$."

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. A solution for processing this query is to scan the whole *stocks* relation, compute the m-day moving average for the closing price of every stock and determine if the resulting sequence is within distance ϵ of the m-day moving average of the close of IBM. The distance predicate needs to be checked for all possible transformations. We refer to this algorithm as the *sequential-scan* method. The cost of this algorithm includes one scan of the whole relation and computing the distance predicate $|stocks| * |T|$ times.

Another approach is for every $t \in T$, apply *t* to the index built on the first few Fourier coefficients of the closing price and do a range query on the new index ([13]). The union of these results for all $t \in T$ 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 includes traversing the index $|T|$ times. Next, we describe a new algorithm that requires a single scan of the index and performs much better than both the *sequential-scan* and *ST-index* algorithms. We shall refer to this new algorithm by *MT-index*, where *MT* stands for 'Multiple Transformations at a time'.

4.1 MT-Index Algorithm for Multiple Transformations

A transformation t is of the form $t = (\vec{a}, \vec{b})$ where $\vec{a}$ and $\vec{b}$ are n-dimensional real vectors. Thus, a transformation can be represented as a point in a $2n$-dimensional space. Given a query that requires a set of transformations to be applied to a set of data sequences (or points), we first construct a *minimum bounding rectangle* (MBR) for all transformations. Having a multidimensional index for time sequences, we can apply the transformation rectangle to entries of the index. For a point data set, entries of a multidimensional index (such as R-tree) are usually in the form of points or rectangles. Since a point can be seen as a special kind of a rectangle with its lower bound equal to its upper bound in every dimension, we only consider applying a transformation rectangle to a data rectangle.

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 transformation rectangles mult-MBR: $< (M_1l, M_1h), (M_2l, M_2h), \ldots >$ and add-MBR: $< (A_1l, A_1h), (A_2l, A_2h), \ldots >$ and data rectangle X $:< (X_1l, X_1h), (X_2l, X_2h), \ldots >$, the result of applying mult-MBR and add-MBR to rectangle X is rectangle Y $:< (Y_1l, Y_1h), (Y_2l, Y_2h), \ldots >$ where

$$\begin{aligned} Y_iL = {} & A_il + \\ & min(M_il * X_il,\ M_il * X_ih,\ M_ih * X_il,\ M_ih * X_ih) \\ Y_ih = {} & A_ih + \\ & max(M_il * X_il,\ M_il * X_ih,\ M_ih * X_il,\ M_ih * X_ih) \end{aligned} \tag{12}$$

for all dimensions i. As an example, consider the points of m-day moving average for $m = 1, \ldots, 40$. Figure 3 shows the magnitudes and the angles of these points at the second DFT coefficient and their decompositions into mult-MBR and add-MBR. It can be observed that points inside mult-MBR make a horizontal line at 1. This is due to the fact that a data point angle is multiplied by 1. Similarly points inside add-MBR make a vertical line at 0 to show the fact that a data point magnitude is added by 0. The result of applying these MBRs to a data rectangle is shown in Figure 4.

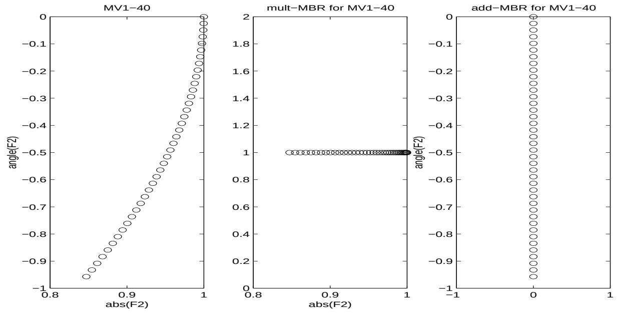

Figure 3. The second DFT coefficients of m-day moving averages (for $m = 1, \ldots, 40$) and their decompositions into mult-MBR and add-MBR

To develop an algorithm for answering Query 1, suppose an R-tree index is available on sequences. We can apply the transformation rectangle to every data rectangle in the index and construct a new index on the fly. The new index is constructed one index rectangle at a time, and each time the new rectangle is checked to see whether it intersects the query region. This process retrieves a set of candidate data items that includes all qualifying data items plus some false positives.

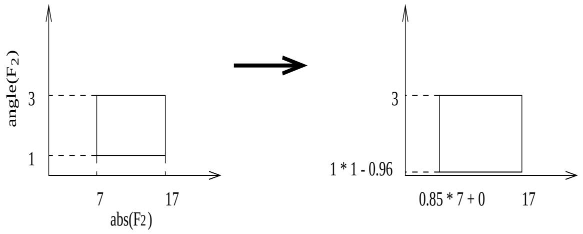

Figure 4. A data rectangle before and after being transformed

The last step of the algorithm removes false positives by applying every member of the transformation set to every candidate data item and selecting data items that intersect the query region. We can write the search algorithm more formally as follows:

Algorithm 1 : Given an R-tree index which is built on the first *k* Fourier coefficients of a sequence and whose root is N, a transformation set T, a threshold ϵ, and a search point $\vec{q}$, use the index to find all sequences that become within distance ϵ of $\vec{q}$ after being transformed by a member of T.

1. Build MBR r for points in T and project r into a mult-MBR and an add-MBR as described above.

2. Build a search rectangle of width ϵ around $\vec{q}$. We call this rectangle q_{rect}.

3. If N is not a leaf, apply the mult-MBR and the add-MBR to every (rectangle) entry of N using Equation 12 and check if the resulting rectangle intersects q_{rect}. For every intersecting entry, go to step 3 and do this step on the index rooted at the node of the intersecting entry.

4. 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.

5. For every candidate entry, retrieve its full database record, apply all transformations inside r 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 given in the next section). We can develop similar algorithms for efficiently processing spatial join and nearest neighbor queries. In a spatial join query, we apply the transformation MBR to all data items used in the join predicate before computing the predicate. For example, we may want to find all pairs of stocks that have similar closing prices with respect to an m-day moving average for some $m \in \{1, \ldots, 40\}$. Having an R-tree index for the closing prices, we can use any well-known spatial join algorithm for R-tree and change the join condition such that the transformation rectangle be applied to both data rectangles involved in the join before testing them for a possible overlap. Similarly in a nearest neighbor query, as we walk down the tree, we apply the transformation MBR to all entries of the node we visit. We can then use any kind of metric (such as MINDIST or MINMAXDIST discussed in [16]) to prune the search.

4.2 Correctness and Analysis of the MT-Index Algorithm

To prove the correctness of Algorithm 1, we have to show that no qualifying sequence is being rejected.

Lemma 1 *Algorithm 1 misses no tuple from the answer set of Query 1.*

proof: We prove this lemma by contradiction. Suppose $\vec{x}$ is a sequence in the answer set of Query 1 but it is not returned by Algorithm 1. Let $t = (\vec{a}, \vec{b})$ be a transformation which makes $\vec{x}$ similar to the query sequence. Let the three rectangles $<(X_1l, X_1h), (X_2l, X_2h), \ldots>$, $<(M_1l, M_1h), (M_2l, M_2h), \ldots>$, $<(A_1l, A_1h), (A_2l, A_2h), \ldots>$ be MBRs that respectively include $\vec{x}$, $\vec{a}$ and $\vec{b}$. For an arbitrary dimension i, we can write the following equations:

$$X_il \leq x_i \leq X_ih \tag{13}$$
$$M_il \leq a_i \leq M_ih \tag{14}$$
$$A_il \leq b_i \leq A_ih \tag{15}$$

If we multiply all sides of Equations 13, 14, 14 respectively by a_i, X_il and X_ih for positive values of a_i, X_il and X_ih, we will get

$$\begin{aligned} a_i.X_il &\leq a_i.x_i \leq a_i.X_ih \\ M_il.X_il &\leq a_i.X_il \leq M_ih.X_il \\ M_il.X_ih &\leq a_i.X_ih \leq M_ih.X_ih \end{aligned}$$

These three equations imply $M_il.X_il \leq a_i.x_i \leq M_ih.X_ih$. If we do the same step for all positive and negative combinations of a_i, X_il and X_ih, we will get different equations which all satisfy the following inequality:

$$\begin{aligned} &min(M_il.X_il, M_il.X_ih, M_ih.X_il, M_ih.X_ih) \leq \\ &a_i.x_i \leq max(M_il.X_il, M_il.X_ih, M_ih.X_il, M_ih.X_ih) \end{aligned} \tag{16}$$

We can add the three sides of Equations 15,16 and get

$$\begin{aligned} &min(M_il.X_il, M_il.X_ih, M_ih.X_il, M_ih.X_ih) + A_il \\ &\leq a_i.x_i + b_i \leq \\ &max(M_il.X_il, M_il.X_ih, M_ih.X_il, M_ih.X_ih) + A_ih \end{aligned} \tag{17}$$

Since i is chosen to be an arbitrary dimension, this inequality holds for all dimensions. Thus $\vec{a}.\vec{x} + \vec{b}$ is a point inside the rectangle defined by Equation 17 which is exactly the same as the one used in Algorithm 1 (Equation 12). For the same reason, sequence $\vec{x}$ will be returned by Algorithm 1. This is a contradiction, so the proof is complete. ∎

Symbols	*Definitions*
C	cost using one transformation rectangle
C_k	cost using k transformation rectangles
$DA_{all}(q, r)$	number of disk accesses for query q and transformation rectangle r in all index levels
$DA_{leaf}(q, r)$	number of disk accesses for query q and transformation rectangle r in the leaf level
CA_{leaf}	average capacity of a leaf node
$NT(r)$	number of transformations inside rectangle r
C_{DA}	cost of a disk access
C_{cmp}	cost of a sequence comparison

There are two determining factors in the cost of Algorithm 1: one is the number of disk accesses required to find and retrieve all candidate data items ($DA_{all}(q, r)$) in steps 3 and 4, and the other is the number of comparisons performed between the full database records of candidate sequences and the query sequence in Step 5. The latter is equal to the product of the number of candidate data items and the number of transformation points inside r which is denoted by $NT(r)$. The CPU cost of applying the transformation rectangle to a data rectangle in step 3 is negligible because these rectangles have no more than a few dimensions; however, sequences involved in the comparison process of step 5 are usually long and the CPU cost is not negligible. If we assume the number of disk accesses at the leaf to be $DA_{leaf}(q, r)$ and the average capacity of a leaf node to be CA_{leaf}, a good estimate of the number of candidate data items is $DA_{leaf}(q, r).CA_{leaf}$. Thus we can write the cost function as follows:

$$\begin{aligned} C = \; & DA_{all}(q, r).C_{DA} + \\ & DA_{leaf}(q, r).CA_{leaf}.NT(r).C_{cmp} \end{aligned} \tag{18}$$

where C_{DA} is the cost of a disk access and C_{cmp} is the cost of a comparison.

Next, we describe a possible technique for improving the performance of Algorithm 1.

4.3 Performance Improvement

A problem with the *MT-index* algorithm is if transformations make several clusters or a few of them spread all over the space, then 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. A solution for this problem is to allow more than one transformation rectangle. As the number of MBRs goes up, the area of each MBR gets smaller, and as a result the filtering power of the MBR increases; but, on the other hand, 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. Given k transformation rectangles $r_1, \ldots, r_k$, we can generalize the cost function in Eq. 18 and express the new cost function as follows:

$$\begin{aligned} C_k &= \sum_{i=1}^{k}(DA_{all}(q, r_i).C_{DA} + \\ &\quad DA_{leaf}(q, r_i).CA_{leaf}.NT(r_i).C_{cmp}) \end{aligned} \tag{19}$$

or equivalently

$$\begin{aligned} C_k &= C_{DA}.\sum_{i=1}^{k} DA_{all}(q, r_i) + \\ &\quad CA_{leaf}.C_{cmp}.\sum_{i=1}^{k} DA_{leaf}(q, r_i).NT(r_i) \end{aligned} \tag{20}$$

Now the question is how we should optimally choose MBRs for a given set of transformations such that the cost C_k becomes minimum. A solution is to estimate the cost for any possible set of MBRs and choose the set that gives the minimum cost. To estimate the cost, we need to estimate $DA_{all}(q, r_i)$ and $DA_{leaf}(q, r_i)$. A first attempt in estimating these two parameters is to assume the number of disk accesses only depends on the area of r_i. However, the cost C_k is then minimum if every MBR includes only one transformation point, i.e. the *ST-index* algorithm is used. There are also techniques (for example, see [6, 19]) to estimate $DA_{all}(q, r_i)$ and $DA_{leaf}(q, r_i)$ for a given R-tree, but none of these techniques take the actual size and distribution of data or directory rectangles into account. Therefore, $DA_{all}(q, r_i)$ and similarly $DA_{leaf}(q, r_i)$ would be independent from r_i and we can take these two terms out of the summations in Eq. 20. As a result of this, if we increase the number of transformation rectangles, the first term in the cost function (Eq. 20) would increase linearly, but the second term would remain unchanged. 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. Thus, we cannot assume the number of disk accesses, $DA_{all}(q, r_i)$ or $DA_{leaf}(q, r_i)$, to be independent from the size and the location of r_i. Intuitively, as we increase the size of the transformation rectangle, it is more likely to make an arbitrary chosen data or directory rectangle intersect the query region. In addition, even if only one (out of $NT(r_i)$) transformation makes a data rectangle intersect the query region, we still need to search all $NT(r_i)$ transformations in the postprocessing step to find those that make a candidate sequence becomes similar to a query sequence, and this needs to be done for all candidates. Thus the trade off is whether we scan the index once and do more postprocessing work with possibly a larger set of candidates or scan the index several times and do less postprocessing work with possibly a smaller set of candidates. Our experiments in the next section over a set of moving averages show that packing six to eight transformations in a rectangle make a major improvement over ST-index, but there is not much improvement if we pack more transformations in a rectangle. 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 [9]) and avoid packing two clusters into one rectangle.

4.4 Ordering Assumption on Transformations

So far, we have made no assumption on any possible ordering among transformations. In this section, we define a notion of ordering among transformations and show that it can be quite useful in guiding the search process more effectively.

Definition 1 *We call* $< T, \prec>$ *an ordering of* $T = \{t_1, t_2, \ldots, t_n\}$ *w.r.t. value domain* `dom` *and distance function* D *if* $\forall v_i, v_j \in \texttt{dom}, \forall t_k, t_l \in T$,

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

Once an ordering is established among transformations, we can use this ordering to guide the search more cleverly. To give an example, consider Query 1 and assume $T = \{2, \ldots, 100\}$ represents a set of scaling factors. It is easy to show that “less than” defines an ordering among members of *T* w.r.t. the domain of *time sequences* and the *Euclidean distance* (see Appendix A for a proof). To find all transformations that make a data sequence to become similar to a query sequence we do not need to apply all scale factors to sequences. Instead, we need to find the largest scale factor that makes the distance predicate true. Suppose s_i is such a scale factor. One way to find s_i is to do a binary search on the set of scale factors. Definition 1 easily implies that the distance predicate is true for all scale factors less than s_i.

We can use the binary search technique in all three algorithms described earlier. In the case of the sequential scan method, we still need to scan the whole stocks relation. However, the number of sequence comparisons drops to $|stocks| * log|T|$. Similarly in the case of the MT-index algorithm, the number of disk accesses still will be the same, but the number of comparisons for every candidate sequence drops to $log|T|$. The ordering assumption reduces the number of index traversals for ST-index to $log|T|$.

On the other hand, the ordering assumption does not hold in general. There are useful transformations that are not ordered w.r.t. time sequences and the Euclidean distance. For example, we can show that no ordering is possible for a set of moving averages w.r.t. time sequences and the Euclidean distance (see Appendix A for a proof).

5 Experimental Results

We implemented both *ST-index* and *MT-index*, on top of Norbert Beckmann's Version 2 implementation of the R*-tree [4]. We ran experiments on both stock prices data obtained from the ftp site "ftp.ai.mit.edu/pub/stocks/results" and synthetic data. All our experiments were conducted on a 168MHZ Ultrasparc station. The stock prices database consisted of 1068 stocks and for each stock its daily closing prices for 128 days. Each synthetic sequence was in the form of $\vec{x} = [x_t]$ where $x_t = x_{t-1} + z_t$ and z_t is a uniformly distributed random number in the range $[-500, 500]$.

For every time series, we first transformed it to the normal form for reasons described in Section 3.2, 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 throw it away. We mapped the mean and the standard deviation of the original time series respectively to the first and the second dimensions of the index. We also mapped the magnitude and the phase angle of the second DFT term (computed for the normal form series) respectively to the third and the fourth dimensions of the index, and the magnitude and the phase angle of the third DFT term respectively to the fifth and the sixth dimensions. We used the symmetry property of DFT in all our experiments over the index.

We report our experiments in two parts. In the first part, we compare MT-index to ST-index and sequential scan. In this part, we made the choice of packing all transformations into one rectangle though it did not necessarily give us the best possible performance of MT-index. In the second part, we varied the number of transformation rectangles from one to its maximum to see the effect of having multiple transformation rectangles on the performance of MT-index. In all our experiments over range queries, we ran each experiment 100 times and 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. We averaged the execution times from these runnings. We also set the correlation threshold fixed to 0.96 for all range queries. We plugged this threshold in Equation 9 to find a value for the Euclidean distance threshold.

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

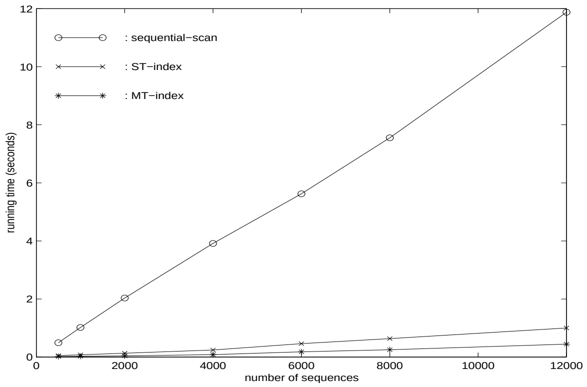

Figure 5. time per query varying the number of sequences

Figure 5 shows the running time of Query 1 using three algorithms *sequential-scan*, *ST-index*, and *MT-index*. In the experiment, we set the number of transformations fixed to 16, but we varied the number of sequences from 500 to 12,000. The experiment ran on synthetic sequences of length 128. The transformations were a set of moving averages ranging from 10-day moving average to 25-day moving average. 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*.

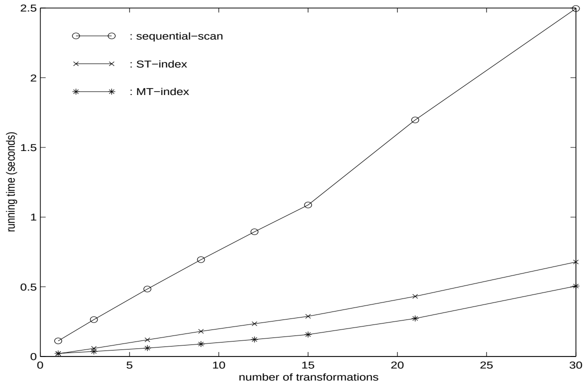

Figure 6. time per query varying the number of transformations

Figure 6 shows the running time of Query 1 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. The experiment ran on real stock prices 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*.

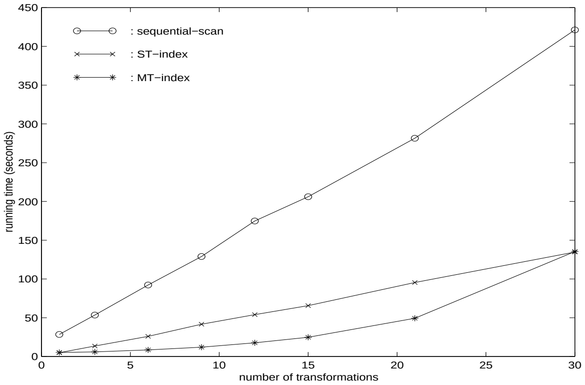

Figure 7. time per query varying the number of transformations

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

> **Query 2:** "Given a set of transformations denoted by *T*, find every pair s_1 and s_2 of stocks and every $t \in T$ such that the correlation $\rho(t(\overrightarrow{s_1.close}), t(\overrightarrow{s_2.close})) \geq 0.99$."

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 7 shows the running time of Query 2 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 also 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 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 1 using *MT-index* algorithm on real stock prices data, but this time we 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 8, 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 performance. We also computed the cost function C_k given in Equation 20 assuming $C_{DA} = 1$ and $C_{cmp} = 0.4 * C_{DA}$, i.e. a sequence comparison takes as much as 40 percent the time of a disk access. It turns out the cost function gives a good estimate of the running time and the best running time is obtained when the cost function has its minimum.

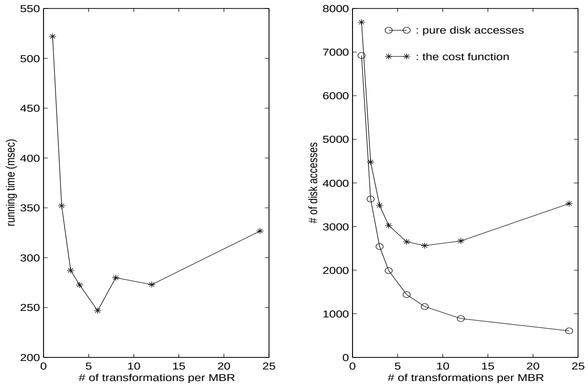

Figure 8. both the running time and the number of disk accesses varying by the number of transformations per MBRs

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 9, 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 the gap between two clusters is included in a transformation rectangle.

These experiments show that as we start packing transformations into rectangles, we see a major performance improvement which continues 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 that of 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

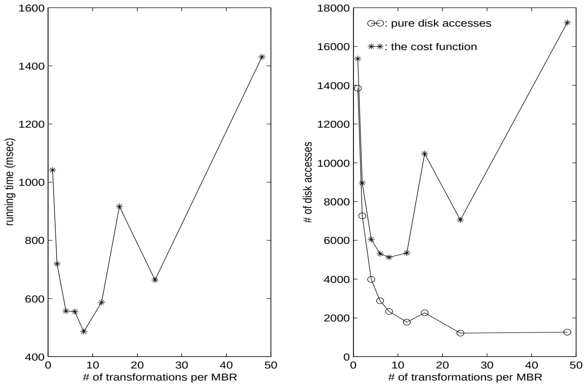

Figure 9. both the running time and the number of disk accesses varying by the number of transformations per MBRs

packing more than one cluster to a rectangle.

6 Conclusions

In this paper, we have proposed an efficient method for processing similarity queries that specify multiple transformations as the basis for similarity. We have shown that, instead of applying many single transformations to the index, we can group transformations and apply a group of them simultaneously to the index. We have discussed the possibility of grouping transformations into multiple rectangles and its effects on the performance of the algorithm. We have also shown that in the presence of some ordering among transformations, the search can be guided more efficiently. We evaluated our method over both real stock prices data and synthetic data. Our experiments confirm that the given algorithm for handling multiple transformations outperforms both the sequential scanning and the index traversal using one transformation at a time. Our contributions may be summarized as follows:

- The 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. The algorithm is guaranteed not to miss any qualifying sequence from the answer set of the query.

- Analytical results and experiments that show the effectiveness of our proposed algorithm.

- The definition of an ordering among transformations and its use in efficiently processing similarity queries.

- The observation that there is a direct relationship between the two similarity measures, the Euclidean distance and the cross-correlation, for normal form sequences. Given a threshold for one, we can easily obtain a threshold for the other.

Acknowledgments

The author would like to thank Professor Alberto Mendelzon for valuable suggestions and helpful discussions.

References

[1] R. Agrawal, C. Faloutsos, and A. 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.

[2] R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. 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.

[3] 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.

[4] 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.

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

[6] C. Faloutsos and I. Kamel. Beyond uniformity and independence: analysis of R-tree using the concept of fractal dimension. In *Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '94)*, pages 4–13, Minneapolis, May 1994.

[7] 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.

[8] 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.

[9] S. Guha, R. Rastogi, and K. 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.

[10] 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.

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

[12] D. Rafiei. *Similarity-based Queries on Time Series Data*. PhD thesis, University of Toronto, 1998.

[13] D. Rafiei and A. 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.

[14] R. Ramakrishnan and D. 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.

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

[16] 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.

[17] 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.

[18] H. Shatkay and S. Zdonic. 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.

[19] Y. Theodoridis and T. Sellis. A model for the prediction of R-tree performance. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '96)*, pages 161–171, Montreal, June 1996.

[20] B.-K. Yi, H. V. Jagadish, and C. 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 2 *For $T = \{a_1, a_2, \ldots, a_m\}$ as a set of scale factors, operator "$<$" (less than) defines an ordering among the members of T w.r.t. the domain of time sequences and the Euclidean distance.*

Proof: Let a_i and a_j be two arbitrary transformations and suppose, without loss of generality, $a_i < a_j$. Let $\vec{x}$ and $\vec{y}$ be two arbitrary time sequences 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 $a_i < a_j$. This will gives us

$$a_i.D(\vec{x}, \vec{y}) < a_j.D(\vec{x}, \vec{y}), \tag{21}$$

but we have

$$\begin{aligned} a_i.D(\vec{x}, \vec{y}) &= a_i.(\sum_{k=0}^{n-1}(x_k - y_k)^2)^{0.5} \qquad (22) \\ &= (\sum_{k=0}^{n-1}(a_i.x_k - a_i.y_k)^2)^{0.5} = D(a_i.\vec{x}, a_i.\vec{y}). \end{aligned}$$

Equations 21,22 imply $D(a_i.\vec{x}, a_i.\vec{y}) < D(a_j.\vec{x}, a_j.\vec{y})$. ∎

Lemma 3 *No ordering is possible for transformation set T as a set of (circular) moving averages w.r.t. the domain of time sequences and the Euclidean distance.*

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

$$\begin{aligned} \vec{s_1} &= [\ 10 \quad 12 \quad 10 \quad 12\ ] \\ \vec{s_2} &= [\ 10 \quad 11 \quad 12 \quad 11\ ] \\ \vec{s_3} &= [\ 11 \quad 11 \quad 11 \quad 11\ ] \end{aligned}$$

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

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

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

- Case 1: $mv2 \prec mv3$
  By Definition 1, $D(mv2(\vec{s_i}), mv2(\vec{s_j})) \leq D(mv3(\vec{s_i}), mv3(\vec{s_j}))$ 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 1, $D(mv3(\vec{s_i}), mv3(\vec{s_j})) \leq D(mv2(\vec{s_i}), mv2(\vec{s_j}))$ 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 4 *No ordering is possible for transformation set T as a set of non-circular moving averages w.r.t. the domain of time sequences and the Euclidean distance.*

Proof: The proof is similar to the proof of Lemma 3. Suppose there is an ordering among members of T. 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

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

where sequences $\vec{s_1}$, $\vec{s_2}$ and $\vec{s_3}$ are those given in Lemma 3. There are two possible orderings between *mv2* and *mv3*:

- Case 1: $mv2 \prec mv3$
  By Definition 1, $D(mv2(\vec{s_i}), mv2(\vec{s_j})) \leq D(mv3(\vec{s_i}), mv3(\vec{s_j}))$ 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 1, $D(mv3(\vec{s_i}), mv3(\vec{s_j})) \leq D(mv2(\vec{s_i}), mv2(\vec{s_j}))$ 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. ∎