Fast And Scalable Similarity and Correlation Queries on Time Series Data

Philon Nguyen

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfilment of the Requirements for

the Degree of Master of Computer Science at

Concordia University

Montréal, Québec, Canada

April 2009

Your file Votre référence
ISBN: 978-0-494-63255-0
Our file Notre référence
ISBN: 978-0-494-63255-0

ABSTRACT

Fast And Scalable Similarity and Correlation Queries on Time Series Data

Philon Nguyen

Time series are ubiquitous in many fields ranging from financial applications such as the stock market to scientific applications and sensor data. Hence, there has been an increasing interest in time series indexing over the past years because there has been an increasing need for fast methods for analyzing and querying these datasets that are often too big for practical brute force analysis. We start with the main contributions to the field over the past decade and a half. We will then proceed by describing new solutions to correlation analysis on time series datasets using an existing index called the Compact Multi-Resolution Index (CMRI). We describe new algorithms for indexed correlation analysis using Pearson's product moment coefficient and using the multidimensional correlation coefficient and introduce a new measure called Dynamic Time Warping Correlation (DTWC) based on Dynamic Time Warping (DTW). In addition to these linear correlation algorithms, we propose an algorithm called rank order correlation on a non-linear/monotonic measure. To support these algorithms,

we revised the Compact Multi-Resolution Index (CMRI) and propose a new index for time series datasets which improves over the sizes, speed and precision of CMRI. We call this index the reduced Compact Multi-Resolution Index (rCMRI). We evaluate the performance of rCMRI compared to CMRI for range queries and range query based queries.

Acknowledgments

I would like to express my deepest gratitude to my supervisor Dr. Nematollaah Shiri. His guidance and encouragement made my thesis work a pleasant and extremely educational experience.

More than anyone else, I would like to thank my parents, Mr. and Mrs. Nguyen and my sister Nathalie Nguyen. It was their continued support and encouragement that made this work possible.

Contents

List of Figures

List of Algorithms

List of Tables

Symbols and Acronyms

Symbols	Definitions
MBR	Minimum Bounding Rectangle
DFT	Discrete Fourier Transforms
DWT	Discrete Wavelet Transforms
PAA	Piecewise Aggregate Approximation
APCA	Adaptive Piecewise Constant Approximation
SAX	Symbolic Aggregate Approximation
X,Y	Reduced time series from x,y
N	Length of a reduced series
n	Length of a series
D(x,y)	Distance between series x and y
$D_{LB}(X,Y)$	Lower-bouding distance between reduced series X and Y
DTW(x,y)	Dynamic Time Warping Distance between x and y
R(x,y)	Correlation between x and y
DTWC(x,y)	Dynamic Time Warping Correlation between x and y
r	Correlation parameter
ε, e	Range query parameter
CMRI	Compact Multi Resolution Index
w	Window size
q	Query length
a,b	Resolution range of multi-resolution indexes
~	Normalization operator
r_X	Ranked transformation of series X
rCMRI	reduced Compact Multi-Resolution Index
kNN	k nearest neighbour query

Chapter 1

Introduction

In the past years, time series have become ubiquitous to many application domains such as stock market data streams, meteorological sensor data or scientific datasets. With advances in technology, voluminous amount of data is being collected for pattern analysis and searches. For this, many techniques have been developed to index time series data for high performance queries and analysis. These techniques are an efficient substitute for sequential scan which is often too costly and even sometimes infeasible when the data size is huge. A first efficient solution for similarity matching on time series datasets was proposed in 1995 by Rakesh Agrawal, Christos Faloutsos and Arun Swami [AFS95].

The first chapter of this thesis is an overview of the main time series indexing techniques that have been proposed since 1994. Since for most cases, these indexes are a mixture of spatial data structures, dimensionality reduction and query and analysis algorithms, we will adopt a component base approach. By this we mean

that we will intensively review each ingredient that time series indexes may contain before considering the different recipes that have been developed. This helps the reader to see the new recipes for time series indexes that can be developed by mixing in an original manner different ingredients together.

A persistent collection of time series is called a time series database. Data volumes expected in time series databases may range from large to enormous. Efficient algorithms are necessary to mine this data since usual brute force algorithms based on sequential scan do not scale. Hence, it is necessary to structure and transform the raw time series in order to be able to perform useful data mining operations and extract valuable knowledge. This process of structuring time series for fast query processing and analysis is called time series indexing.

The components of all time series indexing techniques surveyed in this thesis are dimensionality reduction, matching measures and algorithms and the actual indexing technique. The literature is vast on these components: we will focus on major works relevant to the field of time series mining and we will present a taxonomy of current methods for each of these components.

Recently, interesting new and experimental data mining techniques have surfaced in the media, in sci-fi action movies and in the world of computer science. Users now want to be able to perform online queries over huge petascale databases of multimedia data by content. Here are a few examples,

1. National security agencies want to be able to see if a known terrorist was spotted

in any national airport video surveillance system.

2. Doctors feed MRI, EEG, ECG or any other physiological data sets to a database in order to get an accurate diagnostic.

3. Stock brokers want to see if any stock in the world market followed some pattern in order to make buying and selling decisions.

4. GIS data could be queried by content. For example, users may want to find the number of houses in a 2 mile radius of any nuclear power plant.

5. Scientists need to analyse data coming from the new CERN Large Hadron Collider which is estimated to produce roughly 15 petabytes per year..

All these application domains have a common denominator: the data is spatial. Hence, traditional alpha-numeric databases and data structures are inefficient and sometimes insufficient: we need spatial data structures. These emerging applications are faced with huge computational challenges that can only be addressed by efficient data structures.

Information and data streams expressed in terms of time series are often too large in their raw form to be used by humans for decision making. Economists, for example, are confronted with huge amounts of stock market data which they must relate to micro-economic and macroeconomic data. For example, stock market data can stream every second generating petascale databases. Efficiency is a central issue for correlation analysis and search in time series, and is required for many such

applications, examples of which are as follows.

1. Stock brokers need to correlate stock market patterns to macroeconomic patterns such as interest rates or exchange rates or to correlate stock market patterns amongst themselves. Stocks could also be clustered based on their correlation values [CC07].

2. Sensor network data could be mined using correlation analysis. Airplane engines sensor networks could make use of correlation queries in order to determine if repairs are needed.

3. Click-stream and transactional data analysis could benefit from faster correlation analysis. For example, in the customer relation management domain, we could correlate a marketing event to the number of clicks on certain web pages.

4. In pattern matching, correlation queries could be used when query and target patterns are not identical but only correlated. Since we index the time series data over sliding windows, we can match shifted patterns too [JMK05]. Furthermore, Dynamic Time Warping (DTW) based correlation could correlate skewed patterns.

1.1 Spatial Data Structures

Our review is focused more towards the R-tree family of spatial indexes, from low to high dimensional implementations, as they seem to be ubiquitous in the related

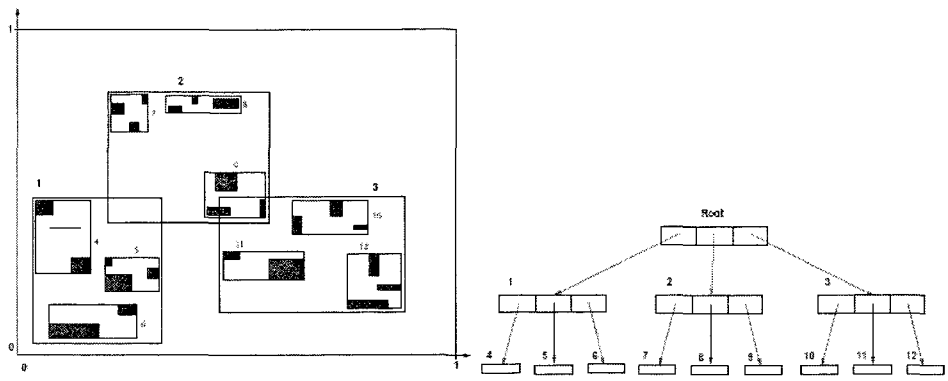

Figure 1.1: R-tree structure [MNPT06].

literature on time series indexing.

1.1.1 Hash-Based Solutions

Grid files are a common hash-based spatial data structure [Sam06]. Technically, they are a multi-dimensional array such that the dimensions $X_1, X_{2,\ldots,X_n}$ correspond to the spatial coordinates $x_1, x_2, \ldots, x_n$ of the object to be indexed. Many queries can be performed efficiently on Grid files. The D-index is another hash-based spatial data structure for metric distance queries [DGSZ03].

Basically, the D-index generates buckets dynamically according to the dataset and from a specific metric distance measure on which different queries can be performed.

1.1.2 Tree-Based Solutions

We quickly review some terms and concepts which will help better understand the related literature that follows. We will rapidly browse through sequential variants of

the original R-tree proposed by Guttman [Gut84] such as the R*-tree [BKSS90], the R+-tree [FSR87], the X-tree [BKK96] and the M-tree [CPZ97].

The R-tree is an efficient data structure for storing spatial objects such as CAD/CAM geometries, GIS data, time series data, or multimedia objects. Any point query (exact match search) can be resolved at best in O(log N), where N is the number of objects in the dataset. In addition to supporting point queries, R-trees also support other types of queries including range queries, nearest-neighbour queries, spatial joins and closest pairs queries [MNPT06].

The basic idea in R-trees is that an object's boundary can be approximated by a Minimum Bounding Rectangle (MBR). Leaf nodes of an R-tree contain the actual data objects while internal nodes contain an MBR which englobe the MBR's of their child nodes. This is shown in Figure 1.1 [MNPT06]. A fuzzy extension to MBR, called fuzzy MBRs, is proposed in [dCdTB04], which can deal with uncertainty in similarity queries.

1.1.3 Variations on the R-Tree

The R+-tree is actually an improvement based on two negative characteristics of R-trees:

1. MBR's overlap and hence multiple paths are visited for a single query.

2. Some large rectangles may increase overlap.

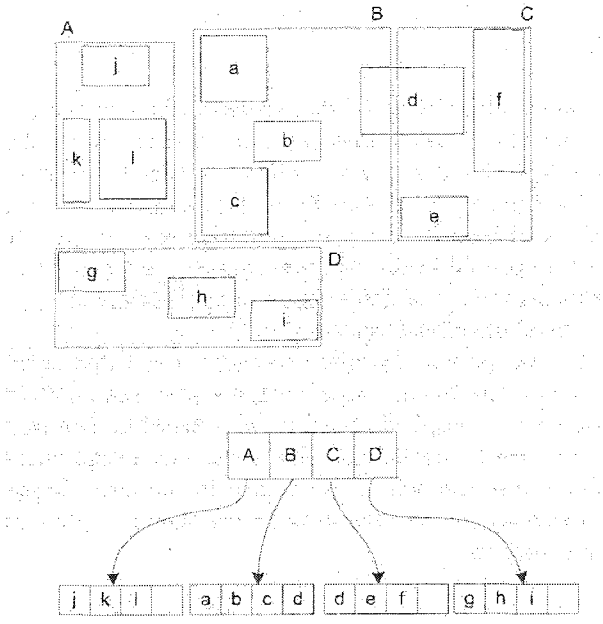

Figure 1.2: R+-tree structure [MNPT06].

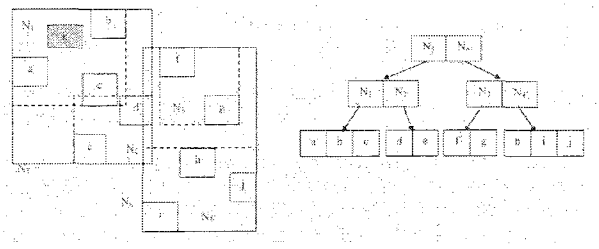

Figure 1.3: R*-tree structure [MNPT06].

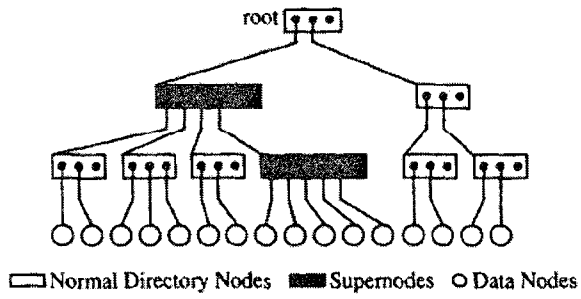

Figure 1.4: X-tree structure [BKK96].

To avoid these problems, the R+-tree lets objects be divided between two or more MBR's hence achieving minimum overlap and balanced MBR size. As shown in Figure 1.2 [MNPT06], object d is divided between MBR B and C.

The R*-tree on the other hand tries to minimize the area covered by each MBR and minimize the overlap between them by using heuristics, one of which is an insertion strategy based on which MBR needs the less increase in size. Figure 1.3 [MNPT06] shows an R*-tree.

Both the R+-tree and the R*-tree are low dimensional data structure meaning that they perform well in practice when their MBR's have a dimension lower than 8.

High-Dimensional R-Trees

The X-tree is an R-tree for higher dimensional data. Its basic heuristic says that when data is highly overlapped, a linear data structure is better whereas when data is not overlapped, a hierarchical data structure is better. In the X-tree, directory rectangles are hierarchical whereas supernodes are linear. Figure 1.4 [BKK96] shows an instance of an X-tree.

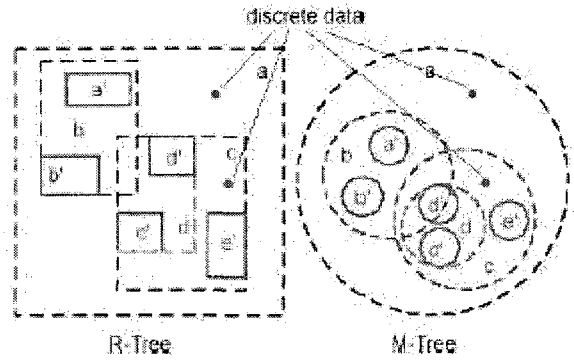

Figure 1.5: An R-tree compared to an M-tree [JOV06].

The M-tree is a metric data structure, which means that it is meant to index objects for metric queries, queries whose distance function follows the triangle inequality. The M-tree indexes objects according to their mutual distance rather than their minimum bounding rectangle. A comparison between the R-tree and the M-tree is shown in Figure 1.5 [JOV06]. The M-tree was shown to perform well for higher dimensionality [CPZ97].

We mention briefly that other variants of R-trees for parallel and distributed architectures exists, such as those proposed in [FK92, FKK96, LS99, HKMW99, MLR07a, MLR07b, JOV06, JOV05, AAF99].

1.2 Dimensionality Reduction Techniques

Let us begin with a simple definition of dimensionality reduction. A dimensionality reduction algorithm projects an n-dimensional object A in R^n onto an m-dimensional object B in R^m where $n > m$ such that some properties of A, for instance distance

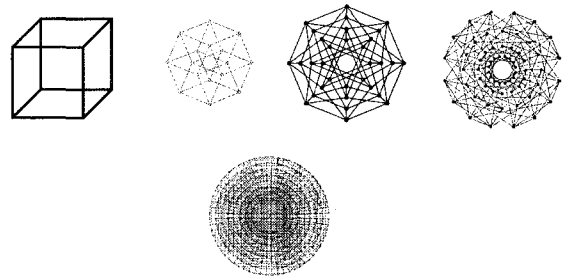

Figure 1.6: Hypercubes [cf. Wikipedia].

predicates, are preserved in B. For example, if an object A can be described by 100 values, a reduced object B obtained from A could for example be represented by 10 values such that some properties of A would be preserved in B.

A first goal of dimensionality reduction is to improve the overall computational efficiency by reducing the total number of operations to be performed on an object and the space required to store the object. However, when dealing with time series, another more important factor is to be considered as shown in [LV06]: as the dimensionality of a data object grows towards infinity, the discriminating power of distance measures reduces. In other words, as dimensionality of random vectors tends towards infinity, the norm of these random vectors approaches a constant, which is the mean of all these random vectors. We can show this in a different way by showing that high dimensional objects behave abnormally. For example, a circle and an imbricated square in the circle have different perimeters but as their dimensionalities grow towards infinity, their perimeters tend to be equal. This is shown in Figure 1.6, where 2D projections of hypercubes are shown for increasing dimensions.

There are three main types of dimensionality reduction techniques used for time

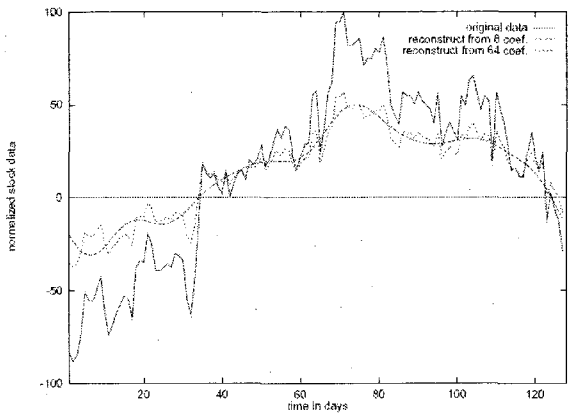

Figure 1.7: DFT of time series data [AWA00].

series mining, as follows. We consider representative works for each type.

1. Spectral: Discrete Fourier Transforms (DFT), Discrete Wavelet Transforms (DWT)

2. Piecewise: Piecewise Aggregate Approximation (PAA), Adaptive Piecewise Constant Approximation (APCA)

3. Symbolic: Symbolic Aggregate Approximation (SAX)

We next review the main techniques developed under the above taxonomy. All these techniques are lower-bounding and follow the requirements of the GEMINI as we will see.

1.2.1 Spectral Methods

Discrete Fourier Transforms

Discrete Fourier Transform (DFT) is a well known technique in fields such as signal processing. The first time series indexing techniques proposed in the literature originally used DFT as dimensionality reduction technique. Given a sequence $s =< x_0, x_1, ..., x_{n-1} >$, the DFT coefficients of s denoted $S =< X_0, X_1, ..., X_{n-1} >$ is defined as follows [SZ02]:

$$X_F = \frac{1}{\sqrt{n}} \sum_{i=0}^{n-1} x_i e^{-2j\pi Fi/n}, F = 0, 1, ..., n-1 \tag{1.1}$$

The inverse DFT is given by:

$$x_i = \frac{1}{\sqrt{n}} \sum_{F=0}^{n-1} X_F e^{2\sqrt{-1}\pi Fi/n}, i = 0, 1, ..., n-1 \tag{1.2}$$

If all the coefficients are kept from the transformation, then the Euclidean distance between two non-transformed sequences and two transformed sequences is the same (Parsival's theorem). However, usually, only the first k coefficients are kept, hence the use of DFT as dimensionality reduction technique. In the case where the first k coefficients are kept, it has been shown that the Euclidean distance between the transformed and reduced coefficients lower bounds the real distance [AFS95]. Figure 1.7 [AWA00] shows a DFT.

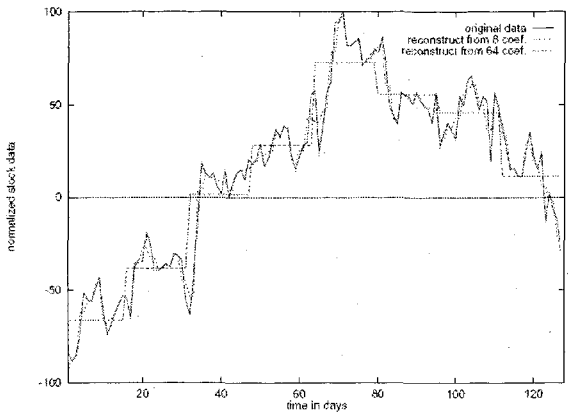

Figure 1.8: DWT of time series data [AWA00].

Discrete Wavelet Transform

The Haar wavelet is the first known wavelet transform [Haa10]. Figure 1.8 [AWA00] shows a Haar wavelet transform. The Haar wavelet is used in [CFY03] for which an exact algorithm is provided, which implements the following expression. The Haar wavelet is used in , its exact algorithm can also be found there and its mathemaitcal expression is given by:

$$\psi_i^j(x) = \psi(2^j x - i), i = 0, ..., 2^{j-1} \tag{1.3}$$

where

$$\psi(t) = \begin{cases} 1 & 0 < t < 0.5, \\ -1 & 0.5 < t < 1, \\ 0 & otherwise \end{cases}$$

given a scaling function:

$$\phi = \begin{cases} 1 & 0 < t < 1, \\ 0 & otherwise \end{cases}$$

1.2.2 Piecewise Aggregate Methods

Piecewise Aggregate Approximation

The Piecewise Aggregate Approximation (PAA) technique was introduced independently in [FY00, CKMP01a]. The idea is to segment a time series into equal length intervals and compute the mean of the points falling in each interval. PAA coefficients are given as follows [CKMP01a] (note that indexes start at 1):

$$X_i = \frac{N}{n} \sum_{j=\frac{n}{N}(i-1)+1}^{\frac{n}{N}i} x_j \tag{1.4}$$

The Euclidean distance between two PAA reduced sequences is [CKMP01a]:

$$D(X,Y) = \sqrt{\frac{n}{N}} \sqrt{\sum_{i=1}^{N} (X_i - Y_i)^2} \tag{1.5}$$

As for the previous methods, the Euclidean distance between reduced sequences using PAA lower bounds the real distance.

Adaptive Piecewise Constant Approximation

The Adaptive Piecewise Constant Approximation (APCA) introduced in [CKMP01b] looks like a DWT or PAA transform. However, in the APCA reduction, segments approximating the series can be of variable length. Hence a leading value and a mean

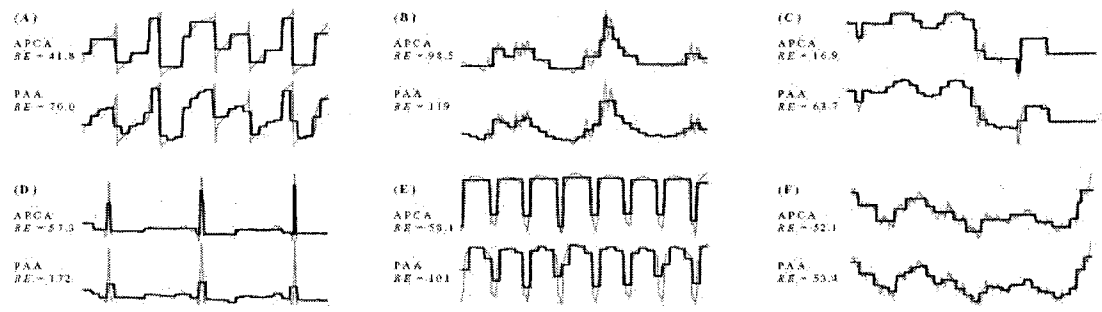

Figure 1.9: PAA compared to APCA reductions [CKMP01b].

approximation value are required for each approximating segment. The algorithm starts by performing a DWT transformation over the time series to be reduced. Then it combines some segments together using an algorithm which runs in O($n$log$n$) and which finds an optimal combination scheme for the DWT segments. Furthermore, [CKMP01b] shows how APCA is lower bounding and how distance measures can be computed over the APCA reduced coefficients. We will not go into the details of these distance measures, however, the basic idea is to bound each APCA segment with a maximum value and a minimum value taken from the original non-reduced time series and compute distance measures from these values. Figure 1.9, taken from [CKMP01b] illustrates how APCA is superior in terms of precision compared to PAA.

1.2.3 Symbolic Methods

Symbolic Aggregate Approximation

SAX was introduced by Keogh et al. [KLLC03, KLLC07]. The idea underlying this technique is to discretize a time series into a string representation such that traditional

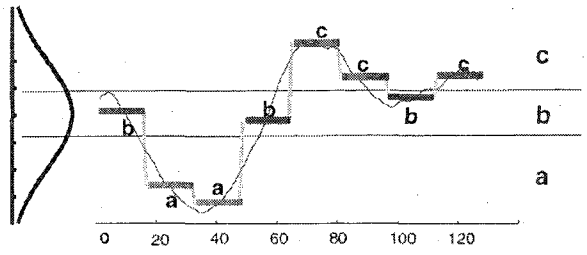

Figure 1.10: SAX transformation [KLLC03].

string data structures (e.g. suffix trees and prefix trees) can now be used to index the sequences. The algorithm starts by transforming the time series into its PAA representation. A normalization is usually applied prior to dimensionality reduction in order to achieve equiprobable occurrence of symbols. A lookup table is then used to convert values falling in a pre-calculated interval to symbols. For example, a rule could be of the form "if $0.18 < x < 0.43$ then x is mapped to A", where x is a PAA coefficient and A is a SAX coefficient. Figure 1.10 taken from [KLLC03] shows a SAX representation of a time series.

1.2.4 Comparisons

When choosing a dimensionality reduction technique, there is a trade-off to consider between size and precision. Note that a common measure of effectiveness of a dimensionality reduction technique is to compare the Euclidean distance between the original time series and their transformed counterparts. Discrete Fourier Transforms (DFT) applied to time series was proposed in [AFS95]. The idea is to use a sliding window of n values over time sequences and transform the n values and keep N

values, where N is the number of DFT coefficients and is much smaller than n, the length of the sliding window. The DFT transformation runs in O(nlogn) and needs 2N values for storage since the transformed data requires storing real and complex values. However, the space utilization of DFT can be reduced to N assuming the complex conjugate property of DFT. The Discrete Wavelet Transform (DWT) [CFY03], improves DFT as it runs in O(n) and requires N coefficients for storage. However, in practice, DWT coefficients are very large and for sequences shorter than 10,000 values, DFT and DWT perform the same. Comparisons between DFT and DWT are discussed in [AHK01]. Piecewise Aggregate Approximation (PAA) performs as well as DWT when the size n of the sliding window is a power of 2, but performs better than DWT for other values of n. One of the issues in DWT is the window size that has to be a power of 2. When it is not the case, DWT pads the values with 0's to make the window size attain a power of 2, hence reducing the precision of the transform. As in DWT, the run time complexity of PAA is O(n). As an improvement over PAA, Adaptive Piecewise Constant Approximation (APCA) proposed in [CKMP01b] runs in O(nlogn) and offers a better approximation than PAA when the number of segments calculated are less or equal than PAA's. However, APCA needs 2N coefficients for storage, a leading value and a mean value. All the above mentioned methods are lower bounding and satisfy, as we will see, the GEMINI framework requirements.

The choice of a particular dimensionality reduction usually involves considering a number of factors including space utilization, precision, complexity, indexability and the different distance measures it supports. As mentioned before, APCA is

more precise than PAA since it uses variable length segments to approximate the original sequence. However, PAA supports Dynamic Time Warping based distance measures; we found no study on this for APCA. SAX is reported as being the fastest dimensionality reduction method, however, an explicit time series index using it was only presented recently [AKFA08].

Other common feature extraction and dimensionality reduction techniques such as those based on Principal Component Analysis (PCA) and Neural Networks, while interesting in some application domains, are not suitable for time series. The main reason for this is that those techniques often require many iterations before they converge and they also require global information, although some implementations may be incremental. In time series, the data volume does not allow iterations over the whole data set and the information about the data may be incomplete at any given time since we do not know what may happen a minute or an hour from now.

1.3 Absolute and Pattern Based Similarity Measures

We call a measure absolute if it computes a distance between two time series according to some given predefined metric. Examples of absolute distance measures are the Euclidean distance, Manhattan distance, Chebyshev's distance or Dynamic Time Warping distance. Since we are dealing with time series, we only consider distance measures over numerical domains and exclude distance measures over categorical

data such as Jacquard's coefficient. Absolute distance measures can be metric or non-metric. For example, the Euclidean distance is metric and the DTW distance is non-metric since it does not respect the triangle inequality. We call a measure pattern-based if it does not necessarily compute an absolute distance but rather, for example, the similarity in form of two time series. For example, a correlation query would look like this: "Find all subsequences in a time series database that move together with a degree of correlation higher than 90%".

1.3.1 Absolute Similarity Measures

Lp Norms

Lp norms are a group of distance measures that can be expressed as follow:

$$\|\vec{x}|\| = (\frac{1}{N}\sum_{n=0}^{N-1}|x_n|^p)^{1/p} \tag{1.6}$$

For $p = 1$, we get the Manhattan distance and for $p = 2$, we get the well known Euclidean distance. We get Chebyshev's distance for $p = \infty$.

Dynamic Time Warping

Exact indexing of dynamic time warping distance measures was proposed in [KR04] using PAA as a dimensionality reduction technique. It uses lower bounding distance measures of dynamic time warping distance that can apply to value space (LB_KEOGH) or to feature space (LB_PAA). Dynamic time warped distance is ob-

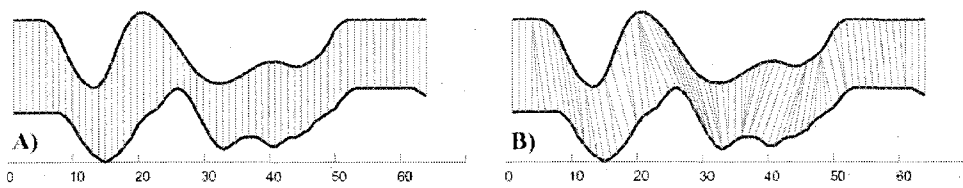

Figure 1.11: Comparison between Euclidean distance and DTW [KR04].

tained from the following recurrence equation where $d(x_1, x_2)$ is any given distance measure (in our case, the Euclidean distance):

$$DTW(i,j) = d(q_i, c_j) + min(DTW\ of\ adjacent\ cells) \tag{1.7}$$

where "'DTW of adjacent cells"' is given by:

$$DTW(i-1, j-1), DTW(i-1, j), DTW(i, j-1) \tag{1.8}$$

Figure 1.11 [KR04] shows how DTW works in comparison with Euclidean distance measures.

1.3.2 Absolute Similarity Queries

Range Queries

A range query could be stated as follow: "Find all sequences of a time series database whose distance from a query sequence is less than a range parameter r." Any distance function could be used such as the Euclidean distance or the Dynamic Time Warping (DTW) distance. Range queries are very common in practice and are often used as the basis of other queries such as nearest neighbor queries and simple correlation queries. Here is the outline of a range query algorithm over a standard R-tree index:

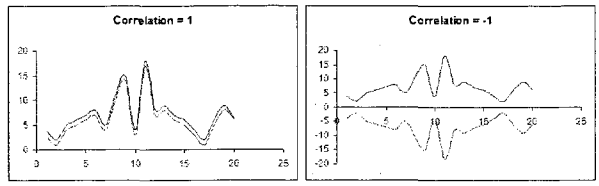

Figure 1.12: Perfect positive and negative correlation cases.

1. Given a range parameter r and a query point Q, construct a circle centered at Q of radius r.

2. Traverse the R-tree from the root and visit the MBR's that intersect the query circle.

3. For every leaf node visited, compute the real distance of the leaf object to Q. If the distance is less than r, return the leaf object.

Nearest Neighbor Queries

A nearest neighbor query could be stated as follow: "Find the 10 closest sequences to a given query sequence." Nearest neighbor queries are also very common and generally use range queries in their algorithms. In this thesis, we will not discuss other spatial queries, however, the reader is referred to [MNPT06] for a complete survey of queries supported by R-tree indexes.

1.3.3 Pattern Based Similarity Measures

Product Moment Correlation

A correlation query could be stated as follows: "Find all subsequences that are correlated to a given subsequence with r¿0.99.". A bivariate correlation query involves two input variables. A frequently used measure for bivariate correlation is Pearson's product-moment coefficient [Pea01], which we use in our correlation queries and is defined as follow:

$$R(X,Y) = \frac{\mu_{X,Y} - \mu_X \mu_Y}{\sigma_X \sigma_Y} \tag{1.9}$$

where $X =< x_1, x_2, \ldots, x_n >$ and $Y =< y_1, y_2, \ldots, y_n >$ are two sequences, R is the Pearson's coefficient with $-1 \leq R \leq 1$, μ is the mean, and σ is the variance. This measure reflects the degree of linear relationship between the two input variables: value +1 indicates perfect positive linear relationship, -1 indicates perfect negative, and 0 indicates there is no linear relationship between the two variables. Figure 1.12 illustrates two examples of input variables and their correlation values.

Multivariate correlation has the following quadratic form when the x_i's are known variables given at query time [KM03]:

$$R^2(X_i, Y) = \rho^T_{X_i,Y} R^{-1}_{X_i,x_j} \rho_{X_i,Y} \tag{1.10}$$

where $\rho_{X_i,Y}$ is the vector composed of Y's Pearson correlation coefficients with X_i and $R(X_i, X_j)$ denotes the correlation matrix of variable X_i with X_j such that $R(X_i, X_j) = 1$ when $i = j$. Note that the matrix is symmetric and that when

$X_i = \{X_1\}$, the multivariate formula reduces to Pearson's coefficient. Geometrically, multivariate correlation measures how well a hyperplane can fit the measured data.

Rank-Order Correlation

Rank order correlation was introduced by C. Spearman [Spe04]. Let n be the length of a sequence X, $r_{X,i}$ be the rank of the i^{th} element in X and $r_{Y,i}$ be the rank of the i^{th} element in sequence Y, then the rank order correlation r can be defined as follow [KG90]:

$$R(X,Y) = 1 - \frac{6\sum_{i=1}^{n}(r_{X,i} - r_{Y,i})^2}{n^3 - n} \tag{1.11}$$

When ties occur in the ranking, the ranking associated to the tied value is the average of all tied rankings. Furthermore, we can define U and V as functions of the number of ties u_i and v_i, where i ranges over the number of values that are tied:

$$U = \frac{1}{12}\sum(u_i^3 - u_i) \tag{1.12}$$

$$V = \frac{1}{12}\sum(v_i^3 - v_i) \tag{1.13}$$

In case of ties, the following formula can be used to compute r:

$$r = 1 - \frac{6\sum_{i=1}^{n}(r_{X,i} - r_{Y,i})^2 + U + V}{n^3 - n} \tag{1.14}$$

Rank order correlation is a useful non-parametric substitute for the product moment correlation coefficient when the data analyzed does not satisfy assumptions such as interval scale measurement, linearity and the normal distribution of the input sequences X and Y. Furthermore, rank order correlation can help detect nonlinear monotonic relationships between sequences [DR05].

DTW Correlation

Since correlation queries can be mapped to Euclidean distances, we proposed a new measure, called Dynamically Time Warped Correlation (DTWC) [NS08a] obtained from the original DTW formula by applying the bivariate correlation condition described in [SZ04]. This yields:

$$R_{DTW}(X,Y) \geq 1 - \epsilon^2 \Rightarrow D_{DTW}(\tilde{X},\tilde{Y}) \leq 2n\epsilon^2 \tag{1.15}$$

where $R_{DTW}(X,Y)$ is the DTWC coefficient, $D_{DTW}(X,Y)$ is the DTW distance between X and Y [MR81], symbol $\tilde{}$ denotes normalization, and n is the pattern length. DTWC has the same advantages over correlation that DTW distance has over Euclidean distance: it can adapt itself to unequal length and warped data sets. For example, consider the following two time series: < -1, 1, 1, 1, 4, 5, 7, 5, 4 > and < -1.1, 1.1, 4.1, 5.1, 7.1, 5.1, 4.1, 4.1, 4.1 > whose normalization are < -2.1547, -0.1547, 2.8453, 3.8453, 5.8453, 3.8453, 2.8453, 2.8453, 2.8453 > and < -2.6665, -0.4665, -0.4665, -0.4665, 2.6335, 3.5335, 5.5335, 3.5335, 2.5335 >. We can see that the series are simply shifted and skewed. The standard correlation between both series is 0.5280 and would not be picked up by most correlation queries. However, the DTWC between the two series is 0.9088 (considering a Sakoe-Chiba band of 2) and could be picked up by a DTWC query with $r > 0.9$. DTWC between the two series can be computed from the correlation between their two DTW expansion which is of length 11. This gives the sequences < -2.1547, -0.1547, -0.1547, -0.1547, 2.8453, 3.8453, 5.8453, 3.8453, 2.8453, 2.8453, 2.8453 > and < -2.6665, -0.4665, -

0.4665, -0.4665, 2.6335, 3.5335, 5.5335, 3.5335, 2.533, 2.533, 2.533 >. Note that the Euclidean distance between the normalized series is 6.7794 which is larger than the DTW distance of 1.087.

1.3.4 Pattern-Based Similarity Queries

Product Moment Correlation Queries

A typical bivariate correlation query would be "Find all subsequences of a stock market time series database that are correlated with a given subsequence of oil price values by a factor greater than 99%." A typical multivariate query would be "Find all subsequences of a stock market time series database that are correlated with a given subsequence of oil price, US interest rate and US inflation rate values by a factor greater than 99%." Simple correlation queries were studied in [SZ02, SZ04, HKSZ03c, HKSZ03a, HKSZ03b, NS08a], and multivariate correlation queries were addressed in [NS08a] using CMRI which is based on R-trees. [SZ02, SZ04] made use of a scheme called Statstream, whereas [HKSZ03c, HKSZ03a, HKSZ03b] developed a data structure called the Cone Tree which made use of the fact that the product moment coefficient can be expressed as a vector product, hence a cosine, since the data structure indexed angles.

In general, simple correlation queries involve computing a range query over normalized time sequences, whereas multivariate correlation queries are more complex and involve segmentating a solution space whose axis are the query variables, gener-

ating a set of rules from the segmentation and then querying an R-tree based index using these rules. For more detailed information, the reader is referred to [NS08a].

Rank-Order Correlation Queries

We proposed a first solution for indexed rank-order correlation queries in [NS08b]. Rank-order correlation queries are similar to bivariate correlation queries but rather than using the product moment coefficient, we use the rank-order correlation coefficient which can detect non-linear monotonic relationships.

The rank-order correlation algorithm consists in a range query over the rank feature space of the time series. Indeed, for rank-order correlation queries, the usual sliding/disjoint windows are not indexed but rather their ordering.

DTW Correlation Queries

DTWC queries are similar to bivariate correlation queries but make use of DTWC measure we introduced in [NS08a]. DTWC queries detect subsequences that would be missed by normal product moment based queries due to skewing or warping.

DTWC queries are similar to DTW queries [KR04], however, they are performed over the normalized time series.

1.4 Time Series Indexes

There are a number of classifications of indexing techniques for time series. This section, we discuss and highlight properties of different indexes. We will then discuss

some of these indexes in more details.

1.4.1 The GEMINI Framework

The GEMINI framework was developed by Faloutsos et al. [Fal96]. It constitutes the basis of most if not all time series indexing techniques. Its statement is simple: An index using a given dimensionality reduction technique will incur no false dismissals if and only if the dimensionality reduction technique lower-bounds the real distance. Incurring no false dismissals means that the index will not omit any correct answer. The lower bounding property actually means that the distance between two reduced time series is smaller or equal to the distance between the two original time series. The GEMINI framework opened the way for time series indexing and efficient search algorithms by providing correctness theorems and a general framework for indexing time series. Usually, an indexing solution will present something called a MINDIST and show that it lower-bounds the real distance measure to be indexed.

1.4.2 Problem Solved Taxonomy

A first taxonomy of indexing techniques could be based on the problem solved by the index. By problem solved we mean the constraints made on the length of the query by the indexing solution for example. Some indexes solve whole matching efficiently, others consider subsequence matching, and others handle variable length queries. Furthermore, some indexing strategies were specifically designed to reduce the index size.

Whole Matching

The index proposed in [AFS95] solved the problem of efficiently retrieving similar time series of fixed length. This is usually called the whole matching problem in the time series indexing literature. By efficiency we mean that the solution avoids sequential scan (at least in most cases where the number of objects retrieved is much smaller than the total number of objects stored). Agrawal et al. [AFS95] proposed to use R-trees to index multi-dimensional points (i.e. the time series). However, the problem of high dimensionality remained since the proposed solution using of R-trees does not perform well with high dimensional data due to overlapping of MBR's. Hence, the researchers proposed to reduce the dimensionality of the data using DFT such that distance predicates used in similarity queries such as Euclidean distance based predicates would be preserved in the transformed feature space.

Subsequence Matching

Another problem set is subsequence matching. Let $X =< x_1, x_2, x_3, \ldots, x_n >$ be a time series. Subsequence matching addresses the problem of matching a subsequence, e.g. x_2, x_3, with all possible sequences of for example length 2 in X. A first solution to this was proposed in [FMR94] and was called the I-Adaptive index. The idea was to index all the sliding windows of length w of a time series into an R-tree. As we will discuss later, this results in large indexes due to the redundancies produced by the use of sliding windows. The I-Adaptive index also solved variable length queries, however better solutions to variable length queries were devised.

Variable Length Queries

Solving variable length queries means that an index must deal with the fact that the length of the query is only known at query time. Although the I-Adaptive index are also suitable for such queries, a better solution (at least in terms of precision defined as a function of the number of false positives retrieved) was proposed in [KS04]. The solution uses a Multi-Resolution Index (MRI) which stores a collection of I-Adaptive indexes at different resolutions 2^a to 2^b. Suppose we already have I-Adaptive indexes for resolutions 8, 16 and 32. Now suppose we have a range query of length 56 and range parameter r, the index is queried by partitioning the query into a query of length 8, one of length 16 and one of length 32 such as 56 = 8+16+32. We then start by querying one of the resolutions of the MRI. [KS04] proved that after each sub query, in general the range parameter r used can be reduced, and hence, further pruning occurs when another query is made on another corresponding resolution.

Variations of the MRI exist. The Compact MRI (CMRI) was proposed as an improvement in size and precision over the MRI [KS07]. The reduced CMRI (rCMRI) further improved over the size and precision of the CMRI by actually combining the CMRI with Duality Matching (that we will discuss in the next section).

Reducing the Index Size

A common problem with sliding window indexes is the size of the index. For each time series of length s, we need to insert $s-w+1$ points with dimension d in the index. This results in storing $d*(s-w+1)$ numbers, where d is the reduced dimensionality.

Duality Matching [LMW00, LMW01] proposes a disjoint window solution. Hence the number of points in a time series database will be roughly equal to the number of points stored in the index. [LMW00] showed that using disjoint windows is sufficient in order to solve range and nearest neighbor queries. In order to do so, the solution uses sliding windows over the query sequence and disjoint windows over the indexed points. Hence, for a query length q, it performs $q - w + 1$ queries over an index that is smaller by a factor of $1/w$. Even with a higher number of queries to be performed, the index is still comparable and sometimes better than sliding window solutions due to the reduced index size.

1.4.3 Structural Taxonomy

Structurally, time series indexes can be first divided into two categories: single resolution indexes and multi-resolution indexes. They can also be divided into two other categories: sliding window indexes and disjoint window indexes. Single resolution indexes can solve whole matching and subsequence matching efficiently. Multi-resolution indexes were developed to increase the precision of single resolution indexes for variable length queries. Precision is here defined as a function of the number of false positives retrieved at query time. Disjoint window indexes are an improvement in terms of size over sliding window indexes. While disjoint window indexes can solve most queries, they cannot answer for example multivariate correlation queries.

Single Resolution Indexes

The whole matching index proposed by Agrawal et al. [AFS95] is a single resolution index. The I-Adaptive index is also a single resolution index, as well as its improvement, Duality matching. All these indexes can serve as backbone for multi-resolution indexes. They solve efficiently the whole matching and the more general problem of subsequence matching.

Multi-Resolution Indexes

When the query length is defined at query time, it is difficult to fine tune a single resolution index for any given query length. Multi-resolution indexes were proposed as an extension over single resolution indexes to address this problem. For example, MRI and CMRI are actually a collection of I-Adaptive indexes, whereas rCMRI is a collection of Duality Matching based indexes. Specific similarity algorithms were developed in order to improve the pruning over multiple index queries required by multi-resolution indexes. Multi-resolution in general improve the precision of the answer set, however, it comes at a cost of significant increase in the index size which is a function of the number of resolutions stored.

Sliding Window Indexes

Examples of indexes that originally made use of sliding windows include I-Adaptive, MRI and CMRI. The main problem with such indexes is the size of the index. Larger indexes also cause the query response time to be slower in general. Sliding windows

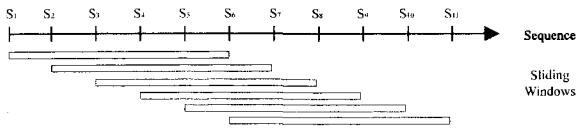

Figure 1.13: Sliding windows.

Figure 1.14: Disjoint windows.

are shown in Figure 1.13.

Disjoint Window Indexes

In our work, we mainly studied two indexes that originally made use of disjoint windows: Duality Matching which was an improvement over the size and precision of the I-Adaptive index and rCMRI which was an improvement over the size and precision of the CMRI. Disjoint windows are shown in Figure 1.14.

1.4.4 Dimensionality Reduction Used Taxonomy

Another classification of time series indexing techniques is the type of dimensionality reduction techniques used. We distinguish three main types of dimensionality reduction techniques: spectral, piecewise aggregate and symbolic. All dimensionality reduction techniques including these should satisfy the GEMINI framework requirements.

Spectral

The whole matching solution proposed by Agrawal et al. [AFS95] originally used DFT as dimensionality reduction technique. I-Adaptive and MRI also used DFT as dimensionality reduction technique. However, this may be misleading, since the three indexes mentioned above are more or less dimensionality reduction agnostic, meaning that they could use either DWT, PAA or APCA. This may not always be the case, since as we will see the TS-tree is dependent on the use of SAX.

Piecewise Aggregate

A few indexes originally used piecewise aggregate methods, for instance: CMRI [KS07] which used APCA and rCMRI [NS] which used PAA. The reason why in rCMRI we used PAA rather than APCA (which is more precise) was that the goal of rCMRI was reducing the index size and PAA reduced series would have half the number of coefficients that APCA series have for the same number of segments. Although it can be argued that PAA may yield better precision than APCA with more approximating segments.

Symbolic

Although [KLLC03, KLLC07] report that traditional string based indexes such as suffix trees can be used to index SAX reduced time series, an important step in indexing SAX sequences for range queries, nearest neighbor queries and DTW queries was put forward by the TS-index [AKFA08]. The TS-index uses a MINDIST over

SAX sequence that can handle the above mentioned queries and that lower bounds the real distance, and hence satisfies the GEMINI framework requirement for SAX.

1.4.5 Spatial Data Structure Used Taxonomy

Another taxonomy would be to classify time series indexes according to the data structure they use, which could be R-trees or Grid files for example.

R-Tree Indexes

Most time series indexes discussed in this thesis make use of the ubiquitous R-tree. The I-Adaptive index, MRI, CMRI, rCMRI and Duality Matching all made use of R-trees originally. However, these indexes could easily swap their R-trees for any data structure in the R-tree family such as X-tree or R*-tree. Furthermore, in general, they could make use of any spatial data structure which supports range queries and nearest neighbor queries.

Grid File Indexes

The use of Grid files makes querying very efficient since we are dealing with a hash-based solution. The only index that originally makes use of Grid files that we will discuss in this thesis is Statstream. Statstream is suitable for very large datasets and very large streams where read-once conditions apply.

1.4.6 High Level Description of Some Time Series indexes

I-Adaptive

The I-Adaptive index was proposed by [FMR94] as an efficient solution for subsequence matching. A family of other indexes using the I-Adaptive index were proposed and obtained by changing only the dimensionality reduction technique or the spatial data structure. Here is the general algorithm:

1. Compute the sliding windows of length n for each series.
2. Reduce the dimensionality of each sliding window from n to N where N is much less than n, denoted $N << n$ using DFT for example.
3. Insert each reduced window in a spatial data structure such as the R-tree

A set of search operations are then possible over the R-tree such as range queries, nearest neighbor queries or correlation queries. While this solution can speed up considerably many data mining operations such as clustering and classification, it is only suitable for fixed length queries. When dealing with variable length queries, the solution degrades. When the length q of query Q is equal to the window size w, querying is straightforward. When $q \ mod \ w = 0$ and $q > w$, the following property [FMR94] holds for querying a sequence S decomposed into a number of disjoint sequences S_i's with a query Q decomposed into a number of disjoint Q_i's:

$$D(Q,S) \leq \epsilon \Rightarrow \vee_{i=1}^{p} D(s_i, q_i) \leq \epsilon/\sqrt{p} \tag{1.16}$$

Finally, when $q \ mod \ w \neq 0$ and $w < q < 2w$, the following [FMR94] holds for subsequences $S[i:j]$ of sequence S and $Q[i:j]$ of query Q:

$$D(Q,S) \leq \epsilon \Rightarrow D(S[i:j], Q[i:j] \leq \epsilon) \tag{1.17}$$

Duality Matching

As can be seen, the size of the previous index grows rapidly since we index every sliding window of a time sequence. Hence, for every value added to a time series, a d-dimensional point is added to the index. For very large datasets, this may be a problem. Duality Matching was developed by Moon et al. to address this issue [LMW00, LMW01, LMW02]. Using Duality Matching, we do not index sliding windows anymore but disjoint windows. This improves the index size by a factor of 1/w where w is the window size. For the query, sliding windows are used instead to segment the query such that each sliding window coming from the query sequence is used to query the disjoint window index. Using sliding windows for the query sequence and disjoint windows for the index is enough to solve the subsequence matching problem.

Duality matching and the I-Adaptive indexes are particular cases of a generalized matching framework [LMW02] where only the overlap between disjoint/sliding windows is different. For a range query, each sliding window obtained from the query sequence is used to query the index of disjoint windows such that the offset of a matching subsequence in a sequence S is given by:

$$offset = dw\_offset - qw\_offset + 1 \tag{1.18}$$

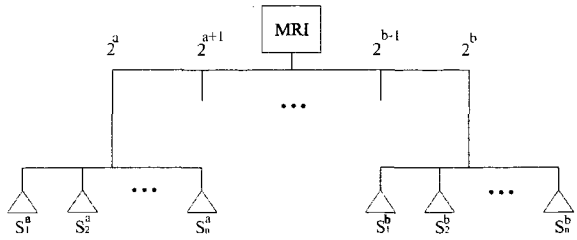

Figure 1.15: MRI index structure.

where dw refers to disjoint windows and qw refers to query sliding windows. A proof of the correctness of this scheme can be found in [LMW00, LMW01]. Values smaller or equal to zero and duplicates are discarded. Hence, the number of required queries is $q-w+1$ whereas in I-Adaptive only one query is required. However, the index size has decreased by a factor of $1/w$. An enhanced dual matching algorithm is described in [LMW00, LMW01] which requires only one query while being less precise than the basic algorithm.

In duality matching, there is a relation defined in [LMW00, LMW01] between the minimum query length, $Min(q)$, and the maximum window size, $Max(w)$, used to index the time series:

$$Max(w) = \left\lfloor \frac{Min(q)+1}{2} \right\rfloor \tag{1.19}$$

This formula is important when determining the optimal window size of a multi-resolution index using duality matching.

MRI

The Multi-Resolution Index was proposed by [KS04] to support variable length subsequence matching. The idea is to store different I-Adaptive at different resolutions using particular search algorithms for range or nearest neighbor queries. Here is an outline of the MRI index construction algorithm as proposed in [KS04]:

1. Compute the sliding windows of length 2^n of each series, where n ranges from a to b (a and b are called the stored resolutions of the index).

2. Reduce the dimensionality of each sliding window as for the I-Adaptive index.

3. Insert each reduced window in their respective t^{th} I-Adaptive index within resolution 2^n such that there will be $t(b-a)$ indexes in total and the t^{th} index corresponds to the t^{th} time sequence in the database.

Querying an MRI index is actually interesting. The query is divided into different powers of 2 length segments. Furthermore, as for range queries further pruning can be done, as the query advances. An MRI is shown in Figure 1.15.

CMRI

The Compact Multi-Resolution index (CMRI) was proposed by [KS07] as an improvement in the size and precision of the MRI and is shown in Figure 1.16. By combining all the I-Adaptive indexes used at any given resolution, CMRI achieves better size and performance. Here is an outline of the CMRI construction algorithm, which mainly differs with MRI in step 3:

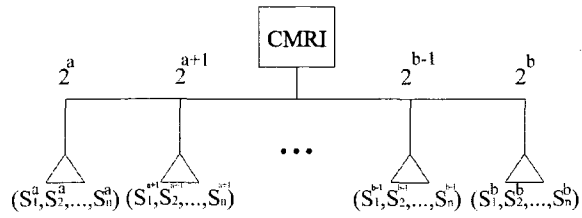

Figure 1.16: CMRI structure.

1. Compute the sliding windows of length 2^n of each series such that n ranges from a to b (a and b are called the stored resolutions of the index).

2. Reduce the dimensionality of each sliding window as for the I-Adaptive index.

3. Insert each reduced window in their respective I-Adaptive index of resolution 2^n such that there will be $b - a$ indexes.

TS-Index

The TS-index developed in [AKFA08] is the most recent of all indexes discussed here. It is actually an indexing solution which makes use of SAX as dimensionality reduction and supports similarity queries such as range and nearest neighbor queries and Dynamic Time Warping queries. The basic data structure it uses is the TS-tree, which is a variant of a B+-tree adapted to SAX data. Partitioning and ordering of the multidimensional SAX data is required in order for pruning to occur at query time.

The authors of the TS-index actually developed different MINDISTs in order to perform different similarity queries. However, they are quite involved and will not

be discussed here for the moment. The original TS-index is a single resolution index and can be combined into multi-resolution techniques in order to improve precision.

Statstream

Statstream was proposed in [SZ02, SZ04] for detection of correlated patterns under high performance requirements and very large volumes of time series. Here correlation means a score using Pearson's product moment coefficient. Shasha and Zhu [SZ02] prove that the product moment coefficient of two sequences is a function of the Euclidean distance between the normalization of those two sequences. The proposed solution uses a grid file, which can be described as a multi-dimensional hash, by which a target pattern is "hashed" to a cell in the grid file. Incoming patterns are hashed to a cell in the grid file. When a pattern is correlated, it is hashed to a cell in proximity to the target pattern. As can be seen, Statstream is appropriate when the target patterns are known in advance and for fixed correlation values. That is why it is a detector.

In this chapter, we have studied various classification of indexing techniques for time series data based on type of queries they support, data structures they use and dimensionality reduction techniques they employed. We next propose efficient correlation algorithms for bivariate and multivariate correlation, DTW correlation and rank order correlation, that all use for the moment CMRI as index.

Chapter 2

Fast Correlation Analysis on a Compact Multi-Resolution Index

There has been increasing interest for efficient techniques for fast correlation analysis of time series data in different application domains. We present three algorithms for (1) bivariate correlation queries, (2) multivariate correlation queries, and (3) correlation queries using dynamic time warping - a new correlation measure we proposed in [NS08a]. To support these algorithms, we propose a variant of the Compact Multi-Resolution Index (CMRI). In addition to conventional nearest neighbor and range queries supported by CMRI, the proposed algorithms compute all answers to user-defined, ad hoc and parametric correlation queries. The results of our experiments indicate a speed-up of two orders of magnitude over the brute force algorithm, and an order of magnitude improvement on average, while offering more functionalities than provided by other techniques such as StatStream [SZ02] and Spatial Cone Tree

[HKSZ03c].

In this chapter, we consider bivariate correlation queries that use Pearson's product-moment coefficient [Pea01]. Example of such a query would be: "*Find all stock market segments that are correlated to the U.S. interest rate measured between two given dates at correlation value exceeding r*". For multivariate correlation queries, an example would be: "*Find all stock market segments that are correlated to the U.S. interest rate measured, the price of housing and the U.S. exchange rate at correlation value exceeding r.*" Multivariate correlation search over indexed time series has not been studied before to the best of our knowledge. We will also introduce a new correlation measure based on Dynamic Time Warping (DTW) [MR81] and propose an algorithm which uses this measure for searching time series. We furthermore propose a solution for indexed rank order correlation queries. To support these queries, we extend algorithms proposed in [KS07] and present new ones.

2.1 Our Proposal

Our goal in this thesis is to develop algorithms for fast correlation analysis which are flexible for solving problems from different domains and with different requirements. A desired solution should support variable parameter queries, variable length queries, variable number of parameters, and different correlation measures. To support variable parameter queries, we will use indexing schemes that do not hard-code query parameters. To handle variable length queries, we propose a variant of the

CMRI [KS07] which also supports usual similarity queries such as range and nearest neighbor queries. Finally, to support variable number of parameters and different correlation measures, we propose four algorithms that use this variant of the CMRI index.

2.1.1 Bivariate Correlation Queries

A frequently used measure for bivariate correlation is the Pearson's product-moment coefficient [Pea01], which we use in our correlation queries. Using PAA as dimensionality reduction and a variation of CMRI as indexing scheme, we will show that bivariate correlation queries on PAA feature spaces can be done without false dismissals. Given the following mapping from bivariate correlation to Euclidean distance D [SZ04], it holds that:

$$D^2(\tilde{x}, \tilde{y}) = 2w(1 - R(x, y)) \tag{2.1}$$

Where w is the window size, $R(x, y)$ is the Pearson product-moment correlation coefficient between time series x and y and $\tilde{\cdot}$ is the normalization operator. We note that the following condition holds:

$$R(X, Y) \geq 1 - \epsilon^2 \Rightarrow D^2_{LB}(\tilde{X}, \tilde{Y}) \leq 2w\epsilon^2 \tag{2.2}$$

where D_{LB} is the lower bounding distance measure in feature space and capitalized variables X and Y denote PAA reduced series obtained from X and Y respectively. Note that D_{LB} is given by the following [CKMP01a]:

$$D_{LB}(X, Y) = (\frac{n}{N} \sum_{i=1}^{N} (X_i - Yi)^2)^{\frac{1}{2}} \tag{2.3}$$

Condition 2.2 indicates that we can do bivariate correlation analysis without modifying drastically the CMRI index and PAA algorithm. For queries where correlation is negative, we have the following condition:

$$R(x,y) \leq 1-\epsilon^2 \Rightarrow D_{LB}^2(-\tilde{X},\tilde{Y}) \leq 2w\epsilon^2 \tag{2.4}$$

Algorithm 1 shows steps of performing bivariate correlation queries on a compact multi-resolution index. This algorithm is a slight modification from [KS07]: the range variable e_0 is set as a function of the correlation parameter r.

Algorithm 1 Bivariate Correlation Algorithm

Input: A correlation parameter r, a query window q, the root of the i^{th} index of the CMRI denoted $root_i$, the window size w, and the number c of R-tree indexes in the CMRI.

Output: The match results *results*.

1: p_i = PARTITION(q)

2: $e_0 = \sqrt{2w(1-r)}$

3: **for** $i = 0$ to c **do**

4: $\quad results$ = RANGE_SEARCH(root, e_i, p_i)

5: $\quad e_{i+1} = max_{B \in results}(\sqrt{e_i^2 - d(q_i,B)^2})$

6: **end for**

7: results = POSTPROCESS(results)

2.1.2 Dynamic Correlation Queries over Non-Normalized Time Series

It is often required to store a non-normalized feature space since standard complex distance queries are performed on such spaces. Although we want to keep our feature and value spaces non-normalized, the bivariate correlation defined earlier requires that the feature space on which we pose queries to be normalized. Here we show that normalization can be done at query time when using PAA. For this, we apply at query time, the following transformation to the non-normalized feature space of dimension N of a value space of dimension n:

$$\tilde{X} = -\frac{n}{N}\mu + \frac{N}{\sigma n} \sum_{j=\frac{n}{N}(i-1)+1}^{\frac{n}{N}i} x_j \tag{2.5}$$

where i ranges from 1 to N. We can simply add "digests" to our data structure, as proposed in [SZ04], for the mean and standard deviation. The digest kept at every node for the variance σ and the average μ can be propagated recursively to containing nodes as follow:

$$DIGEST_{LOW} = \{max_{i \in child}(\mu_i), max_{i \in child}(\sigma_i)\} \tag{2.6}$$

$$DIGEST_{HIGH} = \{min_{i \in child}(\mu_i), min_{i \in child}(\sigma_i)\} \tag{2.7}$$

The subscripts LOW and HIGH denote the low and high value of the containing Minimum Bounding Rectangle (MBR) used in CMRI. Note that this will no longer be possible when using an rCMRI index.

2.1.3 Multivariate Correlation Queries

Our solution for multivariate correlation queries is implemented on a standard R-tree [Gut84]; we could also use R*-trees [BKSS90], R^+-trees [FSR87], or X-trees [BKK96] without modification to the algorithm. We focus on the 2D case where we have two query time series such as: "*Find all stock market segments that are correlated to the U.S. interest rate measured, the price of housing and the U.S. exchange rate at correlation value exceeding r*". Our proposed algorithm considers the 2D case, however, it can be extended to consider more than 2 time series.

2.1.4 Segmentation of the Solution Space

From the quadratic form definition of multivariate correlation, we can describe a solution space whose boundary is a hyper-ellipsoid. Figure 2.1 shows an example of the 2D case. Solutions to a multivariate correlation query where correlation needs to be greater than or equal to a user-defined variable r lie on the boundary or outside the boundary.

While proposed for a different purpose, we adapted an algorithm from [KR04] to segment the upper and lower parts of the solution space. Hence, after sampling points of the solution space boundary, we generate a lower upper-bounding segmentation U_i and an upper lower-bounding segmentation L_i using the following formulas for the 2D solution space (see Figure 2.1):

$$U_i = max(x_{\frac{n}{N}(i-1)+1}, ..., x_{\frac{n}{N}i}) \tag{2.8}$$

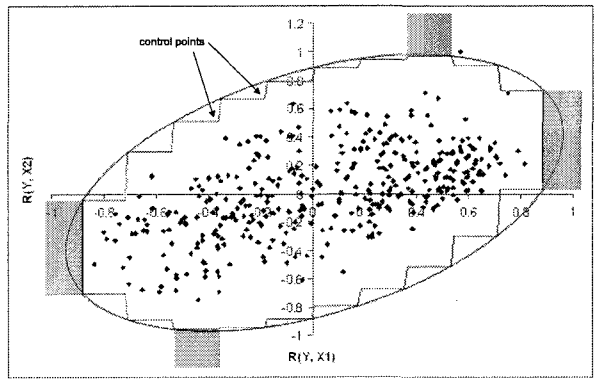

Figure 2.1: Segmentation of solution space with $r = 0.97$ and grayed out rules.

$$L_i = min(x_{\frac{n}{N}(i-1)+1}, ..., x_{\frac{n}{N}i}) \tag{2.9}$$

In general, for an n-dimensional solution space, a set {U,L} is required for each unordered pair of coordinate axis. From this segmentation of the solution space, a set of *control points* is obtained, each of which includes the vertices of the segmentation. A set of rules $\{R_i\}$ can be derived from the inner control points c_i with coordinates $c_{(x)i}$ and $c_{(y)i}$ of the segmentation for the 2D case, but can be generalized to n-dimensions. Algorithm 2 below shows the derivation of $\{R_i\}$ for the 2D case.

The number of rules needed is $O(c)$, where c is the number of control points. These rules can be implemented simply as IF-ELSE clauses. From Figure 2.1, we can see that if a point in solution space is a valid solution, then it must pass at least one of the checks in $\{R_i\}$, even though the converse may not be true. An "upper-bounding in absolute value" property guarantees that no false dismissals will incur when using $\{R_i\}$ to prune the solution space as we will see in Section 3.6. Notice that the precision of our pruning depends on the number of control points chosen to

Algorithm 2 Generate Rules

Input: Lower upper-bounding control point in U and upper lower-bounding control point in L.

Output: A set of *rules*.

1: $rules = \{\}$

2: **for** every cp_i in L from $i = 0$ **do**

3: **repeat**

4: ADD_RULE : $\{x \leq cp_{(x)i}\ AND\ y \geq cp_{(y)i}\}$ TO *rules*

5: **until** $cp_{(y)i}$ *reaches* $max(L_i)$

6: **for** $i =$ *remaining control points* **do**

7: ADD_RULE : $\{x \geq cp_{(x)i}\ AND\ y \geq cp_{(y)i}\}$ TO *rules*

8: **end for**

9: **end for**

10: **for** every cp_i in U from $i = 0$ **do**

11: **repeat**

12: ADD_RULE : $\{x \leq cp_{(x)i}\ AND\ y \leq cp_{(y)i}\}$ TO *rules*

13: **until** $cp_{(y)i}$ *reaches* $min(U_i)$

14: **for** $i =$ *remaining control points* **do**

15: ADD_RULE : $\{x \geq cp_{(x)i}\ AND\ y \leq cp_{(y)i}\}$ TO *rules*

16: **end for**

17: **end for**

18: ADD_BOUNDARY_RULES() #see grayed areas in Fig.1

19: **return** *rules*

segment the solution space.

2.1.5 Transformation of the Solution Space

Transformation of distance measures for quadratic form distance measures was first introduced in [KSUY01]. We have adapted those techniques in our context. Once a set P_i of points is found from the segmentation of the solution space, we determine whether they are inside or outside the boundary. Since the boundary is a rotated quadratic form hyper-ellipsoid, one way to decide this is to compute a transformation matrix composed of two linear transformation matrices. The first matrix rotates the hyper-ellipsoid such that its major and minor axes are orthogonal to the coordinate axes. It can be obtained, according to the Principal Axes Theorem, from the normalized eigenvectors of the correlation matrix R_{x_i,x_j}. In the 2D case, this matrix will always be the same, given that R_{x_i,x_j} is a symmetric matrix with elements on its diagonal that are equal, as follows:

$$M_{R_1} = \begin{bmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{bmatrix}, \ \alpha = \frac{\pi}{4}$$

The second linear matrix transforms the hyper-ellipsoid into a hyper-sphere, as follows.

$$M_{R_2} = \begin{cases} 1/m_i & i = j, \\ 0 & i \neq j, \end{cases}$$

where the m_i's are the value of the radius of the major/minor axes parallel to the hyper-dimensional coordinate system. The final transformation matrix is given as:

$$M_R = M_{R_1} \cdot M_{R_2} \tag{2.10}$$

Each point of P_i is multiplied by the transformation matrix, and a point is a solution if and only if the norm of its transformed point is larger than the radius of the hyper-sphere (usually set to 0.5), i.e:

$$|M_R P_i| > radius \tag{2.11}$$

This condition is a second pruning step which yields a set of points that are exact solution to the multivariate correlation query when no dimensionality reduction is used and which may yield false positives if one is used.

2.1.6 Cosine Definition of Bivariate Correlation

To perform the pruning process described previously, we map the value space or feature space to bivariate correlation space since the solution space is described in terms of bivariate correlation variables. If we use an index over feature space, we must also ensure that the mapping from feature space to bivariate correlation space is such that the feature space correlation value upper bounds the real bivariate correlation in absolute value. The cosine definition of bivariate correlation satisfies this condition and corresponds to the cosine of the angle between the two target vectors x and y. It is defined as follows:

$$R(x, y) = cos(\tilde{x}, \tilde{y}) = \frac{\tilde{x} \cdot \tilde{y}}{|\tilde{x}||\tilde{y}|} \tag{2.12}$$

Hence, we have the following:

$$|cos(X, Y)| \geq |cos(x, y)| \tag{2.13}$$

where X and Y are the PAA feature space sequences, and x and y are the corresponding sequences in the value space.

2.1.7 Querying over an R-Tree

To implement the multivariate correlation algorithm, we need to be able to apply what we described previously to an index such as the R-tree, for which we define an algorithm to prune the solution space using the segmentation rules $\{R_i\}$ over Minimum Bounding Rectangles (MBR) of the R-tree. To achieve this, we propose the concept of spanning angles which corresponds to the minimum and maximum angles α_{min} and α_{max} that an MBR can have with a query point Q_i. To speed up the computation, in order to prune MBR's that do not contain points outside of the solution space segmentation given in Section 3.4, we use the following facts, adapted from [HKSZ03c]:

$$R(x, y) > r \Rightarrow cos^{-1}(cos(\tilde{x}, \tilde{y})) < cos^{-1}(r) \tag{2.14}$$

$$R(x, y) > -r \Rightarrow cos^{-1}(cos(\tilde{x}, \tilde{y})) < cos^{-1}(-r) \tag{2.15}$$

$$R(x, y) < -r \Rightarrow cos^{-1}(cos(\tilde{x}, \tilde{y})) > \pi - cos^{-1}(r) \tag{2.16}$$

$$R(x, y) < r \Rightarrow cos^{-1}(cos(\tilde{x}, \tilde{y})) < \pi - cos^{-1}(-r) \tag{2.17}$$

Hence, if a rule R_i generated from the segmented solution space is of the form "r > 0.9", this rule can be mapped to "$\alpha_{min} < cos^{-1}(0.9)$." If a rule R_j generated from the segmented solution space is of the form "r < -0.9", this rule can be mapped to "$\alpha_{max} > \pi - cos^{-1}(0.9)$." In general, equations (16) and (17) above use α_{min}, and equations (18) and (19) use α_{max}. This approach can be extended to multiple correlation parameters. We next summarize our algorithm for multivariate correlation queries.

2.1.8 Multivariate Correlation Queries Algorithm

Algorithm 3 below shows steps of performing the multivariate correlation queries. It first obtains the solutions space transformation matrix described in Section 3.5 from the quadratic form matrix of equation (9). This is possible because we know the x_i values at query time. It then segments the solutions space given by the quadratic form matrix into upper-bounding and lower-bounding segments for the 2D case from equations (10) and (11). Finally, through tree traversal operations, we obtain for each MBR the minimum and maximum spanning angles to a query point and compare them against the set of rules $\{R_i\}$ obtained from segmentation of the solution space.

Algorithm 3 Multivariate Correlation Algorithm

Input: A correlation parameter r, query windows q_i, the *root* of an R-tree, the reduced data dimension d, the window size w and a solution space matrix R

Output: A answer set of *results*

1: p_i = PAA(q_i)

2: M_{R_1}=ROTATION_MATRIX(q_i,r)

3: M_{R_2}=SKEW_MATRIX(q_i)

4: $M_R = M_{R_1} \cdot M_{R_2}$

5: U = UPPER_SEGMENTATION(q_i)

6: L = LOWER_SEGMENTATION(q_i)

7: rules = GENERATE_RULES(U,L)

8: temp_results = TRAVERSE_TREE(root, rules, p_i)

9: **for** every *result in temp_result* **do**

10: **if** $|result \cdot M_R| >$ radius **then**

11: ADD *result* TO *results*

12: **end if**

13: **end for**

14: *results* = POSTPROCESS(*results*)

Algorithm 4 Adjunct method: ROTATION_MATRIX(q_i,r)

Input: Query windows q_i and a correlation parameter r

Output: A rotation matrix

1: COMPUTE R_{x_i,x_j} FROM q_i and r

2: COMPUTE M_{R_1} from *eigenvectors* of R_{x_i,x_j}

Algorithm 5 Adjunct method: SKEW_MATRIX(q_i,r)

Input: Query windows q_i and a correlation parameter r

Output: A skewness matrix

1: COMPUTE R_{x_i,x_j} FROM q_i and r

2: COMPUTE M_{R_2} from *major* and *minor axis*

Algorithm 6 Adjunct method: UPPER_SEGMENTATION(q_i)

Input: Query windows q_i

Output: A set of upper segmenting values

1: x_i = sample points of the lower-bounding solution space

2: $U_i = max(x_{\frac{n}{N}(i-1)+1}, ..., x_{\frac{n}{N}i})$

Algorithm 7 Adjunct method: LOWER_SEGMENTATION(q_i)

Input: Query windows q_i

Output: A set of lower segmenting values

1: x_i = sample points of the upper-bounding solution space

2: $L_i = min(x_{\frac{n}{N}(i-1)+1}, ..., x_{\frac{n}{N}i})$

Algorithm 8 Adjunct method: TRAVERSE_TREE(root, rules, p_i)

Input: An R-tree root, a set of rules, PAA points

Output: Results

```
1: if !IS_LEAF(root) then
2:   for every node in root do
3:     (α_min, α_max)=GET_MBR_SPANNING_ANGLES()
4:     if CHECK_RULES(rules, α_min, α_max) then
5:       TRAVERSE_TREE(node,rules,p_i)
6:     end if
7:   end for
8: else
9:   if COMPUTE_CORRELATION(p_i, node) then
10:    ADD node TO results
11:  end if
12: end if
13: return results
```

Algorithm 9 Adjunct method: CHECK_RULES(rules, $\alpha_{min}, \alpha_{max}$)
Input: MBR bounding points α_{min} and α_{max}

Output: A rule

1: **if** rule is of type r>k **then**

2: USE $\alpha_{min} < cos^{-1}(k)$

3: **end if**

4: **if** rule is of type r<k **then**

5: USE $\alpha_{max} > \pi - cos^{-1}(k)$

6: **end if**

2.1.9 From 1-dimensional to n-dimensional Solution Spaces

As the quadratic form equation for multivariate correlation is a generalization of the bivariate correlation equation, the algorithm for multivariate correlation search is a generalization of the bivariate correlation search algorithm. For 1-dimensional solution spaces (i.e., bivariate correlation), the quadratic form equation for multivariate correlation reduces to:

$$R(x_1, y)^2 = [R_{x_1,y}] \cdot [R_{x_1,x_1}] \cdot [R_{x_1,y}]^T = R^2_{x_1,y} \geq r \tag{2.18}$$

Graphically, a 1-dimensional solution space is a line. Furthermore, the rules generated for such a solution space is directly given by the previous expression $R(y, x_1)^2 \geq r$. In general, the number of terms in a rule for an n-dimensional solution space is n and the number of rule types (rules having the same inequality operator for each term in

the rule) is k, where k is the number of quadrants in the solution space.

2.1.10 Dynamically Time Warped Correlation

In chapter 1, we proposed a correlation measure based on DTW. Here, we show how to answer DTWC queries using CMRI. Let x and y be time series. We know the following holds between the DTW distance and the Euclidean distance D:

$$DTW(x, y) \leq D(x, y) \tag{2.19}$$

The DTW distance is the largest when no warping occurs or, in other words, when the warping path is a straight line. When this occurs, it holds that $DTW(x, y) = D(x, y)$. If the warping path deviates, this means that it has found a path where $DTW(x_i, y_i)$ is smaller than $D(x_i, y_i)$, as can be seen from the DTW distance recurrence relation.

Since the DTW distance is always smaller or equal to the Euclidean distance, the DTWC will always be greater than or equal to the standard correlation. That is:

$$|DTWC(x, y)| \geq |R(x, y)| \tag{2.20}$$

This property means that correlation queries using DTWC might return time series that would not have been returned by correlation queries using Pearson's coefficient. As correlation queries over sliding windows were robust face to translated datasets, DTWC is robust faced to skewed datasets. Note that the exact value of the DTWC can be computed from the standard correlation between the DTW expansions of two time series. Algorithm 10 shows the steps for performing DTWC queries. The

Algorithm 10 DTW Correlation Algorithm

Input: A correlation value r, a query window q, the root of the i^{th} index of the CMRI $root_i$, the window size w and the number of R-tree indexes c in the CMRI.

Output: The results *results*.

1: p_i = PARTITION(q)

2: $e_0 = \sqrt{2w(1-r)}$

3: **for** $i = 0$ *to* c **do**

4: $results$ = DTW_RANGE_SEARCH(root, e_i, p_i)

5: $e_{i+1} = max_{B \in results}(\sqrt{e_i^2 - d(q_i, B)^2})$

6: **end for**

7: results = POSTPROCESS(results)

basic idea behind DTW range queries is to find an upper bound and a lower bound to a query sequence from which we calculate a measure called LB_KEOGH, which lower bounds DTW. For PAA sequences, an equivalent lower bounding measure called LB_PAA exists. The proof of correctness of this algorithm (i.e., no false dismissals will occur) relies on the correctness of the DTW range query, which is shown in [KR04]. We do not show details on DTW range queries and the reader can refer to [KR04].

2.1.11 Rank Order Correlation Queries

This section presents a polyhedral indexing technique for rank order correlation queries for time series data. Rank order correlation is a nonparametric and non-

linear measure that can detect nonlinear monotonically increasing or decreasing relationships between sequences. Rank order correlation has been extensively used in cases when the nature of the problem to be solved is non-parametric or when the relationship between two variables is nonlinear and monotonic. In such cases, linear correlation measures, such as the product moment coefficient, are inadequate and fail to detect correlative relations. Furthermore, rank order correlation has an interesting geometrical interpretation that lends itself to indexing by spatial indexes, e.g. R-trees or R-tree based indexes such as the Multi-Resolution Index (MRI) or the Compact Multi-Resolution Index (CMRI). Our technique provides signficant improvement over sequential scan, the only other solution at the moment, by one to two orders of magnitude in some cases.

Pearson's product moment correlation coefficient has been successfully indexed in [HKSZ03a, HKSZ03b, HKSZ03c, NS08a, SZ02, SZ04]. However, to the best of our knowledge, indexing time series databases for rank order correlation queries has not been studied before.

Our approach to address rank order correlation queries is to map the time series data onto the polyhedral solution space of rank order correlation, hence the term "polyhedral index", and make use of current indexing techniques for variable length queries such as MRI [KS04] and CMRI [KS07]) to index the mapped spatial data. The proposed solution makes use of the geometric model of rank correlation described in [CS83, Sch79] in order to build a polyhedral R-tree based index which will offer significant speed-up over sequential scans which are the only other alternative. The

aforementioned geometric model makes it possible to map d-dimensional query and data points into a polyhedral solution space of size O(d!) which can then be handled by spatial indexes [Gut84, BKSS90, FSR87, BKK96]. The size of the polyhedral solution space does not affect the size of the polyhedral index, which is of size O(n) where n is the number of data points in the time series dataset. Hence, changes are made to the underlying insertion and index construction algorithms, and a new rank order correlation query algorithm is proposed, which itself is based on range queries.

In this section, we discuss the mathematical expression of rank order correlation and its properties. We also discuss a geometrical model of rank order correlation as described in [CS83, Sch79]. Finally, we discuss dimensionality reduction methods and spatial and multi-resolution indexes that can be used for time series data.

A Geometric Model of Rank Correlation

The rank order correlation of two sequences X and Y of length n can be projected onto a n-dimensional polyhedral with n! vertices [CS83, Sch79]. Let $X =< x_1, \ldots, x_n >$ and $Y =< y_1, \ldots, y_n >$, then for each sequence in value space, there exists a corresponding sequence in rank space $R_X =< r_{X,1}, \ldots, r_{X,n} >$ and $R_Y =< r_{Y,1}, \ldots, r_{Y,n} >$. Note that these rankings can be expressed as a set of pair-wise permutation from a null ranking vector $R_0 =< 1, 2, \ldots, n >$. Furthermore, we have that any two vertices R_X and R_Y are connected by a segment if and only if they are of the form given in [Sch79] where r and r+1 are an actual rank values that can be in non-adjacent

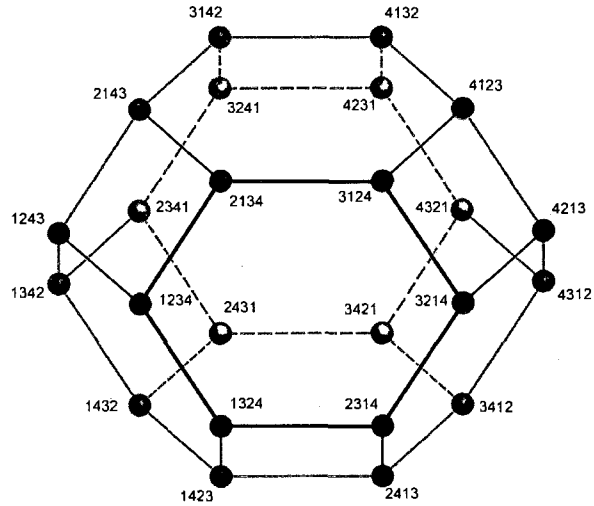

Figure 2.2: Polyhedral solution space for rank order correlation.

positions in the rank space transformation of the series:

$$R_X =< \ldots, r, \ldots, r+1, \ldots > \tag{2.21}$$

$$R_Y =< \ldots, r+1, \ldots, r, \ldots > \tag{2.22}$$

If we map the rank space to Euclidean space, we obtain an n-dimensional polyhedra with n! vertices, which is in fact an (n-1)-dimensional polyhedral rotated in the n^{th} dimension. Figure 2.2 adapted from [CS83, Sch79] shows the case where n = 4. The color of each vertex shows the 4^{th} dimension. Note that the length of each edge can be proven to be $\sqrt{2}$ [Sch79].

The geometrical model of rank order correlation presented here can be applied not only to Spearman's coefficient but also to other tests using l_2 norm, vector product and levels [CS83]. From this polyhedral model, we can effectively determine that the rank order correlation coefficient between X and Y as a function of the distance

between the rank space vectors of X and Y [Sch79]. In comparison, for other correlation measures such as Kendall's tau, the value of the coefficient becomes a function of the shortest path between two vertices of the polyhedral solution space, which also corresponds to the minimum number of swaps required to transform R_X into R_Y.

Polyhedral Transformation of Time Series

Before a time series data can be indexed, it needs to be transformed into its rank space equivalent. By performing this transformation to polyhedral space, we are effectively generating points in Euclidean space that can be indexed by a spatial data structure. For example, for a common sliding window of size 32, we could have a polyhedral space of 32! vertices of dimension 32. Since such high dimensionality does not practically perform well on R-tree based indexes, it is common practice to reduce the dimensionality of the data to 8 using a dimensionality reduction method such as PAA [FY00, CKMP01a], APCA [CKMP01b], DWT [CFY03] or DFT [FMR94]. This results in a polyhedral solution space has 8! = 40,320 vertices which is enough to guarantee that the transformed data in rank space will be spread out enough for distance based queries to be significant assuming that the data is more or less uniformly distributed across rank orders. Furthermore, let $R_{X_{PAA}} =< r_{X_{PAA},1}, \ldots, r_{X_{PAA},n} >$ and $R_{Y_{PAA}} =< r_{Y_{PAA},1}, \ldots, r_{Y_{PAA},n} >$ be the PAA representation of R_X and R_Y. We then have the following inequality from the lower bounding property of PAA [FY00, CKMP01a]:

$$D^2(R_X, R_Y) \geq D^2(R_{X_{PAA}}, R_{Y_{PAA}}) \tag{2.23}$$

This condition will ensure that no false negatives will occur during query processing although we may have false positives. Now we consider the case where ties may occur. From [KG90], we know that, when ties occur, the sum of ranks stays the same whereas the sum of rank squares is reduced by the following factor:

$$\Delta = \frac{1}{12}(u^3 - u) \tag{2.24}$$

If we expand the summation of the squared rank differences in the definition of rank correlation, we have the following:

$$\sum_{i=1}^{i=n}(r_{X,i} - r_{Y,i})^2 = \sum r_{X,i}^2 + \sum r_{y,i}^2 - 2\sum r_{X,i}r_{Y,i} \tag{2.25}$$

Hence, the sum of squares for r_X and r_Y is reduced by -U and -V, which is why the definition of tied rank correlation adjusts for U and V. Now we derive the conditions under which we can guarantee that PAA reduced measures will incur no false negatives. We know the following:

$$D^2(R_X, R_Y) \geq D^2(R_{X_{PAA}}, R_{Y_{PAA}}) \tag{2.26}$$

Considering the definition of the tied rank order correlation, we have the following:

$$U \geq 0, V \geq 0 \Rightarrow D^2(R_X, R_Y) + U + V \geq D^2(R_{X_{PAA}}, R_{Y_{PAA}}) \tag{2.27}$$

This condition ensures a lower bound for of distance measures over PAA rank space values which in turn guarantees no false negatives will occur at query time. Hence, distance measures can be done on feature space without error.

Mapping Rank Order Queries to Range Queries

Once the polyhedral feature space is obtained from the polyhedral rank space (which itself is obtained from value space) and properly indexed into an R-tree based index such as MRI [KS04] or CMRI [KS07], the only step left is to map the rank order correlation query to a range query. Rank order correlation is a function of the Euclidean distance between rank vectors. Highly correlated rank vectors are close to each other in polyhedral space, whereas uncorrelated rank vectors are far from each other in polyhedral space. Hence, it is possible to perform a range query that will retrieve the higly correlated sequences since, by the definition of rank order correlation, distance between rank vectors is itself a function of correlation. This expression of distance between rank vectors as a function of correlation is given in the following equation:

$$D^2 = \sum_{i=1}^{n} (r_{X,i} - r_{Y,i})^2 = \frac{(1-r)(n^3-n)}{6} \tag{2.28}$$

For any given user-defined parameter r, it is possible to define a distance based range query. This range query is as follows:

$$e \leq \sqrt{\frac{(1-r)(n^3-n)}{6}} \tag{2.29}$$

Next we give algorithms for insertion in a polyhedral index and for rank order correlation queries.

Rank Order Correlation Algorithms

In this section, Algorithm 11 describes insertion in an R-tree based index such as MRI or CMRI and Algorithm 12 describes the rank order correlation query based on range

queries. Basically, these algorithms transform the input time series data into rank polyhedral space. In our algorithms, we consider using a CMRI index. The insertion algorithms in an R-tree can be found in [Gut84], the PAA reduction reduction in [FY00, CKMP01a] and the range query on R-trees and CMRI in [KS07, MNPT06].

Algorithm 11 Insertion in CMRI Algorithm for Rank Order indexing

Input: A set T_i of CMRI R-trees, a set of sliding windows $\{s_{n-2+1}, ..., s_n\}$, CMRI R-tree resolution ranging from a to b.

Output: An updated CMRI index.

1: **for** i=a to b **do**

2: $< r_1, \ldots, r_{2^i} >_i = \mathsf{RANK}(< s_{n-2+1}, ..., s_n >)$

3: $< r_1, \ldots, r_d >_i = \mathsf{PAA}(< r_1, \ldots, r_{2^i} >_i)$

4: $\mathsf{INSERT}(< r_1, \ldots, r_d >_i)$ in T_i

5: **end for**

Algorithm 12 Rank Order Correlation Query Algorithm

Input: A correlation value r, a query window q, the root of the ith index of the CMRI $root_i$, the window size w and the number c of R-tree indexes c in the CMRI.

Output: The results *results*.

1: p_i = PARTITION(q)//

2: $e_0 = \sqrt{\frac{(1-r)(w^3-w)}{6}}$

3: **for** i=1 to c **do**

4: results = RANGE_SEARCH(root$_i$, e_i, p_i)

5: $e_{i+1} = max_{B \in results}(\sqrt{e_i^2 - d(q_i, B)^2})$

6: **end for**

7: results = POSTPROCESS(results)

Chapter 3

Reduced Compact Multi-Resolution Index

We present an improvement and extension of the Compact Multi-Resolution Index (CMRI) which supports variable length range and nearest neighbor queries. We call our index the reduced Compact Multi-Resolution Index (rCMRI). Central to the proposal is the concept of duality matching [LMW01]. Our improved and extended index combines both multi-resolution methods and duality matching into a single index. We study performance of rCMRI and compare it with CMRI and Duality Matching. While reducing the size of the CMRI by using duality matching, our proposal also improves the precision of duality matching by using multiple resolutions. We present an index construction algorithm for rCMRI, and also develop algorithms for range queries and nearest neighbor queries, in addition to linear and non-linear correlation queries. Using rCMRI, larger time series can be indexed efficiently since

Table 3.1: Size Comparisons Between CMRI and rCMRI

Index and Data Scenarios	**Per Day**	**Per Minute**
Raw Data	10MB	11GB
CMRI	580MB	875GB
rCMRI	18MB	27GB

more can be loaded into main memory and less information is required to index correctly the full set of subsequences. In general, rCMRI is smaller than the CMRI by a factor of $1/w$ where w is the window size, and decreases the number of false positives and has a slightly better query response time.

As illustration of index sizes, a time series database holding all end of day values of 6,500 stocks over a period of 1 year requires approximately 10 MB to store. A CMRI for that database, assuming a feature extraction method keeping 8 coefficients and 5 resolutions, would require approximately 580 MB. If the time series database stores values sampled at every minute, it would require roughly 11 GB. A CMRI for that database under the same assumptions would require roughly 875 GB, a case which cannot be handled efficiently by conventional systems. In contrast, the rCMRI index we propose here, assuming 8 coefficients and 5 resolutions, would require roughly 18 MB for an end of day sampling case and 27 GB for a per minute sampling case, both of which can be easily handled by a conventional system. Table 3.1 summarizes this information.

3.1 Our Proposal

In this section, we first explain the need for multi-resolution indexing schemes by showing how they can improve the precision of query results. We then present our rCMRI and explain its features and improvements, and discuss the optimal window size at each resolution used by rCMRI. Finally, we discuss various algorithms for index construction and insertion, for range queries, for nearest neighbor queries and for linear and non-linear correlation queries.

3.1.1 Motivations for Multi-Resolution

Multi-resolution is a concept that aims at increasing the precision of the answer set returned by an indexed query over indexed data. By storing data at different resolutions (often powers of 2), query sequences (also decomposed in powers of 2) provide a closer match to the indexed subsequences. In comparison, the I-Adaptive index relies on some heuristics in order to deal with variable length queries, both of which incur precision loss when used. Duality matching relies on a minimum query length, $Min(q)$, from which we can derive a maximum window length, $Max(w)$, both of which need to be tightly matched in order to offer optimal performance. Figure 3.1 shows the number of subsequences returned by duality matching when a query length is 64 for a window size w varying from 2 to 32. As the window size decreases, the number of subsequences returned increases which includes many false positives since the query at window size 32 incurs no false negatives. The precision penalty of not

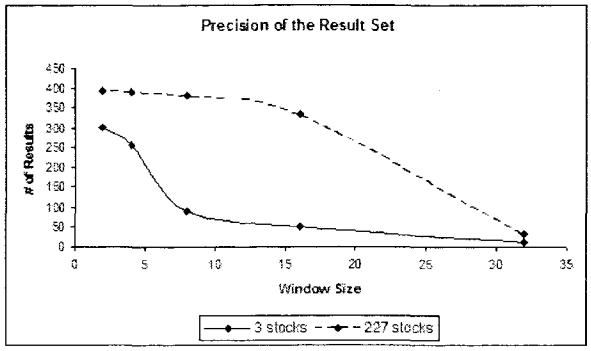

Figure 3.1: Number of results retrieved as a function of the window size for query length = 64 and range query parameter $\epsilon = 3.5$.

using optimal window sizes quickly grows to an order of magnitude. This justifies the use of multi-resolution which is an optimization method for when the query length is not known in advance. We have used real life stock market data sampled every day for a period of 400 days to perform this test.

Furthermore, in Figure 3.1, we have only shown the number of valid subsequences returned by duality matching (i.e. offsets that are greater than 0 and not duplicated). However, it may be interesting to note that for 3 stocks (approximately 1000 different subsequences), the total number of subsequences (valid or invalid) returned was 64 for a window size $w = 32$, 121 for $w = 16$, 273 for $w = 8$, 616 for $w = 4$, and 1611 for $w = 2$.

The proportion of valid subsequences returned to different subsequences in the input time series database we noted varied from 1% to 30% for 3 stocks, whereas the proportion of valid and invalid subsequences returned to the same input varied from 1% to 160%. Even though invalid subsequences are pruned out at query processing

time and do not incur I/O cost, when the number of such subsequences is large, it increases overall processing time to identify and remove them out from the result set. These observations indicate that large window sizes incur less performance penality due to invalid subsequences. This is a second reason for using multi-resolution as a solution feature in our work.

3.1.2 rCMRI

Similar to CMRI, the rCMRI index stores all sequences for a given resolution in a single R-tree. However, instead of storing sliding windows (Figure 1.13) over each sequence, it stores disjoint windows (Figure 1.14). This reduces the size of the index by a factor of $1/w$, where w is the window size. We will discuss below the optimal window size for each resolution. The size of rCMRI will always be smaller than the original CMRI if the series is longer than the window size.

To furthermore improve on the size of the original CMRI, rCMRI makes use of PAA as dimensionlity reduction rather than APCA, hence decreasing the index size by half. Note that for streaming data, rCMRI only inserts a feature space value in the index when w values have elapsed since the last insertion since we are dealing with disjoint windows. The values between two insertions are accounted for in queries through Equation 1.17.

Hence, rCMRI stores as many as $b-a$ trees at resolution $2^a, 2^{a+1}, ..., 2^{b-1}, 2^b$. Each resolution contains disjoint windows of size w whose dimensionality has been reduced using PAA (typically $N = 8$). Here we discuss optimal size of w. From Equation

1.19, we get a relationship between the maximum window size and the minimum query length, a proof of which is given in [LMW01]. From Figure 3.1, we can see that the precision of the result set decreases with the window size for a given query length q. This is referred to as the *window size effect* [LMW01]. From Equation 4, we see that the largest possible value of w for a given query length q is obtained when:

$$w = \left\lfloor \frac{q+1}{2} \right\rfloor \tag{3.1}$$

In other words, the optimal size of w is roughly half of q. This leads us to the following: rCMRI will store at resolution 2^i, disjoint windows of size $w = 2^i/2$. This improves the storage requirement of rCMRI by a factor of $1/w$ over the size of the original CMRI but is larger than the space required by duality matching by a factor of $b - a$, the number of resolution used (typically around 5), which aimes at improving over precision of queries while keeping the size of the index manageable for large datasets.

3.1.3 Index Construction and Insertion Algorithms

In this section, we present our index construction algorithm for rMCRI and for insertion operations. As discussed previously, rCMRI resembles closely the original CMRI with the difference that it stores disjoint windows rather than sliding windows, hence reducing the index size by a factor of $1/w$. Algorithm 13 describes the index construction algorithm.

Algorithm 14 below describes insertion operation into an rCMRI index for stream-

Algorithm 13 Construction of rCMRI

Input: R-tree resolutions $\{T_a \ldots T_b\}$, sequence $S_i = \langle s_1, s2, \ldots, s_k \rangle$, a reduced dimension d.

Output: The rCMRI index.

1: **for** $i = a$ *to* b **do**

2: **for** $j = 1$ *to* k, $j = j + (2^{i-1})$ **do**

3: $\langle r_1, \ldots, r_d \rangle = \mathsf{PAA}(\langle \mathsf{s_j}, \ldots, \mathsf{s_{j+(2^i/2)}} \rangle)$

4: $\mathsf{INSERT}(\langle \mathsf{r_1}, \ldots, \mathsf{r_d} \rangle \text{ in } \mathsf{T_i})$

5: **end for**

6: **end for**

ing values of a sequence S. Different sequences S_i may require different number of iterations to terminate. The main feature of this algorithm is that it inserts only disjoint windows. Insertion occurs only when $(k \, mod \, 2^i)/2 = 0$, for $k \geq 1$ for each sequence. This releaves the system of constant insertion as in CMRI. Interestingly, note that the streaming values between two insertions are still accounted for at query time due to Duality Matching correctness axioms which guarantee that if a sequence is within ϵ distance of a query, any subsequence of the subsequence and the query will also be within ϵ. The INSERT algorithm used is the one used for standard R-tree insertion method (e.g. [MNPT06]).

Note that there is a significant improvement between index construction and insertion algorithms for the original CMRI compared to the one proposed for rCMRI since insertion into the index now only occurs every $w = 2^{i-1}$ times a value is saved in

Algorithm 14 Insertion in rCMRI

Input: A set T_i of R-trees, streaming values s_k for $k \geq 1, \ldots,$ R-tree resolutions ranging from a to b.

Output: Updated rCMRI index.

1: **for** $i = a$ *to* b **do**
2: **for** every streaming k **do**
3: **if** $k \, mod \, 2^{i-1} = 0$ **then**
4: $\langle r_1, \ldots, r_d \rangle = \mathsf{PAA}(\langle \mathsf{s_{k-(2^{i-1})+1}}, \ldots, \mathsf{s_k} \rangle)$
5: $\mathsf{INSERT}(\langle \mathsf{r_1}, \ldots, \mathsf{r_d} \rangle \text{ in } \mathsf{T_i})$
6: **end if**
7: **end for**
8: **end for**

the database for a given resolution i in addition to an index size reduced by a factor of $1/w$. This makes rCMRI more desirable for dealing with streaming values.

3.1.4 Range Queries

Range queries over subsequences are common on time series databases. They retrieve all subsequences of a set of sequences S_i such that $D(S_i, Q) \leq \epsilon$ where Q is a query, D is a distance function, and ϵ is a range query parameter. For range queries over an rCMRI index, we first divide the query in powers of 2, divide the subqueries obtained into $k = n - w + 1$ sliding windows where n is the number of values in the given sequence and then construct a range query for each query sliding window. The range query in question is similar to the one proposed for the original CMRI which itself is based on a standard range query algorithm [MNPT06]. An additional parameter p is required for range queries over rCMRI which gives the minimal number of included windows for a given query length q as defined in general by [LMW01]:

$$p = \left\lfloor \frac{q+1}{w} \right\rfloor - 1 \tag{3.2}$$

For rCMRI, this parameter is always equal to 1 since $q = 2w$. Algorithm 15 describes the range query algorithm. Note that multiple range queries are required for rCMRI.

As in any range query over feature space, a post-processing phase is required to eliminate false positives. An enhanced version of a duality matching range query can be found in [LMW01] where a single range query is performed rather than $n - w + 1$ queries. That algorithm can be adapted as we adapted the basic algorithm to a

Algorithm 15 Range query over rCMRI

Input: A range parameter ϵ, a query Q of length q, the root of the i^{th} index of the CMRI $root_i$, the window size w and the number of R-tree indexes c in the CMRI.

Output: The answer *results*.

1: Q_i = PARTITION(Q)

2: $e_1 = \epsilon/\sqrt{p}$

3: **for** i=1 to c **do**

4: $\langle q_i, \ldots, q_w \rangle$ = SLIDING(Q_i)

5: **for** j=1 to q-w+1 **do**

6: dw_offset = RANGE_SEARCH($root_i, e_i, q_j$)

7: $offset = dw\_offset - j + 1$

8: **if** $offset > 0$ *and not duplicated* **then**

9: ADD $offset$ TO *temp_results*

10: **end if**

11: $e_{i+1} = max_{B \in temp\_results}(\sqrt{e_i^2 - d(q_J, B)^2})$

12: **end for**

13: **end for**

14: results = POSTPROCESS(results)

multi-resolution scheme. Rather than using q_j in the range query, we use MBR_Q defined as the MBR containing all the sliding windows of Q. This corresponds to finding the centroid of MBR_Q and performing a single range query with the centroid as query sequence and with $e = Max(radius_{MBR_Q}) + \epsilon$.

3.1.5 Nearest Neighbor Queries

k-nearest neighbor (kNN) queries are another common queries in time sequence databases. Algorithm 16 describes kNN queries over an rCMRI index which is based on standard kNN queries that can be found in [MNPT06]. The algorithm first finds the k closest subsequences to the largest power of two decomposition of query Q in its corresponding R-tree in rCMRI using a standard kNN query. Since we construct $n-w+1$ subqueries for each sliding window of the query at resolution 2^i, an additional post-processing phase incurs where the $n - w + 1$ result sets are merged to obtain a single set of k subsequences. Then, the distance ϵ to the k^{th} subsequence retrieved is used in a range query such as the one described in the previous subsection.

3.1.6 Correlation Queries

Algorithm 17 shows how to perform a Pearson bivariate correlation query on an rCMRI index. This algorithm is a slight modification from [KS07]: the range variable e_0 is set as a function of the correlation parameter r as in Chapter 2.

Note that all correlation queries are to be performed on normalized sequences and rCMRI does not contain enough information to be able to normalize the sequences

Algorithm 16 KNN query over the rCMRI index

Input: An integer k, a query Q of length q, the root of the i^{th} index of the CMRI $root_i$.

Output: The results *results*.

1: Q_i = PARTITION(Q)
2: **for** $i = Max(i)$ **do**
3: $\langle q_i, \ldots, q_w \rangle$ = SLIDING(Q_i)
4: **for** j=1 to q-w+1 **do**
5: dw_offset = KNN_SEARCH($root_i, q_j$)
6: $offset = dw\_offset - j + 1$
7: **if** offset > 0 and not duplicated **then**
8: ADD offset TO results
9: **end if**
10: **end for**
11: **end for**
12: $k\_results$ = POSTPROCESS(temp_results)
13: $\epsilon = Max_i(D(Q, k\_results_i))$
14: $results$ = RANGE(Q, ϵ)
15: $results$ = POSTPROCESS(results)

Algorithm 17 Correlation query over rCMRI

Input: A range parameter ϵ, a correlation parameter r, a query Q of length q, the root of the i^{th} index of rCMRI $root_i$, the window size w and the number c of R-tree indexes in rCMRI.

Output: The query results *results*.

1: Q_i = PARTITION(Q)

2: $e_1 = \sqrt{2w(1-r)}/\sqrt{p}$

3: **for** i=1 to c **do**

4: $\langle q_i, \ldots, q_w \rangle$ = SLIDING(Q_i)

5: **for** j=1 to q-w+1 **do**

6: dw_offset = RANGE_SEARCH($root_i, e_i, q_j$)

7: $offset = dw\_offset - j + 1$

8: **if** offset > 0 and not duplicated **then**

9: ADD offset TO temp_results

10: **end if**

11: $e_{i+1} = max_{B \in temp\_results}(\sqrt{e_i^2 - d(q_J, B)^2})$

12: **end for**

13: **end for**

14: results = POSTPROCESS(results)

at query time, unlike done in [NS08a]. Hence, we should now use an index with normalized values, different than the one used for range and nearest neighbor queries where subsequences are non-normalized. Since the size of rCMRI is smaller than the original CMRI, this may not be an issue.

3.1.7 Rank Order Correlation Queries

In this section, we present a rank order correlation query algorithm that uses rCMRI. Theoretical background on rank order correlation queries was given in the previous chapter. The algorithm is similar to a Pearson correlation query, but uses data that is stored in rank space. Hence, Algorithm 18 is a slight variation on Algorithm 17 but on rank space data rather than on normalized space data.

Algorithm 18 Rank order correlation query algorithm over rCMRI

Input: A range parameter ϵ, a correlation parameter r, a query Q of length q, the root of the i^{th} index of rCMRI $root_i$, the window size w and the number of R-tree indexes c in rCMRI.

Output: The results *results*.

1: Q_i = PARTITION(Q)

2: $e_0 = \sqrt{(1-r)(w^3-w)/6}/\sqrt{p}$

3: **for** i=1 to c **do**

4: $\langle q_i, \ldots, q_w \rangle$ = SLIDING(Q_i)

5: **for** j=1 to q-w+1 **do**

6: dw_offset = RANGE_SEARCH($root_i, e_i, q_j$)

7: $offset = dw\_offset - j + 1$

8: **if** offset > 0 and not duplicated **then**

9: ADD offset TO temp_results

10: **end if**

11: $e_{i+1} = max_{B \in temp\_results}(\sqrt{e_i^2 - d(q_J, B)^2})$

12: **end for**

13: **end for**

14: results = POSTPROCESS(results)

Chapter 4

Performance Evaluation and Analysis

In this chapter, we describe the experiments conducted for the evaluation of our proposed solutions for various queries over rCMRI and present the results. We begin by correlation queries followed by multivariate correlation queries and rank order queries. Then we proceed with rCMRI analysis.

4.1 Performance of Correlation Queries

We have evaluated the performance of our algorithms for fast correlation analysis and queries on time series datasets on a typical desktop PC using 3.2 GHz Intel Pentium IV processors with 504 MB of RAM running Windows XP. Our test programs are written in C++ and MATLAB programs were used to verify correctness of

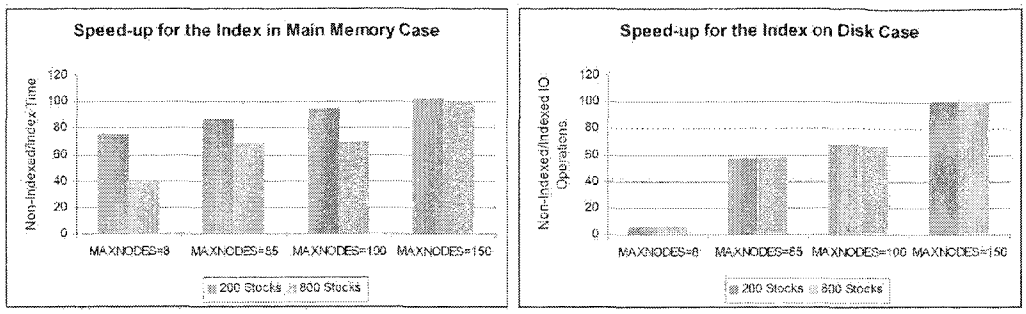

Figure 4.1: Average performance of bivariate correlation queries for different fan-outs and cases.

the results of the programs. We ran 15 runs of different correlation query instances that returned from 0 to 1000 records. Our analyses of the results indicated that the proposed algorithms provide online response times to complex correlation queries in standard computing environments and showed significant improvement over brute force sequential methods. In our experiments, we used real life stock market data, about 800 stocks sampled every day for a period of 400 days, of total size approximately 3 MB generating an index size of approximately 115 MB (using 5 resolutions and reduced data dimensionality of 8).

4.1.1 Performance of Bivariate and Multivariate Correlation Queries

We have studied the performance of complex correlation queries with single tree accesses to our indexing scheme, rCMRI. When multiple tree accesses are needed, in the worst case (when the query and the indexed data do not allow pruning between

tree accesses which happens when there is an exact match), the execution time and the number of disk I/O operations required increased linearly with the number of tree accesses. In both cases where data was in memory and on disk, for the bivariate and multivariate correlation query algorithms, even when we have up to 5 different tree accesses for a single query, we achieved an order of magnitude improvements in run time over brute force algorithms. Theoretically, point queries on R-trees offer at best $O(logn)$ performance where n is the number of data points. The complexity of range queries is $O(n + (n-1)/(fanout-1))$ in the worst case which occur when all the n points are retrieved. Since the number of query results is often very small compared to n, the performance on average is closer to point queries.

As mentioned earlier, to the best of our knowledge, no other correlation analysis algorithm offers the same flexibility with ad-hoc, user-defined queries. However, here we compare our work to other solutions described in the literature, i.e., StatStream [SZ02, SZ04] and the Spatial Cone Tree [HKSZ03a, HKSZ03b, HKSZ03c], for fixed correlation parameter values (to compare with StatStream) and fixed query length (to compare with the Spatial Cone Tree). StatStream offers a worse case run time of $O(d)$, where d is the number of dimensions of the grid structure used to index the time series data. Hence, for queries with fixed correlation parameters, StatStream offers the best solution. However, if we want variable values of correlation parameters through implementing StatStream, we would need to scan the whole dataset in order to regenerate the grid structure, i.e., similar to the brute force sequential algorithm. [HKSZ03a, HKSZ03b, HKSZ03c] report 45% to 85% savings on computational costs

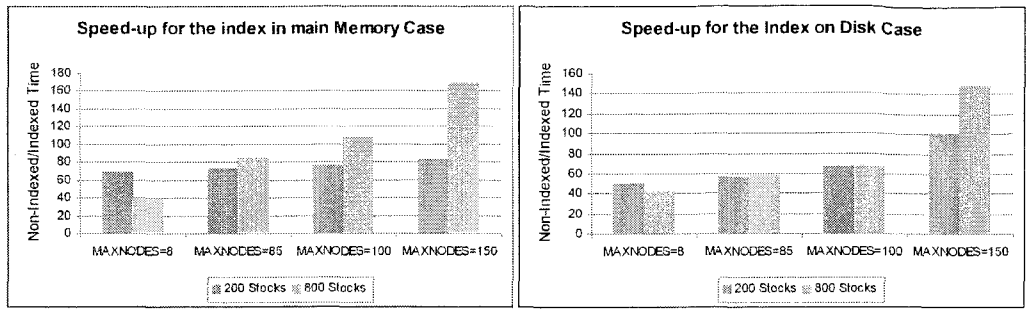

Figure 4.2: Average performance of multivariate correlation queries for different fan-outs and cases

for the Spatial Cone Tree solution. An advantage of our proposed algorithms is that they are scalable: for single tree accesses (fixed query lengths), our solution offers 80% to 95% savings on I/O operations compared to sequential algorithms on real life data. The main advantage of our solution is its support for ad-hoc correlation queries, multivariate correlation queries, and dynamically time warped correlation queries.

Figures 4.1 and 4.2 show average speed-ups over the brute force algorithms for queries returning 0 to 1000 results. The in-memory case measures performance in terms of run time and the on-disk case measures performance in terms of I/O operations. Performance is measured against different values of MAXNODES, which is the number of nodes an R-tree node can contain.

We do not present performance data for DTWC queries since no performance altering modifications to the algorithm presented in [KR04] are made. Interested reader can refer to that paper for detailed performance evaluations.

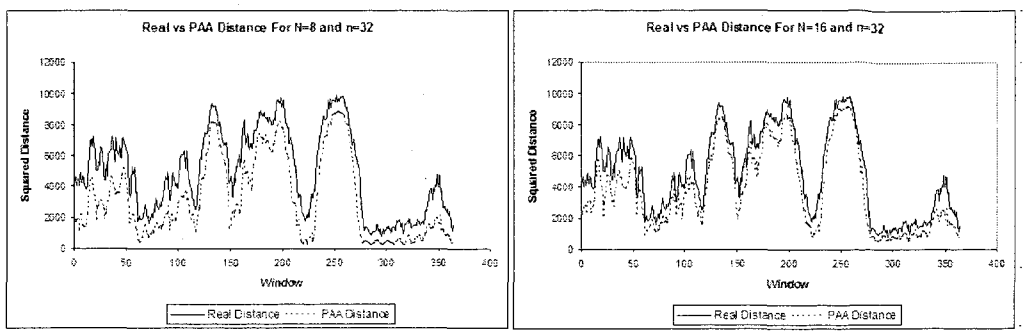

Figure 4.3: Rank order reduction (a) Precision of for N=8 (b) Precision for N=16.

4.1.2 Performance of Rank Order Correlation

In this section, we first address the precision of PAA as a dimensionality reduction over rank order value spaces. We then proceed to analyse the performance of PAA correlation queries compared to sequential scan (the only other solution to the best of our knowledge) for the case where data resides in memory and the case where data resides on disk.

4.1.3 Precision of PAA Reduction over Rank Order Data

PAA reduced time series yield lower bounding distance measures within 20% of the real distance measured in value space when the dimension of the feature space is set to N=8 and 13% of the real distance measured in value space when the dimension of the feature space is set to N=16. In both cases, the dimensionality n of the value space is set to 32, which is also the sliding window size. Figure 4.3 shows the real squared distance of a given set of windows from a given query as a solid line and the lower bounding PAA distance of a reduced window from a given query. Note that

the lower bounding PAA distance is given by the following equation for two PAA sequences X and Y [CKMP01a]:

$$D_{PAA} = \sqrt{\frac{n}{N}\sum_{i=1}^{N}(X_i - Y_i)^2} \tag{4.1}$$

Figure 4.3(a) shows the case where the window size 32 in value space is reduced to a feature space of dimension 8, and Figure 4.3(b) shows the case where the dimension of the feature space is 16.

4.1.4 Runtime and I/O Operations of Rank Order Queries

Figure 4.4(a) shows the speed-up for the case where data is in memory, where the speed-up is defined as the ratio $T_{sequential}/T_{indexed}$ for queries returning 10 to 100 records and correlation coefficients set from 0.85 to 0.95. Figure 4.4(b) shows the percentage of I/O operations saved for the case where data is on disk, where this percentage is defined as *total windows/nodes accessed.* In both cases, we can see significant performance gains through using the polyhedral index for rank order correlation queries. If the result set approaches the total number of elements in the index, performance degrades. However, most often the number of results returned by a range query is much smaller than the total number of data points stored in the R-tree index.

Figure 4.4(c) shows performance comparisons using real NYSE stock market data, normally distributed synthetic data and uniformly distributed synthetic data. We can see that the performance of the polyhedral index over each datasets stays in the same

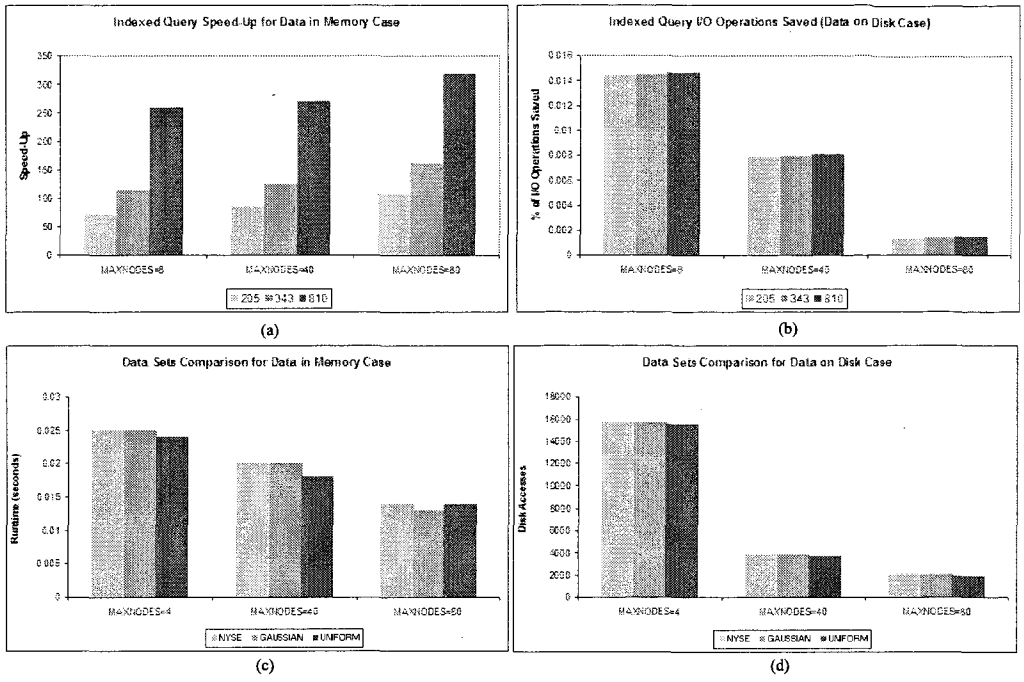

Figure 4.4: Rank order query performance (a) Data in memory case (b) Data on disk case (c) Dataset comparisons (real and synthetic) for data in memory case (d) Dataset comparisons (real and synthetic) for data on disk case.

order of magnitude and that the polyhedral index performs well over real as well as synthetic data.

Furthermore, we remark that the memory consumption of the CMRI index is O(kn), where k is the number of resolution trees used to index the data where n is the number of windows stored such as n = s(t-w+1), where s is the number of stocks, t is the number values in each stock and w is the window size.

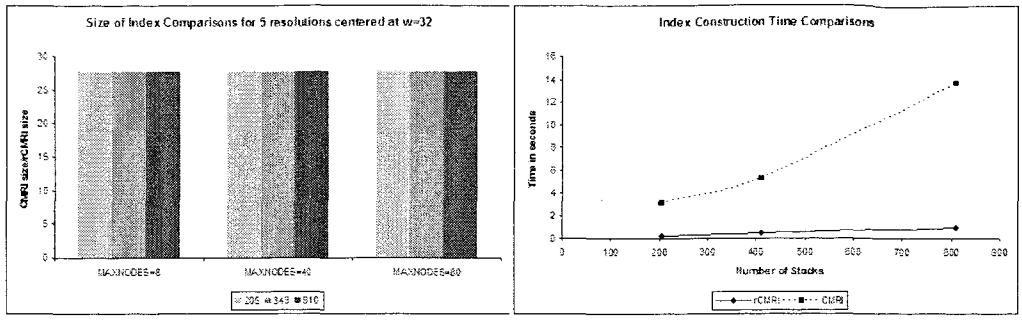

Figure 4.5: (a) Size Comparison between rCMRI and CMRI (b) Index construction time comaprison between rCMRI and CMRI.

4.2 Performance Analysis of the rCMRI Index

In this section, we provide different comparisons between rCMRI based on Duality Matching and the original CMRI. These comparisons span issues such as the index size, which is smaller by a factor of 1/w for rCMRI, the index construction time, for which we provide runtime performance measures, and the runtime and I/O operations required for range queries, which can be used as benchmarks for other queries such as correlation queries and nearest-neighbour queries since range queries are often the basis of these other queries. We also compare the two indexing schemes based on precision, experimental data and data volume. In general, we can say that rCMRI improves over CMRI in terms of index size, and hence index construction time. However, as we will see, query time for both indexes stays in the same order of magnitude when no buffering is allowed. When buffering is used, query response time is improved with rCMRI when measured through the number of I/O operations.

The general setup for our experiments was the following: we used real stock

market data from the New York Stock Exchange (NYSE). Each stock was sampled over 400 days, hence producing 400 floating point values per stock. We increased the data volume from approximately 200 to 2000 stocks in our experiments on rCMRI and from approximately 200 to 1000 stocks in our comparisons between the reduced CMRI and the original CMRI. As we will see, rCMRI indicated better performance or was comparable to CMRI in all the experiments carried out. Furthermore, according to our experiments using real stock market data, rCMRI retrieves 20% less false positives than the original CMRI, which is consistent with the fact that Duality Matching was shown to be more precise than the I-Adaptive index. Most of these characteristics are due to the fact that rCMRI improves by a factor of 1/w the size of the original CMRI although the complexity of the algorithms itself is increased since, as for range queries, multiple queries over a smaller index are required.

4.2.1 Comparisons on the Index Size and Construction Time

The main contribution of rCMRI is the reduction in the size of the index compared to CMRI while still retaining the advantages of a multi-resolution index. In this section, we present the results of our experiments on the size the reduced CMRI.

Figure 4.5(a), shows the ratio $CMRIsize/rCMRIsize$. As shown in the figure, for different values of MAXNODES (the underlying R-tree fanout), rCMRI yields an order of magnitude improvement. In theory, the improvement in the index size would be approximately $1/w$. In our case with $w = 32$, the improvement found experimentally was about 27.

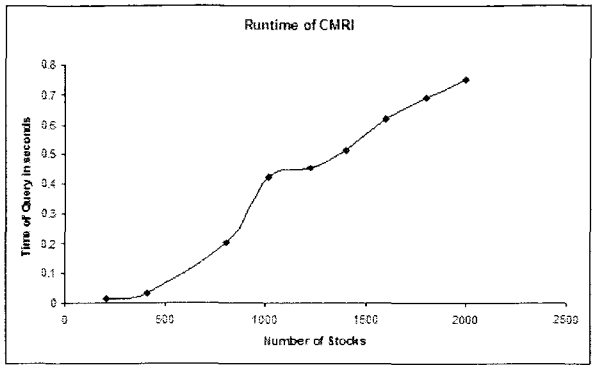

Figure 4.6: Runtime performance of rCMRI.

Figure 4.5(b) shows the improvement in the index construction time. As can be seen when the data volume increases, the improvement in index construction time increases to an order of magnitude: for approximately 800 stocks, the index construction time of the original CMRI is over 10 seconds whereas the index construction time of the reduced CMRI stays below one second. These properties indicate that rCMRI is more suitable for handling large volumes of data for example in data streams. This is mainly due to the use of disjoint windows rather than sliding windows.

4.2.2 Comparisons on the Runtime and I/O Operations

In this section, we compare rCMRI and CMRI on query response time (runtime and I/O operations).

In Figure 4.6, we show the runtime for range queries over rCMRI. The complexity of the underlying algorithm seems to be linear with the input size. In general, runtime comparisons between rCMRI and CMRI only compares slightly advantageously in favour of rCMRI, while not offering at least an order of magnitude improvement.

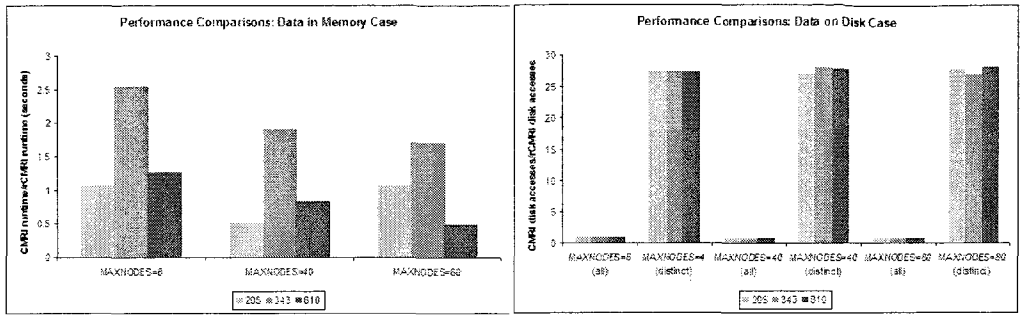

Figure 4.7: Range query performance comparison between rCMRI and CMRI (a) Data in Memory (b) Data on Disk.

Hence, when comparing runtime, we only get improvement ratios slightly higher than 1. This is shown in Figure 4.7(a). In Figure 4.7(b), for the case where no buffering is allowed (i.e. cases marked as "all" for "all disk accesses"), a similar analysis holds: improvement ratios are slightly over 1 and less than order of magnitude. This is explained as follow: for any query, rCMRI performs multiple sub-queries over the same smaller index. Hence, the same page or window may be accessed many times for a given query. When measuring all disk accesses (i.e. access to a window), we do not see any significant improvement in index performance. However, when measuring all distinct disk accesses, which approximates the behavior of the index when buffering is used, we get a significant improvement ratio which is above an order of magnitude. This improvement can also be numerically compared to the improvement in the size ratio described in the previous section. As a note, in our experiments, we use runtime as a measure of index performance when the data fits in memory, whereas we use I/O operations as a measure of index performance when the data cannot fit in memory

and resides on disk.

Chapter 5

Conclusion and Future Work

In this thesis, we presented algorithms for correlation analysis and queries and described how they differ from those proposed in [HKSZ03a, HKSZ03b, HKSZ03c, SZ02, SZ04]. The fastest solution is StatStream which runs in $O(d)$. However, the grid structure it uses to store the sliding windows is built and optimized for a fixed correlation query parameter. Hence, to use variable correlation query parameters, one would need to rebuild the entire index which corresponds to scanning the whole time series database. The Spatial Cone Tree solves this problem by making use of a specialized tree-based index. However, the Spatial Cone Tree did not originally support variable length queries efficiently. Both StatStream and Spatial Cone Tree are specialized index structures built for bivariate correlation analysis. Our solutions make use of R-trees and MRI/CMRI [KS04, KS07] variants. Hence, standard complex similarity queries with variable length such as range queries and nearest neighbor queries can be handled. Furthermore, the algorithms we propose over those index

solve variable bivariate correlation queries, multivariate correlation queries and the new algorithm we proposed for DTWC queries.

We introduced a new way of indexing time series for rank order correlation queries. Rank order correlation has been extensively used when dealing with nonlinear monotonic relationships between time series. Our method offers significant speed-up and I/O operation savings when compared to sequential scan, the only other solution. The overhead of maintaining an index is still linear in the number of windows stored which itself is in the order of the number of values in the input time series. This overhead makes sense when dealing with large datasets that cannot be sequentially scanned in a reasonable amount of time.

One of the problems with CMRI is its size. We addressed this issue and we showed how rCMRI, a scalable multi-resolution index, can be used to index time series in order to efficently answer different queries and how this solution helps in a query-response environment. Our experiments show that rCMRI can be built optimally and how it improves on existing solutions such as the original CMRI and Duality Matching. We also proposed algorithms that can be used in conjunction with rCMRI to solve efficiently distance based queries such as range and nearest neighbor queries and pattern based queries such as product moment and rank order correlation queries. Furthermore, we remark that rCMRI is not bound to a particular underlying spatial data structure such as the one we used in our experiments; we could use any data structure that can answer range queries without false dismissals such as R*-tree, R+-tree, M-tree, X-tree, etc.

As future work, we plan to extend the implementation of rCMRI to parallel and distributed environments. This is important since the huge real life datasets to be indexed may be scattered around multiple hosts. We also plan to investigate ways to answer multiscale queries and nonlinear non-monotonic correlation queries.

Bibliography

[AAF99] A.E. Abbadi, D. Agrawal, and H. Ferhatosmanoglu. Concentric hyperspaces and disk allocation for fast parallel range searching. *International Conference on Data Engineering (ICDE)*, pages 608–615, 1999.

[AFS95] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. *Proceedings of the 4th Int'l Conf. on Foundations of Data Organization and Algorithms*, pages 64–84, 1995.

[AHK01] C.C. Aggarwal, A. Hinneburg, and D.A. Klein. On the surprising behavior of distance metrics in high dimensional space. *Proceedings of the Eight International Conference on Database Theory*, 1973:420–434, 2001.

[AKFA08] I. Assent, R. Krieger, and T. Seidl F. Afschari. The ts-tree: Efficient time series search and retrieval. *International Conference on Extending Database Technology (EDBT)*, 2008.

[AWA00] D. Agrawal, Y.L. Wu, and A. El Abbadi. A comparison of dft and dwt based similarity search in time-series databases. *Conference on Information and Knowledge Management (CIKM)*, pages 448–495, 2000.

[BKK96] S. Berchtold, D.A. Keim, and H.P. Kriegel. The x-tree : An index structure for high-dimensional data. *International Conference on Very Large Data Bases (VLDB)*, pages 28–39, 1996.

[BKSS90] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust method for points and rectangles. *Proceedings ACM SIGMOD Conferences on Management of Data*, 19(2):322–331, 1990.

[CC07] A. Chatterjee and B.K. Chakrabarti. Econophysics of stock and other markets. Springer, New York, 2007.

[CFY03] K.P. Chan, A.W. Fu, and C. Yu. Haar wavelet for efficient similarity search of time-series : With and without time warping. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):686–705, 2003.

[CKMP01a] K. Chakrabarti, E. Keogh, S. Mehrotra, and M.J. Pazzani. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information System*, 3(2):263–286, 2001.

[CKMP01b] K. Chakrabarti, E. Keogh, S. Mehrotra, and M.J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *SIGMOD*, 30(2):151–162, 2001.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. *Proceedings of the 23rd VLDB International Conference*, pages 426–435, 1997.

[CS83] W. D. Cook and L.M. Seiford. The geometry of rank-order tests. *The American Statistician*, 37(4):307–311, 1983.

[dCdTB04] R. de Caluwe, D. de Tre, and G. Bordogna. Spatio-temporal databases. 2004.

[DGSZ03] V. Dohnal, C. Gennaro, P. Savino, and P. Zezula. D-index distance searching index for metric data sets. *Multimedia Tools and Applications*, pages 9–33, 2003.

[DR05] N. Dendukuri and C. Reinhold. Correlation and regression. *American Journal of Roetgenology*, 185(1):3–18, 2005.

[Fal96] C. Faloutsos. Searching multimedia databases by content. Kluwer Academic Press, 1996.

[FK92] C. Faloutsos and C. Kamel. Parallel r-trees. *ACM SIGMOD International Conferent on Management of Data*, pages 195–203, 1992.

[FKK96] C. Faloutsos, I. Kamel, and N. Koudas. Declustering spatial databases on a multi-computer architecture. *International Conference on Extending Database Technology (EDBT)*, pages 507–518, 1996.

[FMR94] C. Faloutsos, Y. Manolopoulos, and M. Ranganathan. Fast subsequence matching in time-series databases. *SIGMOD*, pages 419–429, 1994.

[FSR87] C. Faloutsos, T. Sellis, and N. Roussopoulos. The r+-tree: A dynamic index for multi-dimensional objects. *International Conference on Very Large Data Bases (VLDB)*, pages 507–518, 1987.

[FY00] C. Faloutsos and B.K. Yi. Fast time sequence indexing for arbitrary lp norms. *International Conference on Very Large Data BasesVLDB*, pages 385–394, 2000.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. *Proceeedings ACM SIGMOD Conference*, pages 47–57, 1984.

[Haa10] A. Haae. Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, 69:331–371, 1910.

[HKMW99] H. Horinokuchi, K. Kaneko, A. Makinouchi, and B. Wang. Parallel r-tree search algorithms on dsvm. *Database Systems for Advanced Applications (DASFAA)*, pages 237–245, 1999.

[HKSZ03a] Y. Huang, V. Kumar, S. Shekkara, and P. Zhang. Correlation analysis of spatial time series datasets : A filter-and-refine approach. *Proc. of the*

7th Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2637:532–544, 2003.

[HKSZ03b] Y. Huang, V. Kumar, S. Shekkara, and P. Zhang. Exploiting spatial autocorrelation to efficiently process correlation-based similarity queries. *Proc. of the 8th Intl. Symp. on Spatial and Temporal Databases*, pages 449–468, 2003.

[HKSZ03c] Y. Huang, V. Kumar, S. Shekkara, and P. Zhang. Spatial cone tree : An index structure for correlation-based similarity queries on spatial time series data. *International Workshop on Next Generation Geospatial Information*, pages 19–21, 2003.

[JMK05] R. Juday, A. Mahlanobis, and B.V.K Vijaya Kumar. Correlation pattern recognition. Cambridge University Press, New York, 2005.

[JOV05] H.V. Jagadish, B.C. Ooi, and Q.H. Vu. Baton: A balanced tree structure for peer-to-peer networks. *International Conference on Very Large Data Bases (VLDB)*, 2005.

[JOV06] H.V. Jagadish, B.C. Ooi, and Q.H. Vu. Vbi-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. *International Conference on Data Engineering (ICDE)*, 2006.

[KG90] M. Kendall and J.D. Gibbons. Rank correlation methods. Edward Arnold, London, 1990.

[KLLC03] E. Keogh, J. Lin, S. Lonardi, and B. Chiu. A symbolic representation of time series with implications for streaming algorithms. *ACM SIGKDD*, 2003.

[KLLC07] E. Keogh, J. Lin, S. Lonardi, and B. Chiu. Experiencing sax: A novel symbolic representation of time series. *Data Mining and Knowledge Discovery (DMKD)*, 2007.

[KM03] K. Kelley and S.E. Maxwell. Sample size for multiple regression: obtaining regression coefficients that are accurate, not simply significant. *Psychological Methods*, 8(3):305–321, 2003.

[KR04] E. Keogh and C.A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, 2004.

[KS04] T. Kahveci and A.K. Singh. Optimizing similarity search for arbitrary length time series queries. *IEEE Transactions on Knowledge and Data Engineering*, 16(4):418–433, 2004.

[KS07] S. Kadiyala and N. Shiri. A compact multi-resolution index for variable length queries in time series databases. *Knowledge and Information Systems*, 2007.

[KSUY01] R. Kataoka, Y. Sakurai, S. Uemura, and M. Yoshikawa. Similarity search for adaptive ellipsoid queries using spatial transformation. *Internation Conference on Very Large Data Bases (VLDB)*, pages 231–240, 2001.

[LMW00] W.-K. Loh, Y.-S. Moon, and K.-Y. Whang. Efficient time-series subsequence matching using duality in constructing windows. *AITrc Technical Report*, 2000.

[LMW01] W.-K. Loh, Y.-S. Moon, and K.-Y. Whang. Duality-based subsequence matching in time-series databases. *International Conference on Data Engineering (ICDE)*, pages 263–272, 2001.

[LMW02] W.-K. Loh, Y.-S. Moon, and K.-Y. Whang. General match: A subsequence matching method in time-series databases based on generalized windows. *SIGMOD Conference*, pages 382–392, 2002.

[LS99] S.T. Leutenegger and B. Schnitzer. Master-client r-trees: A new parallel r-tree architecture. *Statistical and Scientific Database Management (SSBDM)*, pages 68–77, 1999.

[LV06] J.A. Lee and M. Verleysen. Nonlinear dimensionality reduction. Springer, New York, 2006.

[MLR07a] C. Du Mouza, W. Litwin, and P. Rigaux. A framework for distributed spatial indexing in shared-nothing architectures. *Bases de Donnees Avancees (BDA)*, pages 23–26, 2007.

[MLR07b] C. Du Mouza, W. Litwin, and P. Rigaux. Sd-tree: A scalable distributed r-tree. *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 296–305, 2007.

[MNPT06] Y. Manolopoulos, A. Nanopoulos, A.N. Papadopoulos, and Y. Theodoridis. R-trees : Theory and applications. Springer, New York, 2006.

[MR81] C. Myers and L. Rabiner. A level building dynamic time warping algorithm for connected word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 2(29):284–297, 1981.

[NS] P. Nguyen and N. Shiri. A scalable compact multi-resolution index for time series datasets. *under review.*

[NS08a] P. Nguyen and N. Shiri. Fast correlation analysis on time series databases. *Proc. ACM 17th Conf. on Information and Knowledge Management (CIKM), Napa Valley, California*, Oct. 26-30 2008.

[NS08b] P. Nguyen and N. Shiri. Polyhedral index for rank order correlation queries. *Proc. ACM 17th Conf. on Information and Knowledge Management (CIKM), Napa Valley, California*, Oct. 26-30 2008.

[Pea01] K. Pearson. Mathematical contributions to the theory of evolution. *Supplement to a memoir on skew variation, Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 197:443–459, 1901.

[Sam06] H. Samet. Foundations of multidimensional and metric data structures. Morgan Kaufmann, San Francisco, 2006.

[Sch79] R.S. Schulman. A geometric model of rank correlation. *The American Statistician*, 33(2):77–80, 1979.

[Spe04] C. Spearman. The proof and measurement for association between two things. *American Journal of Psychology*, 15:72–101, 1904.

[SZ02] D. Shasha and Z. Zhu. Statstream: Statistical monitoring of thousands of data streams in real time. *International Conference on Very Large Data Bases (VLDB)*, pages 358–369, 2002.

[SZ04] D. Shasha and Z. Zhu. High performance discovery in time series: Techniques and case studies. Springer, New York, 2004.