Efficient and Scalable Search for Similar Patterns in Time Series Data

Srividya Kadiyala

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

March 2006

Your file Votre référence
ISBN: 0-494-14323-1
Our file Notre référence
ISBN: 0-494-14323-1

ABSTRACT

Efficient and Scalable Search for Similar Patterns in Time Series Data

Srividya Kadiyala

Popularity of time series databases for predicting future events and trends in applications such as market analysis and weather forecast require the development of more reliable, fast, and memory efficient indexes. In this thesis, we consider searching similar patterns in time series data for variable length queries. Recently an indexing technique called Multi-Resolution Index (MRI) has been proposed to solve this problem [Kah01, Kah04] which uses compression to reduce the index size. However, the processor workload and memory curtails the opportunity of utilizing compression as an additional step. Motivated by the need and limitations of existing techniques, the main objective of this thesis is to develop an alternative multi-resolution index structure and algorithm, to which we refer as Compact MRI (CMRI). This new technique takes advantage of an existing dimensionality reduction technique called Adaptive Piecewise Constant Approximation (APCA) [Keo01]. Advantages of CMRI is that it utilizes less space without requiring any compression and gains high precision. We have implemented MRI and CMRI and performed extensive experiments to compare them. To evaluate the precision and performance of CMRI, we have used both real and synthetic data, and compared the results with MRI. The experimental results indicate that CMRI improves precision, ranging from 0.75 to 0.89 on real data, and from 0.80 to 0.95 on synthetic data. Furthermore, CMRI is superior over MRI in performance as the number of disk I/Os required by CMRI is close to minimal. Compared to sequential scan, CMRI is 4 to 30 times faster, observed on both real and synthetic data.

Dedicated to my Parents, Brother and Husband

ACKNOWLEDGEMENTS

I wish to express my gratitude and indebtedness to my supervisor, Professor Nematollaah Shiri for his care, encouragement, guidance, support, constant help and devoting his valuable time for me always in spite of his busy schedule. I also wish to express here that Dr. Shiri has been my constant source of inspiration for the past two years with his ever ending enthusiasm to teach, learn and work on new areas. I thank him for giving me the opportunity to work under him.

My sincere thanks to Professor Haarslev for his support and understanding. I am thankful to Stan, a system analyst in the ENCS faculty, for his support in solving my system related problems.

I would like to thank Ali Kiani for his constant help and involvement in discussions. I also wish to thank Anand Thamildurai and Mihail Halachev, for their guidance and help in gnome research. I thank every member of the database group for supporting me during my experiments. I thank my friends Sri, Vasavi, and Naren for their support. Last but not the least I show gratitude to my husband Prasad for his understanding, involvement in discussions and constant motivation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

TSD	Time Series Data
DWT	Discrete Wavelet Transform
DFT	Discrete Fourier Transform
APCA	Adaptive Piecewise Constant Approximation
MRI	Multi-Resolution Index
RMRI	Reduced MRI
CMRI	Compact MRI
GDR	Global Dimensionality Reduction
LDR	Local Dimensionality Reduction
HWT	Haar Wavelet Transformation
PAA	Piecewise Aggregate Approximation
SVD	Singular Value Decomposition
MBR	Minimum Bounding Rectangle
SBR	Skyline Bounding Region
KNN	K Nearest Neighbours
VBR	Virtual Bounding Rectangle
CoMRI	Compressed MRI
IN	Internal Nodes
DP	Data Points

Chapter 1

Introduction

A time series is a sequence of real numbers, each of which represents a value, at a particular time, of some attribute such as price, temperature, etc. Such time series are normally very large as the data is stored at regular time intervals to preserve the history. Each subsequence of a time series (TS) consisting of n values may be viewed as a window of size n. Consequently, time series (TS) are often viewed and treated as multi-dimensional. A collection of time series is called as *Time Series Database* (TSDB). We may also refer to such data as time sequence or sequence data. Pattern discovery algorithms have been developed to analyse the sequences in a TSDB to identify patterns which are implicit in the data in order to predict future events.

The popularity of time series databases for predicting future events and trends is growing fast in many applications, such as agriculture, finance, sales, production, industry, genomes, proteins, astronomy, chemistry, crime, transport, tourism, etc

[Raf97, Keo00, Keo01]. For example, crime databases are used to analyze the crime rate or pattern in previous months or years, and to take necessary actions to reduce crimes. Biological sequence data is used, when dealing with an unknown species, to search for similar gene patterns to find the most relevant classification of the species. Range queries and nearest neighbour queries are typical queries used to search similar matches. These queries are studied in detail in Chapter 3.

When a TSDB is very large, traditional sequential scan of the database would be inadequate to search for similar patterns in the database. The demand to support efficient query processing over such data is increasing [Raf97, Keo00, Keo01]. Since time series data is multi-dimensional, in order to support efficient searching, we need multidimensional index structures. A preprocessing step to reduce the dimensions of time series is required at this stage before creating an index over the database, without which the size of the index becomes prohibitively large. For this, there has been a number of techniques proposed to reduce the dimensions of the time series in order to improve the index both in terms of its size and performance. The latter is measured in terms of the the number of disk I/Os performed to process queries as well as the precision/accuracy. These techniques, referred to in the literature as *dimensionality reduction techniques*, are studied in Chapter 2. After applying such a technique, only a few high magnitude dimensions are considered for creating the index. There is a trade off between accuracy, speed, and memory consumed depending on the number of dimensions chosen to create the index. As the number of dimensions chosen increases, the memory and precision/accuracy of the index increase while the speed decreases, and vice versa.

Given a query posed to TSDB, the index supports searching for similar matches in the database. This becomes more complicated when the query length is variable. An index over TSDB should then be flexible to support such queries. That is, the index should be able to support varying length queries and thus providing the user a better chance to identify events, trends, or similarity to decide/predict a future event. Furthermore, a desired index to support such queries should occupy less space, perform less number of disk I/O operations, and be fast in answering the queries. This is our motivation in this research to study the aforementioned issues and develop a desired index structure and technique to support variable length queries on time series databases for pattern analysis and discovery. We refer to time series also as time sequences, interchangeably.

1.1 Thesis Contributions

In order to develop an efficient index structure to support variable length queries on TSDB, we first study existing techniques for dimensionality reduction of TSD, and study their impact. An appropriate dimensionality reduction technique must be chosen for the index, depending on the data. For this, the dimensionality reduction techniques we consider includes Discrete Wavelet Transform (DWT), Discrete Fourier Transform (DFT), and Adaptive Piecewise Constant Approximation

(APCA). Of these, DWT and DFT are global dimensionality reduction techniques. That is, for every sequence in the database, dimensionality reduction is performed in the same way. APCA is a local dimensionality reduction technique that represents each time series in a unique way on the basis of energy/values of the signal. Our proposed index technique uses APCA as the dimensionality reduction technique because it gives the highest performance to the index [Keo01]. The gain in precision of our indexing technique is due to the APCA representation, where precision is the measure of accuracy of the index. We remark that precision is defined as the ratio of sequences in the actual result to the candidate sequences obtained through the index.

The next step in our research was to study alternative index structure that could support variable length queries. For this, we study several multidimensional indexing techniques such as R-tree (in Section 3.3), I-adaptive index [Fal94], and Multi-Resolution index (MRI) (in Section 3.4). Of these, MRI is the most efficient indexing technique for variable length queries till now. In our study, we noted that MRI could be further improved in terms of its size and performance. To identify opportunities for such improvements, we carried out a detailed analysis of MRI and studied memory management and storage techniques for cases where the index exceeds the size of available main memory.

We used the analysis results and developed an efficient and scalable index structure, in two stages. First we propose Reduced MRI (RMRI for short) which has a better performance than MRI. Then the structure of RMRI is further refined and improved, to which we refer as Compact MRI (CMRI, Section 4.3). An optimized search algorithm for exact search on the index has also been proposed (in Section 4.3.1). The study of memory utilization and optimization in the context of CMRI has been done (in Section 4.4).

1.2 Thesis Outline

We study efficient and scalable index structure for variable length queries on TSDB and introduce a new multi-resolution index structure. The rest of this report is organized as follows. Chapter 2 gives the background knowledge of dimensionality reduction on TSD in section 2.1. Global Dimensionality reduction techniques which have similar performance are studied in section 2.2 (DFT and DWT). A local dimensionality reduction technique, APCA, used in our work is studied in section 2.3. Chapter 3 explains the need of index, reviews the related indexing techniques. and describe some indexing techniques in detail. Section 3.1 explains different types of similarity matching queries and the need of index on time series data. Section 3.2 provides an overview of the existing indexing techniques. A detailed explanation of R-tree structure is provided in section 3.3. As our proposed technique improves MRI, a detailed study of the MRI structure, query partitioning and searching are presented in section 3.4. Section 3.5 discusses the impact of different parameters such as dimensions of index, threshold, and query length on the performance.

Chapter 4 introduces the structures of RMRI and CMRI, our proposed techniques. Section 4.1 explains redundancy of insertions in MRI and shows the structure obtained by avoiding this redundancy, known as RMRI. The purpose of RMRI is to serve as an intermediate structure to support correctness of CMRI. The construction of our technique CMRI is introduced in section 4.2. We also describe why our technique occupies less space compared to MRI. Section 4.3 introduces the modified structure CMRI obtained from RMRI. We also provide a construction algorithm for CMRI. This section also includes modified search algorithms for CMRI, obtained from those proposed for MRI. In section 4.4, we study the memory utilization by index.

In Chapter 5, we report our experimental results for range and nearest neighbour queries on both real and synthetic data, obtained by changing various parameters such as dimensions, threshold, number of neighbours, and query lengths. Section 5.1 presents our experimental results for range queries on index created for different dimensions. In section 5.2, we discuss the results obtained for nearest neighbour queries at dimension 8 of the index. Memory consumption of MRI and CMRI on stock market data is shown in section 5.3. Section 5.4 explains the impact of variable query lengths on the index at some particular dimension. Section 5.5 presents the results obtained by different threshold values. Section 5.6 shows the results for range queries on mean value index.

Chapter 6 includes summarization of our work together with some suggestions for future work.

Chapter 2

Background

In this chapter, we describe the background concepts and techniques used in the development of an index on time series data. In section 3.1, we introduce several concepts on TS and discuss the use of different dimensionality reduction techniques on time series. Section 3.2 give details about the dimensionality reduction techniques DWT and DFT. Even though we do not use DFT in our work, we discuss DFT since it has the same performance as DWT when the conjugate property is considered [Raf98]. We also give detail explanation of APCA representation of time series, since it is used in our work.

2.1 Dimensionality Reduction on Time Series

To scale up to higher dimensions for multi-dimensional TSD, a commonly used technique is dimensionality reduction. A multi-dimensional index structure is then created for the reduced space. The curse of dimensionality refers to the exponential growth of hyper volume as a function of dimensionality [Bel61]. In other words this curse makes it difficult to reduce/increase the dimensions, since reduction in the number of dimensions will increase the candidate set, and increase in the dimensions will increase the memory utilization by the index. Candidate set is defined as the number of matches returned by the index.

Through dimensionality reduction, most of the information in the sequence is reduced to few values (dimensions), and only some of the principal components are used to build the index. There are several techniques proposed for reducing dimensions of time series data. They can be divided into two categories *Global Dimensionality Reduction* (GDR) techniques and *Local Dimensionality Reduction* (LDR) techniques. If a *Global Dimensionality Reduction* technique [She80, Aga93, Fal96, Fal95, Cha99] is used, every sequence in the database is reduced in the same way. Such reduction techniques work well when the data is globally correlated. However, this may not always be the case, in particular when correlation of data varies. In such cases several *Local Dimensionality Reduction* (LDR) techniques [Keo00, Keo01] proposed recently can be used instead, which improves efficiency of search since the reduced sequences carry more information [Cha00, Keo01].

The dimensionality reduction techniques include Singular value decomposition (SVD) [Fal96, Kan98, Keo00, Kan98], Discrete Fourier transform (DFT) [Aga93], Multi-dimensional scaling [She80], Fast Map and its variants [Fal95], Discrete Wavelet Transform (DWT) [Cha99, Wu 00], Piecewise Aggregate approximation (PAA) [Keo00,

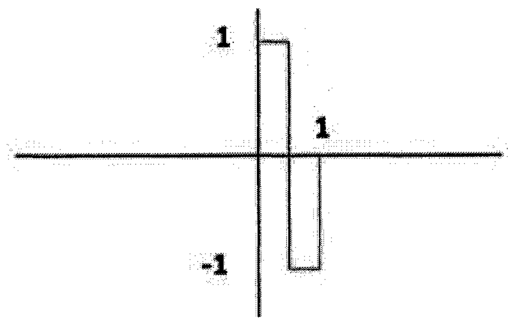


Figure 2.1: Haar Wavelet Component

Yi 00], and Adaptive Piecewise Constant Approximation (APCA) [Keo01]. The practical aspects of an application and preferences of the user will decide selection of an appropriate technique. Agrawal et al. [Aga93] used DFT to transform time sequence into frequency domain, and considered only the first few coefficients to reduce the dimensionality and built an R-tree index [Gut84]. DFT maps a one dimensional time domain discrete function into a representation in frequency domain. Chan and Fu [Cha99] proposed Haar wavelet transformation (HWT) for dimensionality reduction, used for image, speech and signal processing, and showed its superiority over DFT. Haar wavelet component used for transforming a signal is shown in Fig. 2.1 [Keo00, Wu 00]. This component is scaled and shifted to represent the original signal. DWT gives time frequency localization of the signal while DFT transforms it into different frequency parts. Their experiment results showed that pruning power of index using DWT is more than DFT for dimensionality reduction.

They proved that DWT performs better than DFT. Later on, Rafiei and Mendelzon showed that DFT can be improved by considering the symmetry property of the Fourier transforms, i.e., coefficients in the rear are complex conjugates of the coefficients in the front [Raf98]. This indicates that energy is not only concentrated on the first few coefficients but also on the last few, symmetrically. It suggests that while creating an index, one must also consider the last M coefficients along with the first M coefficients because most of the energy of the signal is preserved in few initial and last coefficients. Fig. 2.2 [Wu 00] shows how complex conjugate property of DFT helps in reconstructing the signal. The whole idea of selecting the coefficients is that selecting highest magnitude coefficients is to minimize the error involved in reconstruction of the signal. The complexity of DWT is O(n) whereas for DFT it is O(n log n), where n is the number of values in the input sequence.

Wu et al. [Wu 00] performed several experiments to compare these transformation techniques. Their results showed that none of these techniques is superior if we consider the conjugate property of the Fourier transform but rather their performance depends on the type of data in similarity search. That is any one of these transformations can be used for dimensionality reduction of the data. There are numerous methods such as edit distance, Euclidean distance, etc., to measure similarity of two time sequences. Euclidean distance is the most commonly used distance measure, defined as the straight line distance between two points [Aga93, Cha99, Raf97]. In N dimensions, the Euclidean distance between

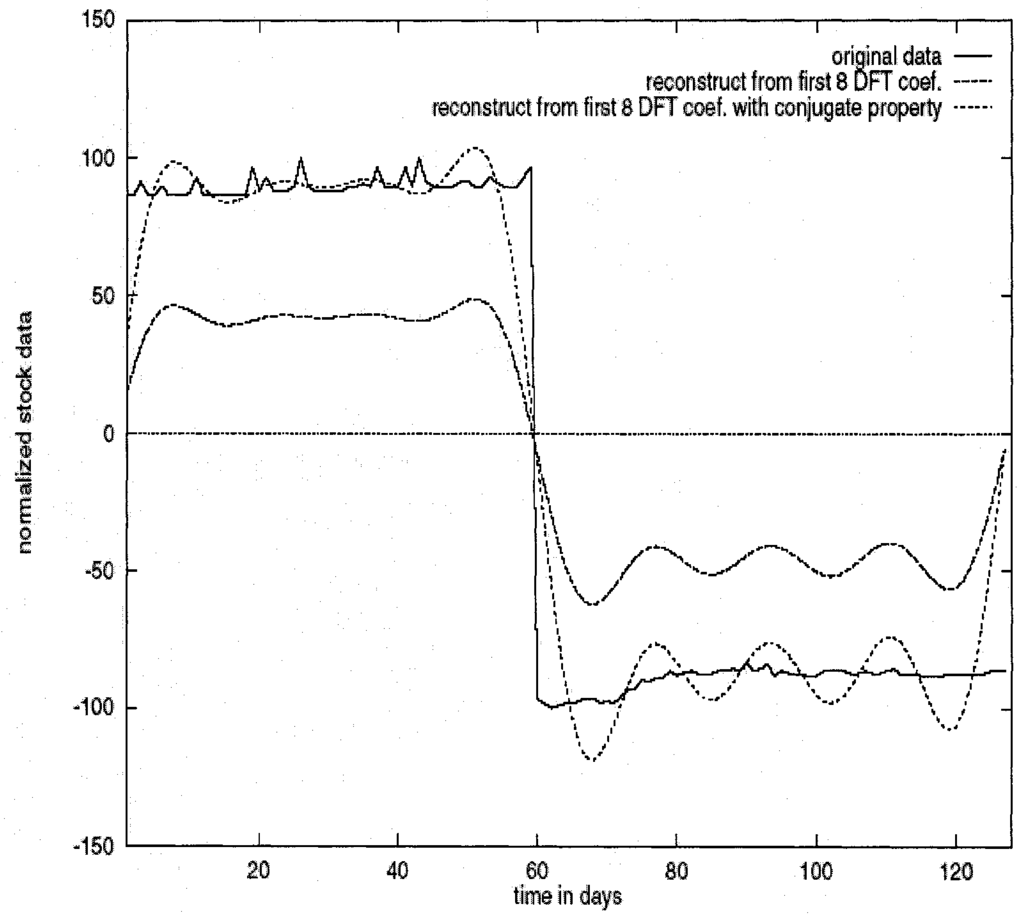


Figure 2.2: Conjugate property of Fourier Wavelet [Wu 00]

two points p and q is $\sqrt{(\sum_{i=1}^{N}(p_i - q_i)^2)}$, where p_i (or q_i) is the coordinate of point p (or q) in dimension i. Non-Euclidean metrics have also been used to determine similarity of time sequences. Smoothing techniques are employed to reduce irregularities (random fluctuations) in time series data [Raf97]. They provide a more clear view of the true underlying behavior of the series. The most common smoothing technique is the moving average which replaces each element of the series by either the simple or weighted average of n surrounding elements, where n is the width of the smoothing "window". Since the type of seasonality in data varies in general from series to series, so must the type of smoothing. Fig. 2.3 shows a simple graph for 10 day moving average smoothing. In the graph, the smooth curve is obtained by applying 10 day moving average technique, showing the overall behavior of the original curve. We explained this even though we don't use this technique, to provide knowledge about use of various techniques.

Mehrotra et al. [Cha00] introduced LDR through clustering. Keogh et al. [Keo00] proposed to split the time sequence into equal sized windows. The average of the values in each window is used to represent all the entries in the window. This compression technique is called Piecewise Aggregate Approximation (PAA) [Keo00]. In their experimental results, they show that PAA can be computed faster than SVD, Haar, and DFT. The pruning power of PAA is similar to that of DFT and Haar, but is poor compared to Singular value decomposition (SVD) [Keo00]. The authors also proposed subsequence searching by sequentially sliding the query sequence over

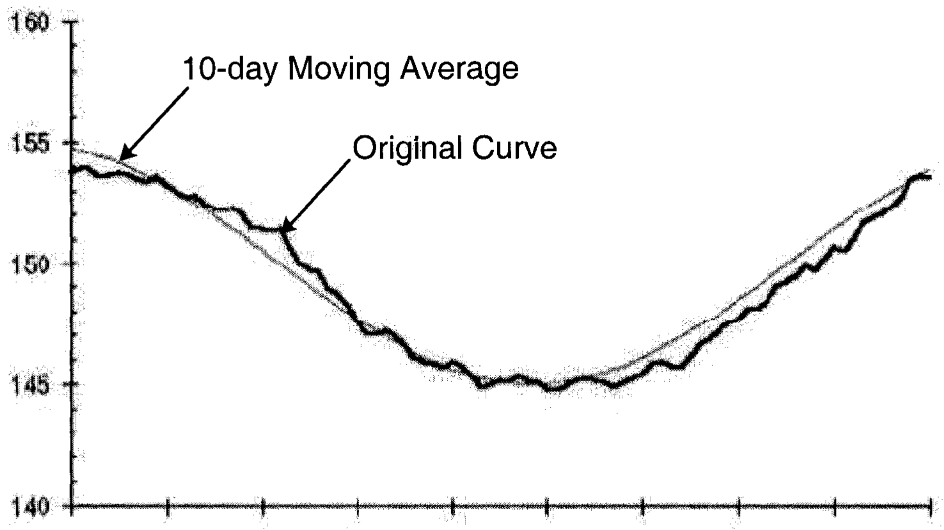


Figure 2.3: 10-day moving average

all the database sequences. Later on they proposed a different technique for LDR, known as Adaptive Piecewise Constant Approximation (APCA), which improves the pruning power of the index [Keo01]. This is a modification of PAA by splitting the time sequences into varying size windows [Keo01].

Popivanov and Miller [Pop02] later on show that PAA is poor in precision and performance compared to DFT, Haar wavelets, and Daubechies wavelets, where precision is measure of accuracy. According to this result, Daubechies wavelet gives the best precision and require lowest number of disk accesses among the three techniques. This implies that there is no clear winner among all global dimensionality reduction techniques and the results of a particular technique depends on the type of data used in the experiments. Details about DWT, DFT and APCA are given in next two sections.

2.2 Introduction to DWT and DFT

In this section, we quickly review important commonly used dimensionality reduction techniques. Understanding these would help understand the preprocessing phase of our index proposed in this research to support variable length queries on time sequences.

2.2.1 DWT: Haar Wavelet Transform

Wavelets allow a time series to be viewed in multiple resolutions. Each resolution reflects a different frequency. The wavelet technique takes averages and differences of a signal, and breaks the signal down into a spectrum. To calculate the Haar transform for a sequence of n elements, we first check if n is in power of 2 format. If n is not in this format we pad the sequence with zeros and increase n until it is in the required format (power of 2). Next we find the average of each pair of elements. Then we find the difference between each of these $n/2$ averages and the elements it was calculated from. This gives $n/2$ differences. We then consider an array whose length is n and fill the first half of the array with averages and second half with differences. This process is repeated on the first half of the array, averages, until the array has only one average element. The following example illustrates this process of computing DWT for a sequence.

Example:

Consider the following sequence s of eight elements

$s:$	7	1	6	6	3	-5	4	2

Average / Difference

Two elements, l and r, can be expressed as an average a and a difference d:

$a = (l + r)/2$

$d = a - l = r - a$

This operation is reversible, that is:

$l = a - d$

$r = a + d$

Averages:

Calculate averages for each consecutive, non-overlapping pair.

$(7 + 1) / 2 = 4$

$(6 + 6) / 2 = 6$

$(3 + (-5)) / 2 = -1$

$(4 + 2) / 2 = 3$

Differences:

$(7 - 4) = (4 - 1) = 3$

$(6 - 6) = (6 - 6) = 0$

$(3 - (-1)) = (-1 - (-5)) = 4$

$(4 - 3) = (3 - 2) = 1$

Averages and differences are combined to give HWT of a sequence. The first four elements are the averages, and the last four elements are the difference calculated above.

4	6	−1	3	3	0	4	1

This process is repeated on averages until we have one average element in the array, as shown in Table. 2.1.

Resolution	Average	Difference
3	7, 1, 6, 6, 3, −5, 4, 2	
2	4, 6, −1, 3	3, 0, 4, 1
1	5, 1	−1, −2
0	3	2

Table 2.1: Haar wavelet transformation of sequence s

The Haar wavelet transform of sequence s is as follows, where 3 is the single average element we got, and (2), (-1,-2), (3,0,4,1) are the difference values we got at resolutions 0, 1, and 2 respectively.

3	2	−1	−2	3	0	4	1

2.2.2 Discrete Fourier Transform

Performance of DFT when used with conjugate property is same as that of DWT. They only differ in the complexity of transformation. Since DFT can be used as alternative to DWT, more details about DFT is given below.

A Fourier transform is an operation which converts signals from time to frequency domain. The situation becomes more complicated if the data has an overall non-constant trend or includes noise. DFT is a special case of Fourier Transform, used in such cases where the data is not periodic i.e., DFT is used when both the time and frequency variables are discrete [Opp96]. In DFT the signal is considered as a periodic signal over one period [Opp96]. The discrete Fourier transform changes an N point input signal into two $N/2+1$ point output signals [Smi97]. The input signal contains the signal being decomposed, while the two output signals contain the amplitudes of the component sine and cosine waves. Moreover the first half of the DFT is the complex conjugate of the second part i.e., first half of the transformed signal is a mirror image of the second half. The term frequency domain is used to describe the amplitudes of the sine and cosine waves.

In typical problems, one seeks a representation of the signal, valid for $t \in [0, T]$ for period T for a sequence p of length N. Since there are N sample values, DFT gives a set of N equations for the unknown coefficients . For N samples or N unknowns, we want to determine N unknown coefficients $A_0; A_1, \cdots, A_{N/2}$ and $B_1, \cdots, B_{N/2-1}$ as:

$$f(t) = 1/2A_0 + \Sigma_{p=1}^{N/2}[A_p \cos(w_p, t) + B_p \sin(w_p, t)] \tag{2.1}$$

where the angular frequencies A_p and B_p is given by:

$$\begin{aligned} A_p &= 2/N \textstyle\sum_{n=1}^{N} y(n)\cos(2\pi(p-1)n/N); \quad for p = 1, 2, \cdots, N/2 \\ B_p &= 2/N \textstyle\sum_{n=1}^{N} y(n)\sin(2\pi(p-1)n/N); \quad for p = 1, 2, \cdots, N/2 \end{aligned} \tag{2.2}$$

The first sample A_0 of the transformed series is the DC component, more commonly known as the average of the input series.

$$A_0 = 1/N \sum_{n=1}^{N} y(n)$$

$$A_{N/2} = 1/N \sum_{n=1}^{N} y(n)\cos(n\pi)$$

$$B_0 = B_{N/2} = 0$$

For more details, interested readers are referred to [Opp96, Smi97].

2.3 APCA

Global dimensionality reduction techniques cannot capture the entire information of a time sequence in general since the intensity of a segment in a time series is not always the same. The performance of an index can be improved if a local dimensionality reduction is used instead. This is the key idea in using the APCA reduction technique which allows representation of segments of different sizes with minimized approximation error. To be more precise, given a time series $C =< c_1, \cdots, c_n >$, the APCA technique splits C into variable length segments, based on the data values, and represents the series as a collection C' of segments of the form $< cv_k, cr_k >$, where cv_k is the mean of values in the k^{th} segment and cr_k is the index of the last value in that segment. That is,

$$C' = (< cv_1, cr_1 >, < cv_2, cr_2 >, \cdots, < cv_M, cr_M >), \quad cr_0 = 0 \tag{2.3}$$

Length of the k^{th} segment can be calculated as $cr_k - cr_{k-1}$. Fig. 2.4 taken from [Keo01] shows an example of a time series represented by M APCA segments, for $(M = 4)$, in this example. Table. 2.2 gives the summary of notations used by APCA representation [Keo01].

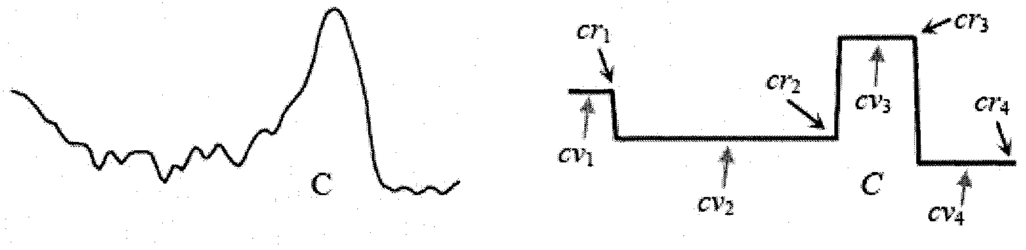


Figure 2.4: Time series with M = 4 and its APCA representation [Keo01]

2.3.1 APCA Representation

The algorithm works by first applying the Haar wavelet transform to the sequences in the database and converting the result to the APCA representation. The algorithm computes Haar wavelet in O(n) time and sorts the coefficients in decreasing order of their normalized magnitude and truncates the smallest coefficients. These values are used to reconstruct the signal and the segments may have approximate mean

Symbols	Definition
n	Length of time series
$C =< c_1, \cdots, c_n >$	Time series in the data base
N	Dimensionality of the index structure,N<n
M	No. of APCA segments, $M = \lfloor (N/2) \rfloor$
$C' = (< cv_1, cr_1 >, < cv_2, cr_2 >, \cdots, < cv_M, cr_M >)$, $cr_0 = 0$	C' is APCA representation of C with cv_i having the mean value of segment and cr_i has the index value of the last element of the segment
$D(Q, C)$	Euclidean distance
$DLB(Q', C')$	Lower bounding approximation of Euclidean distance
$cmax_i, cmin_i$	Min and Max values of the i^{th} segment of the APCA representation
$C = ((< cmin_1, cr_1 >, \cdots, < cmin_M, cr_M >), (< cmax_1, cr_1 >, \cdots, < cmax_M, cr_M >))$	MBR of a point C
$R = (L, H)$ where $L = (l_1, l_2, \cdots, l_N)$ and $H = (h_1, h_2, \cdots, h_N)$	MBR associated with a node with Lower(L) and Higher(H) boundaries
$MINDIST(Q, R)$	Minimum distance between the query time series Q and MBR R
$MINDIST(Q, R, t)$	Minimum distance between MBR R and query Q at time t

Table 2.2: Summary of the notations used by APCA representation [Keo01]

values. There are two more issues that should be considered for this approach:

1. DWT is defined only for time series whose length is 2^a for any integer value a. If the length of time series is not of this form, then the sequence is padded with zeros and the corresponding segments are then truncated from HWT, after the transformation.

2. Only the largest M coefficients are then retained after DWT is applied and

if the reconstructed signal has more than M segments, adjacent segments are repeatedly merged until we get M segments. The segments are merged in such a way that the increase in reconstruction error is minimal. This is explained later.

APCA construction of a signal/series is performed after obtaining the HWT of the signal. The following example illustrates the steps of computing the APCA representation for the time series $C = < 7, 5, 5, 3, 3, 3, 6, 4 >$ of length $n = 8$ and consider the index value starts with 1. Suppose $M = 3$ is the number of APCA segments to be represented by this series. The APCA algorithm needs HWT of C to get the APCA representation. APCA representation of a sequence is obtained through the following steps.

Example for Computing APCA(C):

Resolution	Average	Difference	Normalization value
3	$7, 5, 5, 3, 3, 3, 6, 4$		
2	$6, 4, 3, 5$	$1, 1, 0, 1$	$1/2(1/\sqrt{2})^2$
1	$5, 4$	$1, -1$	$1/\sqrt{2}(1/\sqrt{2})^1$
0	4.5	0.5	$1(1/\sqrt{2})^0$

Table 2.3: Haar wavelet transform of C

1. Table. 2.3.1 shows the computation of DWT for the given C. The Haar coefficients of C are $H_C = (4.5, 0.5, 1, -1, 1, 1, 0, 1)$.

2. Multiply Normalization coefficients with Haar coefficients to get normalized coefficients. $H_C = (4.5, 0.5, 1/\sqrt{2}, -1/\sqrt{2}, 1/2, 1/2, 0, 1/2)$ are the normalized

Resolution	Average	Difference
0	4.5	0
1	4.5, 4.5	1, −1
2	5.5, 3.5, 3.5, 5.5	

Table 2.4: Reconstruction of the signal

Haar coefficients of C. Without normalization, the output coefficients would have successively greater energy and in case of a multi-resolution transform (such as wavelets), transformation is done more than once which would lead to much larger coefficients in the transform domain.

3. By sorting the coefficients in decreasing order of the magnitude, we get $H_C =< 4.5, 1/\sqrt{2}, -1/\sqrt{2}, 0.5, 1/2, 1/2, 1/2, 0 >$, in which the last $N - M$ haar coefficients are replaced with zeros. This yields $H_C =< 4.5, 0, 1, -1, 0, 0, 0, 0 >$.

4. Signal is reconstructed as shown in Table. 2.4. This reconstruction of signal continues until the number of elements in the reconstructed signal is at least M. In this case, we terminate the process at resolution 2 (resolution gives power of 2 indicating the number of elements at that point).

5. Approximated values of reconstructed signal <5.5, 3.5, 3.5, 5.5> are replaced by the exact values <6, 4, 3, 5>. For this, either ceiling or floor of these values is considered randomly.

6. Merge the second and third segments since the difference between them is minimum, with the mean value 3.5. We now obtained the 3 segments <6, 3.5,

5>.

7. Represent each segment with its mean value and the index of the last element in the segment. This gives $C' =< 6, 2 >, < 3.5, 6 >, < 5, 8 >$ as the APCA representation of C.

The algorithm for creating APCA representation is provided as follows [Keo01]

Algorithm APCA Construction(C, M):

Input: Time Series C, Number of Dimensions M;

Output: APCA representation of C;

S_1. Verify the length of C and if it is not a power of 2, pad C with zeros until its length is of this form.

S_2. Compute Haar wavelet transform of C.

S_3. Sort the coefficients in the decreasing order of the normalized magnitude and mark all the coefficients except the first M as zero.

S_4. Reconstruct APCA approximation of C from the retained coefficients.

S_5. If reconstructed sequence C is padded with zeros, truncate it to the original length and replace the approximated values with exact values in the signal resulted by reconstruction.

S_6. while the number of segments is greater than M.

```
    $S_{6.1}$. Merge adjacent segments whose difference is minimal
  $S_7$. endwhile
  $S_8$. Find the mean values for each segment and store it with the index value of
      the last element.
End APCA Construction;
```

2.3.2 Bounding Regions

While the APCA representation of a time series C is obtained and calculated as described, the boundaries of this point is computed as follows. Fig. 2.5 [Keo01] shows a time series with four segments. It also shows the minimum and maximum values of the segments used to define the boundaries when creating the index. In the figure, values $cmin_i$ and $cmax_i$ represent the minimum and maximum boundaries

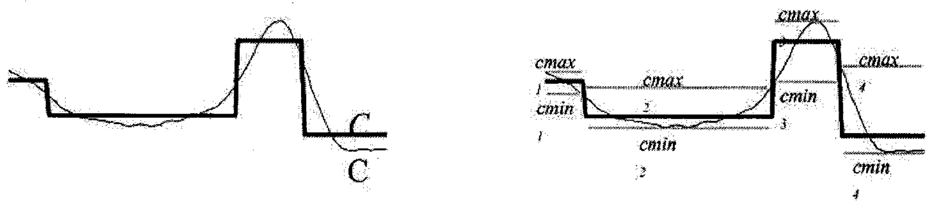


Figure 2.5: Boundaries of a time series C [Keo01]

of the i^{th} segment in C. Then the minimum bounding region(MBR) of the sequence is given by $R = (L, H)$ where $L = (l_1, l_2, \cdots, l_N)$ represents the lower boundary and $H = (h_1, h_2, \cdots, h_N)$ represents higher boundary for a region/ point. The bounding region of a point C is defined as

$$\bar{C} = ((< cmin_1, cr_1 >, \cdots, < cmin_M, cr_M >), \tag{2.4}$$

$$(< cmax_1, cr_1 >, \cdots, < cmax_M, cr_M >))$$

where cr_i is the index value of the last element of the segment. Suppose R is the MBR of a leaf node U. For every time series C inserted into U, the bounding region is computed as follows:

$$cmin = \mathbf{min}_{t=cr_{i-1}+1}^{cri}(C_t) \tag{2.5}$$

$$cmax = \mathbf{max}_{t=cr_{i-1}+1}^{cri}(C_t)$$

$$l_i = \begin{cases} \mathbf{min}_{C \in U} cmin_{(i+1)/2} & if\ i\ is\ odd \\ \mathbf{min}_{C \in U} cr_{i/2} & if\ i\ is\ even \end{cases} \tag{2.7}$$

$$h_i = \begin{cases} \mathbf{max}_{C \in U} cmax_{(i+1)/2} & if\ i\ is\ odd \\ \mathbf{max}_{C \in U} cr_{i/2} & if\ i\ is\ even \end{cases}$$

The following example [Keo01] shows the computation of MBRs.

Example for Computing Bounding region

1. Let $A =< 4, 6, 1, 0, 2 >$ and $B =< 4, 3, 5, 1, 3 >$ be two time series.

2. 2-segment APCA representations of A and B are given by $A = \{< 5, 2 >, < 1, 5 >\}$ and $B = \{< 4, 3 >, < 2, 5 >\}$, respectively.

3. Now the boundaries of A is given by $\bar{A} = ((< min\{4,6\}, 2 >, < min\{1,0,2\}, 5 >$ $), (< max\{4,6\}, 2 >, < max\{1,0,2\}, 5 >))$ and

   $\bar{B} = ((< min\{4,3,5\}, 3 >, < min\{1,3\}, 5 >), (< max\{4,3,5\}, 3 >,$

   $< max\{1,3\}, 5 >))$

   That is, $\bar{A} = ((< 4,2 >, < 0,5 >), (< 6,2 >, < 2,5 >))$ and $\bar{B} = ((< 3,3 >$ $, < 1,5 >), (< 5,3 >, < 3,5 >))$.

4. Assume that both $\bar{A}$ and $\bar{B}$ are in the same node. Then the MBR R of A and B is the smallest rectangle that spatially contains $\bar{A}$ and $\bar{B}$, and R is computed as $((min(4,3), min(2,3), min(0,1), min(5,5)),$

   $(max(6,5), max(2,3), max(2,3), max(5,5)))$

   $= ((3,1,0,5), (6,3,3,5))$.

They also define the distance measures D_{LB} and MINDIST to support the new representation [Keo01]. D_{LB} helps in finding the distance between the query Q and APCA representation of a point in C. This measure finds the APCA representation of Q with respect to the end points of each segment of the APCA representation of C. That is, Q is segmented in the same way as C, and if both are similar, then the distance will be close to zero. Fig. 2.6 shows the segmenting of Q with respect to C taken from [Keo01]. D_{LB} defines the distance between these two points as follows:

$$D_{LB}(Q', C) = \sqrt{\sum_{i=1}^{M} (cr_i - cr_{i-1})(qv_i - cv_i)^2} \tag{2.9}$$

$$Q' = < qv_1, qr_1 >, < qv_2, qr_2 >, \cdots, < qv_M, qr_M > \tag{2.10}$$

$$where \quad qr_i = cr_i$$

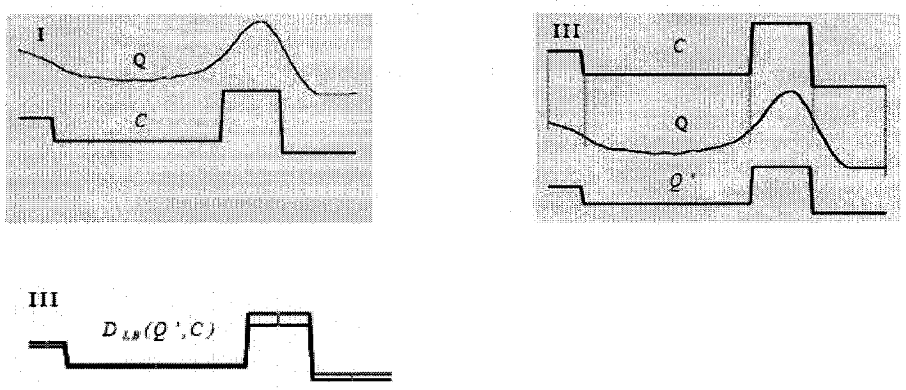


Figure 2.6: (I) Query Q and APCA representation of C. (II) Query Q segmented with respect to the end points of C. (III) D_{LB} as the square root of sum of the product of squared length with length of the segments they join [Keo01].

MINDIST is used to measure the distance between the query Q and MBR R of an internal node. There are two types of MINDIST functions. The first one finds the distance between Q and R whereas the other one finds the distance between Q and R at a particular instant of time. The MINDIST measures are defined as:

$$MINDIST(Q,R) = \sum_{i=1}^{n} MINDIST(Q,R,t) \tag{2.11}$$

$$MINDIST(Q,G,t) = \begin{cases} G[1] - q_t)^2 & if \ q_t < G[1] \\ q_t - G[3])^2 & if \ G[3] < q_t \\ 0 & otherwise \end{cases} \tag{2.12}$$

where G_i^R is the 2 dimensional rectangular region associated with R that fully contains the i^{th} segment of all time series points in U.

$$\begin{aligned} G_i^R[1] &= l_{2i-1} \\ G_i^R[2] &= l_{2i-2}+1 \\ G_i^R[3] &= h_{2i-1} \\ G_i^R[4] &= h_{2i} \end{aligned} \tag{2.13}$$

The proposed index structure uses the APCA approximation and the distance measures proposed originally by Mehrotra et al. [Keo01]. As introduced earlier we use these two distance measures D_{LB} and $MINDIST$ in our work, one to determine the distance D_{LB} between the query and a data item in a leaf node in the index, and the other to determine the distance $MINDIST$ between the query and minimum bounding rectangles (MBR) of the tree, which is an internal node. For more details, interested readers are referred to [Keo01].

Chapter 3

Related Work

In this chapter we review similarity matching and different indexing techniques on time series data. For this we start with queries and need of index on time series data in section 3.1. A brief description of use of similarity match is also given in this section. In section 3.2 we present a review of available indexing techniques from oldest to the newest, describing their benefits and limitations. R-tree is discussed in detail (section 3.3) as it is part of our indexing technique. Our discussion of MRI would also be in detail as our work enhances this index structure (section 3.4). The impact of some parameters such as different dimensions, threshold and query length on the index are also studied in this chapter.

3.1 Similarity Matching

It is a challenge in time series to find patterns which are similar to an input pattern. These queries involve specification of both a query pattern and a range of allowable similarity. The traditional sequential scan is inadequate for this purpose because of the data size. The demand to support efficient query processing over such data is growing fast [Keo00, Keo01], and a number of indexing techniques have been proposed to address the challenge for efficiency. These techniques are discussed below. A desired index should occupy less space and return the result with less number of disk access (I/O operations) as this is the major factor that affects efficiency of the index.

Range queries and nearest neighbour queries for whole matching and subsequence matching are principal queries of interest in time series data. Whole matching corresponds to the case where the query sequence and the sequences in the database are of the same length. Subsequence matching is a different but related problem, where the query sequence in general is much shorter than the sequences in the database. For example, consider the query: "Find the days in which stock X had the same value as the value of X today(Closing price)". The solution to answer this query is to check every sequence in the database.

The use of variable length queries to find similar patterns in the database is obvious, however based on the performance of the index in terms of precision and disk I/O's, there is no best solution for such queries till now [Kah04]. Multi-resolution Index (MRI) solves this problem to some extent but has some drawbacks such as memory requirement, low precision, and speed. Our goal in this work is to increase the precision of MRI for variable length queries and improve its memory utilization by avoiding insertion of redundant subsequences into the index. In our proposed indexing technique, we use APCA representation for dimensionality reduction to create index. Furthermore, we suggest that using an R-tree/I-adaptive index structure at every resolution is sufficient to get the result of a query, increase precision and speed, and decrease memory requirement. However, unlike in MRI, for this indexing technique, there is no need for any compression in our technique as the index size is much less compared to MRI [Kah04].

3.2 Indexing Techniques on Time Series

Given a query sequence Q, problem is to find similar patterns from the database. For accurate and efficient answering of these queries we need an index on time series. Time series databases use high dimensional index structures such as Grid files [Nie81, Nie84], R-Tree [Gut84], R*-Tree [Bec90], I-adaptive Index [Fal94], KD-Tree [Den95], and VP-Tree [Tol99]. All these indexes partition either the data or the space into regions in a way that makes it easier to restrict the search to only those regions which potentially contain "good" matches.

Guttmann introduced R-tree structure to deal with multi-resolution data [Gut84]. Faloutsos et al. [Fal94] modified the insertion algorithm of R-tree to tightly pack the nodes in order to solve the matching problem for fixed length queries. This was called the I-adaptive index, which divides a given data sequence into sub-trails based on a marginal cost, and represent each of these trails in a Minimum Bounding Rectangles (MBRs). Marginal cost of the bounding regions is calculated as follows before inserting a node:

1. Let $L = L_1, L_2, ..., L_n$ be the lengths of the bounding regions along the n-dimensions.

2. $A = \sum_{i=1}^{n}(L_i + 0.5)$ gives the area of bounding region.

3. Marginal cost $mc = A/k$, where k is the number of points in the MBR.

Whether a node should be inserted into this MBR or not is decided based on the marginal cost value. They also proposed Prefix search and Multi-piece search for variable length queries. Prefix search performs a database search using the prefix of the query whose length is equal to the length of the sequences for which the index is built. Multi-piece search splits the query sequence into non-overlapping subsequences of fixed length and performs fixed length queries for all these subsequences.

Li et al. [Li 00] proposed Skyline index structure to improve the performance of dimensionality reduction techniques, such as APCA, by representing a group of related time series data according to their collective shape, known as Skyline Bounding Regions (SBR). They define tight lower bounding distance based on SBR, and hence reduce the number of index pages accessed and the number of data objects retrieved. Even with all these improvements, the performance of these indexes degrades for variable length queries.

As a solution to this problem, Kahveci and Singh [Kah01, Kah04] proposed a Multi-Resolution Index together with corresponding search methods to solve range and nearest neighbour queries. This index is known as multi-resolution index (MRI) because all the sequences in the database are stored in the index at different resolution levels. The MRI index is a grid structure with each row recording the sequences at a particular resolution and each column recording a particular sequence at different resolutions. This index is created using different window sizes for every sequence and by picking the length of the sequence just greater than the highest resolution in the index. As is the case in other indexes, the length of the sequence considered at each row is a power of 2 (2^i), for every integer i, where $a \leq i \leq b$.

The Search algorithms split the query into non-overlapping subqueries in such a way that every subquery has its resolution present in the index. Range search algorithm searches the index for every resolution of the subquery and refines the threshold used after each search. The experiment results in [Kah04] indicate that their method is 4 to 20 times faster than all other techniques. For K-Nearest neighbour (KNN) search, they first find the approximate distance for the k^{th} neighbour, and then using this distance as the threshold, they perform exact range search to find the K nearest neighbours (KNN).

Since the size of MRI index is large, they also used compression techniques to reduce the index size. Sun et al. [Sun03] improved compression of MRI through Virtual Bounding Rectangle (VBR), which they called it as Compressed MRI.

VBRs are index nodes containing and approximating MBRs, and can be represented compactly. Their results show that VBRs save 80% – 93% on storage utilization compared to MBRs. They also optimized the search algorithm which improves the pruning power of the MRI index proposed in [Kah04]. In our experimental results, we compare performance of their search algorithm on our CMRI.

The next two sections discuss R-tree and MRI in detail as our technique uses R-tree/I-adaptive index and enhances MRI. Since the structure of I-adaptive index is same as R-tree and optimization of I-adaptive index is explained above, it is not discussed further.

3.3 R-Tree Index Structure

As a solution to handle multi dimensional data, Guttman proposed a multidimensional index structure known as R-tree [Gut84]. An R-tree is a height-balanced tree similar to a B-tree with index records in the leaf nodes containing pointers to data objects. The structure of R-tree is designed so that a spatial search requires visiting only a small number of nodes. It differs from B-tree in that each node in the R-tree is multidimensional. The root of an R-tree is a node which covers the entire region of the database and has pointers to sub regions or children. Each child in turn covers a sub region that leads to a set of leaf nodes which contain pointers to specific regions. The depth (or levels) of R-tree is not fixed and grows as the data grows. We remark that while a hash index cannot handle range searches, a B-tree index handles range searches only in a single dimension. On the other hand, R-trees can handle range searches on multi-dimensional data.

For example, if an R-tree index is built on an attribute of type point, queries such as *"List all points within a bounding rectangle"* can be answered more efficiently.

A spatial database consists of a collection of data representing spatial objects, and each object has a unique identifier and a pointer which can be used to retrieve it. Let M be the maximum number of entries that can fit in one node and let $m = M/2$ be a parameter specifying the minimum number of entries in each node. An R-tree satisfies the following properties [Gut84]:

1. Every leaf node in the index contains between m and M index records unless it is the root.

2. For each index record in a leaf node, I is the smallest rectangle that spatially contains the n-dimensional data object represented by the indicated tuple.

3. Every non-leaf node in the index contains between m and M children unless it is the root.

4. For each entry in a non-leaf node, I is the smallest rectangle that spatially contains the rectangles in the child node.

5. The root node has at least two children unless the tree has only one node.

6. All leaf nodes appear on the same level.

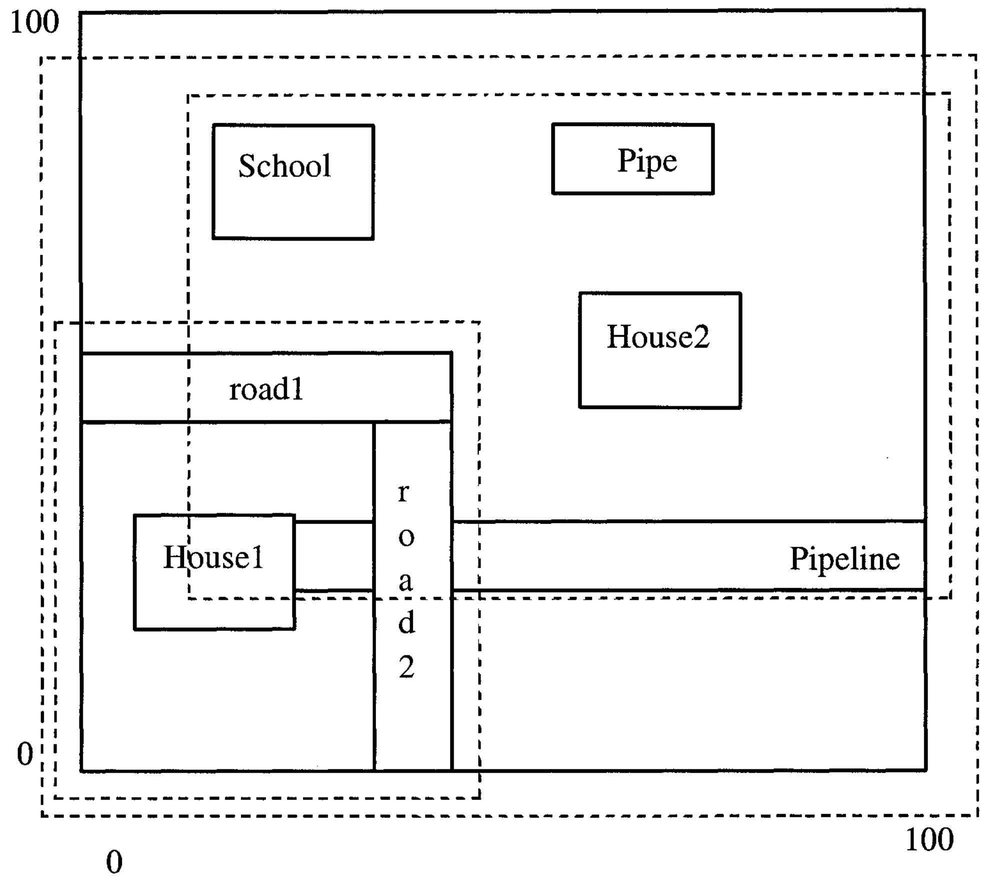


Figure 3.1: Two dimensional representation of 7 regions [Mol02]

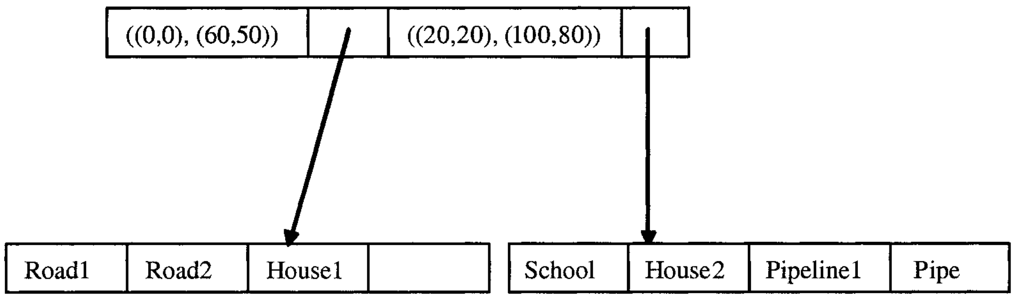


Figure 3.2: Relative R-Tree index to the 7 regions [Mol02]

Fig. 3.1 shows 7 regions in a two dimensional space, and Fig. 3.2 shows how these regions are grouped and indexed into an R-tree, both taken from [Mol02]. The first figure also illustrates the containment and overlapping relationships among the regions. The regions are grouped in such a way that there is minimal increase in the area of an internal node, and splits the leaf node if there is no room. The R-tree is a hierarchical tree structure with nodes at different levels of the tree. Performance of an R-tree index structure for queries is roughly proportional to the area and perimeter of the index nodes. The area covered at level 0 represents the area occupied by the minimum bounding rectangles of the data geometries, the area at level 1 indicates the area covered by leaf-level R-tree nodes, and so on. The original ratio of the area at the root (top most level) to the area at level 0 can change over time based on updates to the dataset. If there is a degradation in that ratio (that is, if it increases significantly), restructuring the index may be required to improve its performance. The only disadvantage of R-tree is that the bounding regions of the nodes in the tree might overlap [Sub98]. Thus, the search on R-tree might lead to different paths increasing the disk I/Os.

For spatial data, there are two important types of basic queries:

1. Range query: Given a query region, find all objects which intersect this region.

2. Nearest neighbour query: Given an object, find its k-nearest neighbours.

We can use an R-tree index to answer these basic queries efficiently. A spatial R-tree index can index spatial data and approximates each geometry by a single rectangle, called the minimum bounding rectangle (MBR), that minimally encloses the geometry. For a layer of geometries, an R-tree index structure consists of a hierarchical index on the MBRs of the geometries in the layer. Let Q be the query rectangular region. Then an R-tree search algorithm starts at root, finds the nodes intersecting with Q, searches the subtrees of these nodes for intersecting nodes, and returns the set of leaf nodes (regions) which intersects with the query rectangle. These points are then extracted to find the exact match with the query.

Before closing this section, it should be mentioned that any column of a table or relation can be used for indexing, if desired. For example, in stock market application, for every stock data we have information on date, values of high, low, close, and the volume. However, we build index only on the closing price, which is the key attribute to discover the trends of time series.

3.4 Multi-Resolution Index (MRI)

MRI is a multi-dimensional index structure that supports range and nearest neighbour searching for variable length queries originally proposed in [Kah04]. It uses multi-dimensional indexes at several resolutions, which is constructed as follows. For sequences $(s_1, s_2, \cdots, s_n)$, MRI stores a grid of trees $T_{i,j}$, where i ranges from resolution a to b, and j ranges over the sequences s_1 to s_n. For this, MRI stores MBRs corresponding to database sequences at different resolutions. More precisely, tree $T_{i,j}$ is the collection of MBRs for window size $w = 2^i$ corresponding to sequence

s_j. The tree is an I-adaptive index, where each MBR is extended to tightly cover the next transformed sequence of length w $(= 2^i)$ until the marginal cost increases, in which case a new MBR will be created. Fig. 3.3 shows the structure of the MRI index, taken from [Kah04]. A separate tree $T_{i,j}$ is built for the MBRs of each sequence s_j at different resolutions i. The i^{th} row of the index is represented as the collection $R_i = \{T_{i,1}, \cdots, T_{i,n}\}$, and the j^{th} column of the index is represented as the collection $C_j = \{T_{a,j}, \cdots, T_{b,j}\}$.

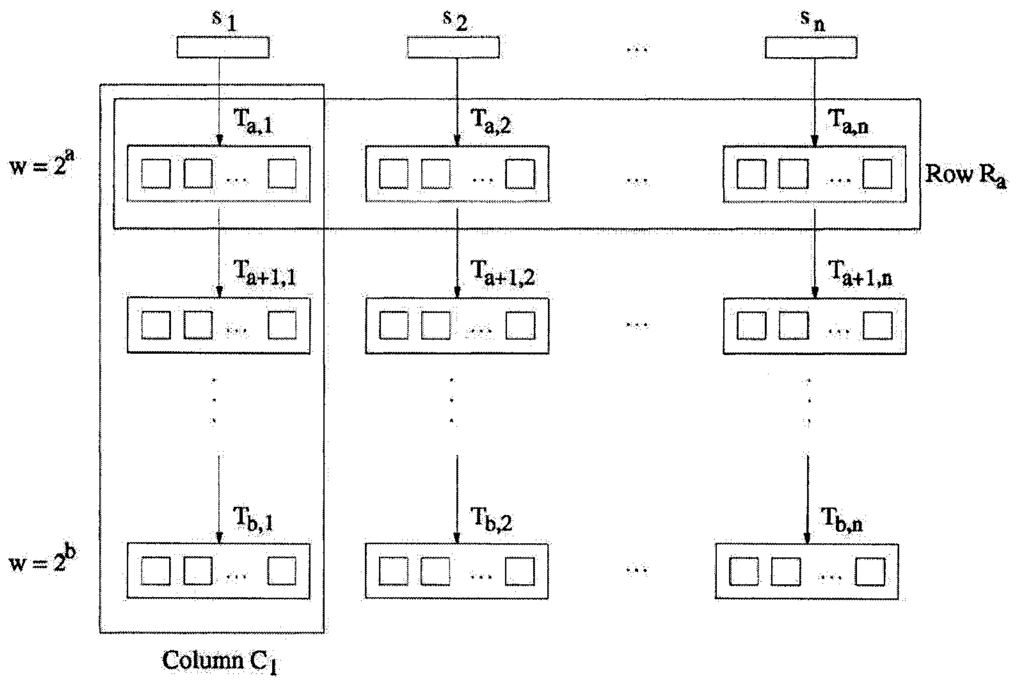


Figure 3.3: Structure of MRI [Kah04]

3.4.1 Query Partitioning

Tamer and Singh also proposed the query partitioning algorithm to support variable length queries on their index structure. Given a query Q of any arbitrary length, the search algorithm partitions the query in a unique way into various subqueries whose lengths match one of the resolutions in the index. The longest prefix of Q whose length is a multiple of minimum window size is considered if the length of Q is not already a multiple of minimum window size. For example, assume that the length of Q is 250 and the minimum and maximum size of window in index is 4 to 8. Then in the search, the largest multiple of 16 (2^a) less than 250, i.e., 240, would be considered as the length of the prefix of Q. But before the search begins, this query is partitioned in the increasing order of the length of subsequences as $q = q_1, q_2, q_3, q_4$, where $q_1 = 16$, $q_2 = 32$, $q_3 = 64$, and $q_4 = 128$. Now the search is performed on each of these subqueries at their respective resolutions on the index and the results from each subquery are combined to give the final result of query Q.

Partition Algorithm($qlen$):

Input: Query length $qlen$;

Output: Partition lengths of $q_1, q_2, \cdots$;

```
$S_1$. Take the largest multiple of $2^a$ less than query length, call this as $qlen$.

$S_2$. Let $len = \lfloor qlen/2^a \rfloor$.
```

```
S_3. Get the binary representation of $len$.

S_4. for every value of 1 in the binary representation, get a partition of length =
     $2^{a+i}$, where $i$ is the position of 1 in the binary representation.

     S_4.1. if $a + i > b$, replace 1 with two partitions of length $2^{a+i-1}$; else replace 1
            with a single partition of length $2^{a+i}$. Repeat this step while the length
            of partition is still greater than $2^b$.

     end for

End Partition Algorithm;
```

Let us consider the following example to illustrate this algorithm. Let Q be a query of length 750. Assume the index has resolutions from 4 to 8. Then partition of q is obtained as follows. We consider a prefix of length 736, since it is the largest multiple of 16 (2^a, where a is the lowest available resolution 4) less than 750. Since we want the minimum length of each partition to be 16 (2^4 or 2^a), we divide the prefix length with 16, divide 736 with 16, which gives 46. Binary representation of 46 is 101110. The array given below shows the binary representation and positions.

Binary	1	0	1	1	1	0
Position	5	4	3	2	1	0

The following partitions are obtained after the first iteration. Notice that each partition is multiplied by 2^4, which is the minimum length of the query.

Partition1	2^9	2^7	2^6	2^5

Following the algorithm, the result after the second iteration of the partitioning is

Partition2	2^8	2^8	2^7	2^6	2^5

Finally we partition the query into subqueries of lengths 32, 64, 128, 256, and 256. Sum of length of all partitions must be equal to the prefix length, which in this case is 736.

3.4.2 Range Query

Range query is the search for sequences that fall within a range (distance/threshold) from the query sequence. For range query, a threshold value is provided to get close enough sequences that match the query. In MRI, range search algorithm starts by partitioning the query sequence Q of any length into a number of subqueries as $q' = q_1q_2 \cdots q_t$, where q' is the prefix of q whose length $|q'|$ is the largest integer j such that $|q| \geq 2^j$, and that $|q_i| = 2^{c_i}$, for $a \leq c_i \leq b$, and $1 \leq i \leq t$ at different resolutions available in the index structure. A partial range query is then performed on all the partitions of the query at the corresponding row of the index structure. The threshold value is updated for range search on the next partition after each partial range query. This process is repeated until the last partition and the subsequences are retrieved based on the candidate set returned.

3.4.3 Nearest Neighbour Query

For a nearest neighbour query, we search for k closest subsequences to the query sequence, where k is the number of neighbours given as input by the user. In MRI, nearest neighbour search is performed in two phases. First, it takes the longest prefix of q whose resolution appears in the MRI index structure and finds the k closest MBRs. The sequences are then read from these MBRs to find the actual distance, and the k^{th} smallest distance (d) is considered as the threshold value for the next phase. Only a small percentage of data is processed at this stage. In the second phase, a range query is performed using the threshold value d on index for query Q. It is guaranteed that k nearest neighbours are retrieved in this phase because distance to actual k^{th} neighbour is at most d. In our work, we use the same algorithm for nearest neighbour search. A modification of these two algorithms is given in next chapter. For more details about these search algorithms interested readers are refered to [Kah04].

3.4.4 Limitations of MRI

Limitations of MRI are summarized as follows.

1. MRI is a complex grid structure.

2. The structure gets complicated as the number of sequences increases.

3. There are redundant sequences inserted into the index.

4. The search algorithm is time consuming as it goes through every sequence.

5. It needs compression to reduce the index size.

6. Precision/accuracy is good when compared to existing techniques, however it is not the best and could be further improved.

3.5 Parameters Affecting Performance

The following are the parameters that play a vital role in the performance of an index.

Dimensions

Number of dimensions used for the index plays an important role in improving performance of the index. The optimum number of dimensions needed for any spatial multidimensional index on a particular database has to be experimented considering the memory consumption, speed, and performance of the index. So before fixing the dimensions, one must experiment by creating the index at different dimensions as the performance of the index is very much data dependent. In our experiments, we evaluate the performance of the index at different dimensions.

Threshold

Threshold is the radius given for the range queries. The performance of the index also depends on the threshold value of the input query. Given a query, a threshold value is also provided by the user. As the threshold value increases, searching through index results in accessing large amount of data. Large number of results is not useful to predict events and so it is the responsibility of the user to provide proper threshold value.

Query Length

Query length also plays an important role in finding similar patterns in time series databases. As the query length increases, there is a higher chance of increase in the distance between similar sequences and hence the threshold value should be larger. Nearest neighbour queries are useful in finding K most similar patterns to the query, if there is no proper result using range search. In case of range search, if the query length is small, then a slight increase in the threshold might lead to increase in the result set as it is more likely to find similar patterns for smaller length queries.

Chapter 4

Our Proposed Index

In the previous chapter, we studied the MRI index structure proposed in [Kah04], and discussed its drawbacks and limitations. One major drawback is the index size, when the data is very large; the MRI index may not fit in the main memory and hence compression techniques are required to reduce the index size. To address this, the authors of MRI proposed compression techniques to reduce the index size while preserving the information content. A question which arises at this point is that whether we could redesign MRI such that we continue to enjoy its capabilities while reducing its space utilization without requiring any compression?

In our investigation of this issue, we found the answer to be positive. In what follows, we introduce a multi-resolution indexing technique as an alternative to MRI which improves MRI in several aspects including precision/accuracy and speed, in addition to the index size. We use APCA as a local dimensionality reduction technique for our index structure, which is responsible for increase in precision. We

Sequence	RMRI	MRI
s_1: 1,2,$\cdots$,68	1-4,2-5,$\cdots$,65-68	1-4,2-5,$\cdots$,65-68
s_2: 2,3,$\cdots$,69	66-69	(2-5,3-6,$\cdots$),66-69
s_3: 3,4,$\cdots$,70	67-70	(3-6,4-7,$\cdots$),67-70

Table 4.1: Indexing sequence file A using RMRI and MRI index structures

use an intermediate index structure called Reduced MRI (RMRI, for short) as a basis to establish the correctness of our technique, called Compact MRI (CMRI) [Sri06]. Section 4.1 explains RMRI which avoids the redundancy of MRI and justifies that a simple tree at every resolution is sufficient instead using multiple trees as in the proposed MRI structure [Kah04]. CMRI uses a tree at every resolution and can be directly constructed by the algorithm given in section 4.3.1.

4.1 Reduced Multi-Resolution Index Structure

Reduced MRI (RMRI) is a simplified structure obtained from MRI by avoiding the redundant insertions of sequences done by MRI. We illustrate this redundancy by an example. Consider a time series database with two datasets A and B, where A includes integers from 1 to 1000 and B includes multiples of 10 from 10 to 10000. Suppose the minimum length of query is 4 and the MRI index is created for resolutions 4 to 64. Also suppose the sequence length is 68, which lies between 2^6 and 2^7.

Tables. 4.1 and 4.2 shows some nodes in the MRI and RMRI index structures at resolution 4 for the datasets A and B, respectively. The first table shows the

Sequence	RMRI	MRI
s_i: 10,20,$\cdots$,680	10-40,20-50,$\cdots$,650-680	10-40,20-50,$\cdots$,650-680
s_{i+1}: 20,30,$\cdots$,690	660-690	(20-50,30-60,$\cdots$),660-690
s_{i+3}: 30,40,$\cdots$,700	670-700	(30-60,40-70,$\cdots$),670-700

Table 4.2: Indexing sequence file B using RMRI and MRI index structures

initial three subsequences s_1, s_2, s_3 in A after being inserted, where x-y indicates the values x to y. The second table shows this for the initial three values in B. Each sequence inserted into the index is separated by comma. These tables show the index after insertion of the first three subsequences in datasets A and B, with MRI column showing the inserted nodes of MRI structure and RMRI column showing the inserted nodes by eliminating the redundant subsequences.

As can be seen from these tables, MRI inserts some redundant sequences, indicated in parentheses in the MRI column. Insertion of such sequences unnecessarily increases the size of the MRI index. In RMRI, we do not index repeated sequences shown in parentheses, which improves space and time over MRI for storing and searching operations. This also explains the name RMRI for this technique. As a result, RMRI essentially follows the structure of MRI in which, for every sequence, we have just one node at every resolution except for the first sequence in the database as shown in the tables. Besides, RMRI uses a local dimensionality reduction technique, as opposed to a global dimensionality reduction technique employed in MRI.

Construction of RMRI proceeds as follows. First we use APCA as the local

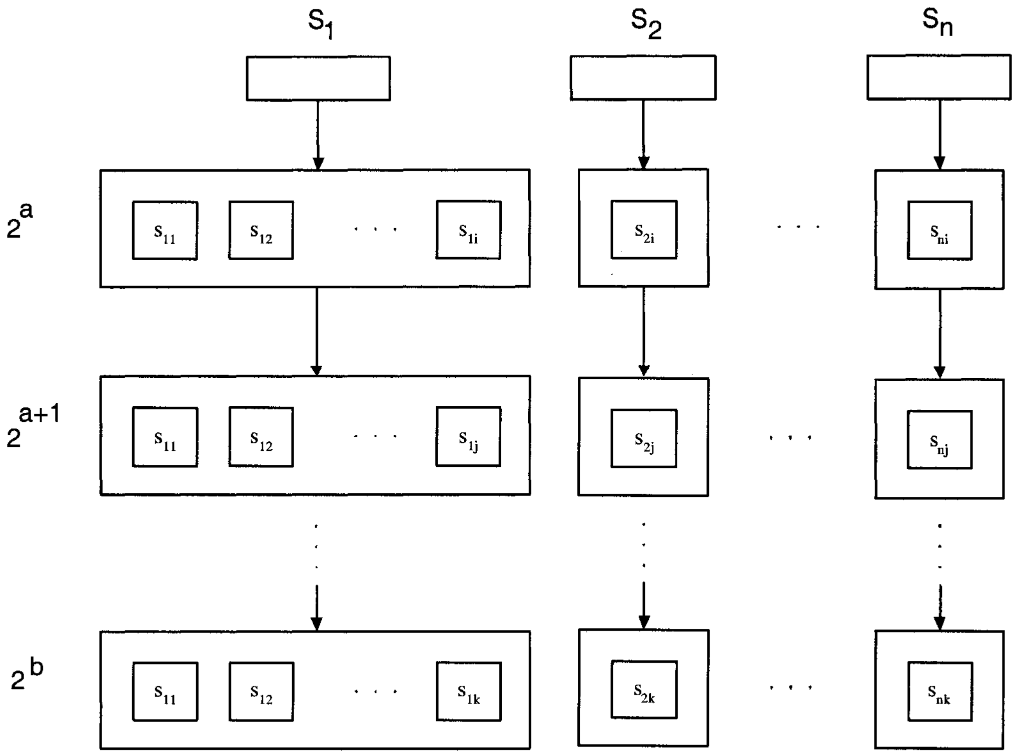


Figure 4.1: Structure of Reduced MRI (RMRI)

dimensionality reduction technique, which maps a small subset of points in an M dimensional space to points in $2M$ dimensional space. Similar to MRI, our index is created for resolutions a to b, described as follows. Let s be the longest sequence in the database, where $2^b \leq |s| \leq 2^{b+1}$ and the total number available sequences in database are n. Each sequence from s_1 to s_n have i, j, k subsequences of length 2^a, 2^{a+1} and 2^b. We use the minimum possible length of a query to be 2^a, for some integer a, where $a \leq b$. We apply Haar wavelet transformation (HWT) for every sequence s in the database. The result of this transformation of sequence

s is the used to create the APCA representation of s, as suggested in [Keo01], to find variable length segments. For each sequence s in each dataset, an I-adaptive index is created for all the subsequences of s except the redundant subsequences from resolutions a to b. The resulting index is the RMRI, whose structure is shown in Fig. 4.1. The correctness of the RMRI index follows from the correctness of the APCA representation, which is an established dimensionality reduction technique.

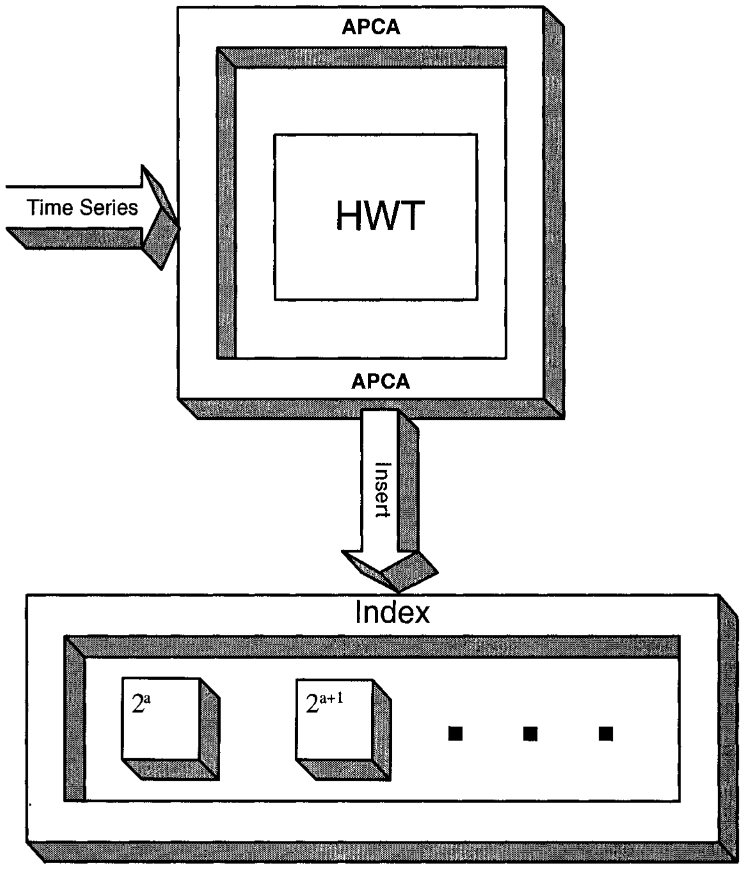


Figure 4.2: Construction of CMRI

4.2 Construction of CMRI

We use the background knowledge acquired from the techniques explained in previous chapter to construct our index CMRI. Fig. 4.2 shows the details of our construction technique. The figure clearly shows the steps performed on time series before it is inserted into the index. APCA representation of the time series is obtained by first applying Haar Wavelet Transform (HWT) on the time series. Bounding region of each sequence is obtained from its APCA representation. These regions are now inserted into the index at the corresponding resolution. This process is repeated until all the sequences in the database are inserted into the index.

4.3 Compact Multi-Resolution Index Structure

In addition to using APCA representation in RMRI, we investigated other possible ways to further improve the index features, including its size. From Tables. 4.1 and 4.2, we noted that the structure of the index nodes at every resolution for MRI could be further reduced to just one node for all data sequences except for the first sequence (as observed in tables), without loss of information. That is, the leaf nodes of the I-adaptive index in RMRI will have the same information as those in MRI. This ensures that there will be no false dismissals or false hits in using RMRI. We further combined the nodes at each resolution in RMRI to form an I-adaptive index. In other words, we have a single I-adaptive index at each resolution for the entire time series data. The resulting index structure is called *Compact MRI* (CMRI, for

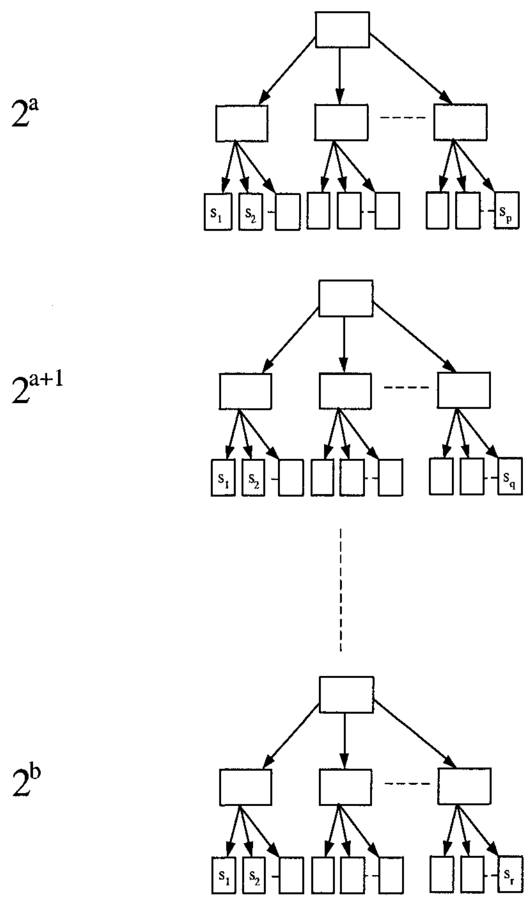


Figure 4.3: Structure of Compact MRI (CMRI)

short) and is shown in Fig. 4.3. All the sequences are in the leaf nodes of the tree. At resolution 2^a an I-adaptive index is created for sequences $s_1, s_2, \cdots, s_p$ where p is the total number of sequences of length 2^a in the database. In the same way q and r represent the total number of sequences in the database of length 2^{a+1} and 2^b.

We reiterate that the introduction of RMRI was just conceptual, as a bridge to establish correctness of CMRI with respect to MRI. Below we suggest a technique to construct CMRI directly by inserting nodes in an I-adaptive index at various resolutions. This is done in $O(n)$ for the simple structure of CMRI as opposed to the complex grid structure of MRI, where n is the number of sequences to be indexed. The construction of CMRI is formally stated as follows, where the database is the collection of the input time series data. Every sequence in the database is transformed using APCA representation and inserted into the I-adaptive index at every resolution.

```
Algorithm Create CMRI($D, a, b, M$):
Input: Dataset D, resolution range $a$ to $b$, and dimension $M$;
Output: M dimensional index structure for resolution a to b;
  $S_1$. for all sequence $s$ in D.
      $S_{1.1}$. for all resolutions of sequence $s$ from $a$ to $b$.
          $S_{1.1.1}$. Apply APCA transformation on $s$.
          $S_{1.1.2}$. Insert $s$ into $2M$ dimensional index tree at corresponding resolution
                  with pointer to location of $s$ in D.
End Create CMRI;
```

4.3.1 Range Query

We propose a range search algorithm for CMRI by adopting the corresponding algorithm proposed for MRI in [Kah04]. Instead of traversing the trees for each sequence as performed in MRI, CMRI traverses the I-adaptive index at a particular resolution to find similar patterns with all sequences in the database. An important consequence of the simple structure of CMRI is that its range search algorithm has just a single loop whereas it is a double loop in MRI. The search algorithm in MRI takes more time as the search proceeds to the next sequence after the current one is complete, whereas in CMRI, a search is performed only at corresponding resolution. Also, there is less chance to prune away a group of nodes at one time in MRI. In our case, we do not have to examine sequence by sequence to find similar patterns but rather we only need to traverse tree by tree based on the resolution. The above steps are formalized as the following range search algorithm for CMRI.

```
Algorithm RangeQuery CMRI ($q, \epsilon$):
Input: Query $q$ and threshold $\epsilon$;
Output: Result set which includes the time sequences which match $q$;

  $S_1$. Partition query $q$ of length $|q|$, where $|q| \geq 2^a$, into $p$ parts as $q_1, q_2, \cdots, q_p$
         in ascending order of their length such that $|q_i| = 2^{c_i}$, for $a \leq c_i \leq b$ and
         $1 \leq i \leq p$;

  $S_2$. Initialize $\epsilon_p = \epsilon$, where $\epsilon$ is threshold for range queries;

  $S_3$. For $i := 1$ to $p$:

      $S_{3.1}$. Perform query $q_i$ on tree $T_k$ at resolution $k$, where $k = log_2|q_i|$. Let $Res_i$
```

```
        be the result set for partition $i$;
    $S_{3.2}$. Set $\epsilon_{i+1} := max_{B \in Res_i}\{\sqrt{\epsilon_i^2 - d(q_i, B)^2}\}$, where $d(q_i, B)$ is the distance
        between query subsequence $q_i$ and bounding region B;
  $S_4$. Filter the results of each partition to find the match for $q$ of length $|q|$;
  $S_5$. Access disk pages indicated in $Res_p$ and perform post-processing to eliminated
      false retrievals.
End RangeQuery;
```

4.3.2 KNN Search

To find the k nearest neighbours to a query sequence, the longest prefix of the query whose resolution appears in the index is taken and distance is computed for the k closest MBRs to it. As in [Kah04], this distance is used as the threshold to find the exact k neighbours. The algorithm for KNN search proposed is given below by [Kah04].

```
Algorithm KNN-Search($Q$,$k$):
Input: Query $q$ and threshold $\epsilon$;
Output: Result set which includes the time sequences;

  $S_1$. Take the prefix of $Q$ in $q_1$ whose length appears in the index. Find the set of
      $k$ closest MBRs to $q_1$ from the index.
  $S_2$. Read the sequences contained in these MBRs and record the distances.
```

S_3. Use the distance of k^{th} neighbour as the threshold value to find the k neighbours using exact range search algorithm.

End KNN-Search;

The precision/accuracy of CMRI is identical to that of RMRI, since the information prevailed in the R-tree is the same as those stored in RMRI. This is also verified in our extensive experiments. On the other hand, the speed of CMRI is significantly improved since it eliminates groups of nodes from consideration in the search. The space utilization by RMRI and CMRI are almost the same, but much smaller compared to MRI for the same time series databases.

4.4 Memory Calculation

Index size plays a major role in measuring the effectiveness of an index. Usually the size of an index is greater than data size. Memory occupied by our index is calculated in the following way. Since it uses an I-adaptive index structure to insert M dimensional data, the tree needs $2 \times 2M$ dimensions (2M: because of APCA representation) for lower and higher boundaries of index entries, k bytes per dimension, 4 bytes for node pointer. Therefore, a total of $(4kM + 4n)$ bytes per node of the tree is consumed, where n is the number of pointers for n children. For each data entry, it needs $(2kM + 4)$ bytes, where $2kM$ is the number of bytes for APCA representation of the data entry and 4 bytes for each pointer to the data in the database. Hence the total size of the tree in bytes would be:

$$IN(4kM + 4n) + DP(2kM + 4) \tag{4.1}$$

where IN is the total number of nodes in the index and DP is the number of data points in the tree. The same equation applies for MRI too. IN and DP values for MRI is higher when compared to CMRI because they insert redundant sequences into the index which reflects in the increase in IN and DP values. Hence memory consumed by MRI is more when compared to our technique.

When the space utilization is crucial, the index can also be created without storing the APCA representation of the data point. The APCA representation of the point can be obtained from the boundaries of the leaf node while processing. This has an additional cost of processing for not storing APCA representation.

Chapter 5

Experimental Results

In this chapter, we report our experimental results and illustrate the validity of our CMRI index and show superiority of our approach over existing techniques such as MRI and sequential scan for prefix search, multi-piece search, and CoMRI searching strategy. These results are discussed in detail in this chapter. For this, we implemented both CMRI and the MRI index in C++ as proposed in [Kah04]. We have used a PC with a Pentium 4 processor running at 2.80 GHz, 1 GB RAM, 140 GB disk space, on Windows XP. The buffer size considered for disk I/Os was 8 KB. We used a template code for R-tree in C++ is written by G. Douglas and is available at `www.serious-code.net/moin.cgi/RTree`. This is a modification or generalization of original code proposed by Guttman to apply on any kind of data type (int, char, float, user-defined class). Code for Haar and other wavelets is also available at `www.bearcave.com/misl/misl_tech/wavelets/packet/download.html`. Code for allocating memory blocks for the index is also obtained from the same package.

Data Type	No.of Sequences	No.of Files	size
Real	$200,000$	201	1.53MB
Synthetic	$200,000$	271	2.60MB
AUSLAN	$1.4 million$	2090	20.4MB

Table 5.1: Details about TSD used in our experiments

APCA representation and MRI are implemented from the papers [Keo01, Kah04]. In these experiments we have studied and compared MRI and CMRI on precision/accuracy, efficiency, and memory utilization using real and synthetic time series data. Details about the data collection we used in these experiments can be found in Table 5.1.

For the real data, we collected stock market information from the Yahoo web site at `http://finance.yahoo.com`. This data includes about 200,000 sequences in 201 collections/stocks with total size of 1.53 MB. Table. 5.2 shows the stocks for which data is collected in alphabetical order. For the synthetic data, we generated 200,000 sequences in 271 collections/files with total size of 2.60 MB, where each file is approximately 10KB. Synthetic data creation was done by generating, for each collection, a random seed $x = random(200)$, which was then altered by introducing some noise to generate other values $x' = x + random(5)$ in the collection.

A study of effectiveness of the index when the data size increases is performed using AUSLAN data from `http://kdd.ics.uci.edu/`. The database have approximately 1.5 million sequences. The dataset consists of sample Australian sign language with 2565 signs of various lengths. Out of these 2090 files of size slightly more

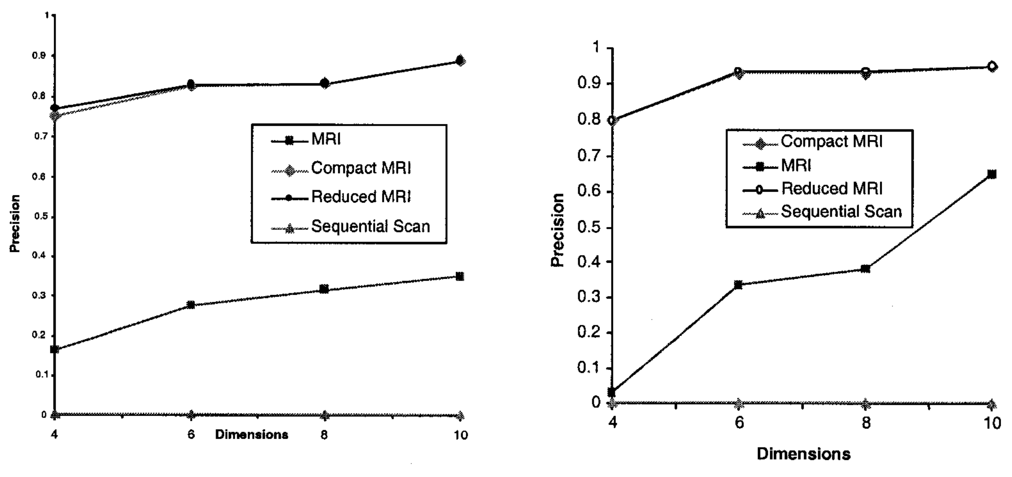
Alphabet	Stocks
a	a, aa, aap, absp, acip, acmr, adam, aem, age, agup, ahcp, amat, ambpm, aos, atgp, atp, atx
b	b, ba, baxp, bb, bcr, bdnpd, bfip, bfo, bgf, bkpe, blvp, bmi, bmyp, bosc, bscpe, bscpf, bwep, bylp, bscpg
c	c, cas, cc, ccbl, cde, cerp, cffi, chp, chrw, clppd, cmh, cof, cpf, cpg, crepe, cxmp
d	d, dd, ddpa, ddpb, ddrpf, ddrpg, decc, dhi, dkhr, dpa, dpu, dxpd, dys
e	e, ecmv, ee, eeln, egppd, eon, eoppg, epny, eqrpd, eqrpe, et, exxa, exxb, ezm
f	f, fbppd, fbppe, fcxpc, ff, flsh, fmfc, fnmpf, fnmpg, fn-mpm, fpa, fps, frepd, fun
g	g, gabpd, gbppd, ge, gept, gg, giii, gis, glrpa, glrpb, gmtp, grtpf, grtpg, gsupg
h	hba, hcnpd, hcnpf, hcppe, hcppf, hh, hiwpd, hmepf, hmtpe
i	idsy, idtc, ibm, intc, intl, it
j	jill
k	k, kimpf, krbpa, krbpd, krbpe, krcpe, krtpe, kt
l	l, lehpd, lehpe, lehpf, lg, lm, lmt, lrt, ltcpe, ltcpf
m	mbt, merpe, mfw, mho, mmm, mrk, msft, mtm, mvt, mwav, mlspe
n	n, navpd, nt
o	ofcpe, ofcpf, ofcpg, ohipd, orcl, ovti
p	paa, pcg, pdfs, pfcb, pfin, pg, pg1, pkypd, pp, psapf, pt, pv
r	rbspd, rbspe, rbspf, rbspm, riop, rfr
s	s, sfipd, sfipe, shlm, shupd, sjm, sdopc, sdopb, sdopa, sdoph, spgpf
t	t, tdsc, tech, tom
v	vnope, vcp
w	wilcf, wmt, wripe
y	yhoo

Table 5.2: Stock market data used in our experiments

than 20MB are considered for the experiments. Several experiments are performed to study the effect of MRI and CMRI for range queries on large datasets. Queries of variable length from 16 to 1024 are considered for the experiments.

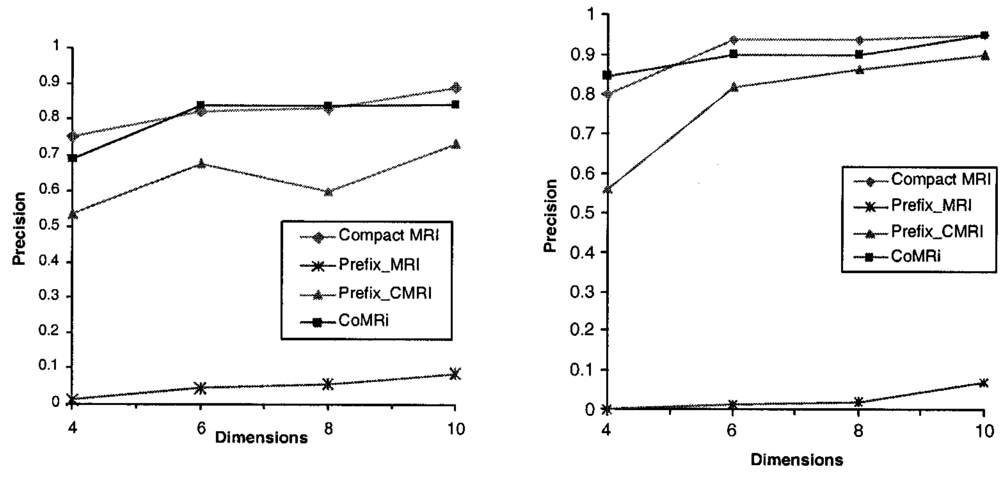
The lengths of queries are assumed to be evenly distributed. For all our experiments on range queries, resolutions of the index considered were in the range from 4 to 8. The experiments are limited to queries of varying lengths ranging from 16 to 1024. Each query sequence was generated by randomly selecting a sequence from the database and modifying about 10% of the sequence. As in [Kah04], the results considered are the average for 100 queries. These parameters and assumptions are considered for all our experiments on MRI, RMRI, and CMRI. We compare CMRI with the basic MRI technique and with the search technique of compressed MRI (CoMRI) proposed in [Sun03], which improves memory utilization of MRI at a slight cost for precision and some overhead for compression. This chapter provides details of these experiments and the results obtained. We also consider the prefix search and compare the results with our range query experiments, which shows that prefix search is not better than multi-piece search.

We have considered range and nearest neighbour queries of variable lengths in our experiments, details of which are provided in the following sections. All the graphs are accompanied with the tables showing the numerical values for each technique.



(a) (b)

Figure 5.1: Precision for Range queries on Real data (a) and Synthetic data (b)



(a) (b)

Figure 5.2: Precision for Range queries on Real data (a) and Synthetic data (b)

Dim	MRI	CMRI	RMRI	Seq.scan	PrefixRI	PrefixCMRI	CoMRI
4	0.16	0.75	0.77	0.005	0.01	0.54	0.69
6	0.28	0.83	0.83	0.005	0.046	0.68	0.84
8	0.32	0.83	0.83	0.005	0.06	0.6	0.84
10	0.35	0.89	0.89	0.005	0.09	0.74	0.84

Table 5.3: Precision for Range Queries on Real Data

Dim	MRI	CMRI	RMRI	Seq.scan	PrefixMRI	PrefixCMRI	CoMRI
4	0.03	0.8	0.77	0.005	0.0002	0.56	0.69
6	0.34	0.93	0.83	0.005	0.015	0.82	0.84
8	0.38	0.93	0.83	0.005	0.02	0.86	0.84
10	0.65	0.95	0.89	0.005	0.07	0.9	0.84

Table 5.4: Precision for Range Queries on Synthetic Data

5.1 Range Queries

For range queries, we considered resolutions of the index ranging from 4 to 8 (4,5,6,7,8). The queries considered were also of varying lengths ranging from 16 values to 1024. For each query sequence, we randomly produced the radii in the range [0.1, 1] as in MRI. The reason for choosing this range is in a way to standardize the comparison of indexing techniques. We assume that the lengths of queries are evenly distributed between 16 and 1024. As for the number of dimensions chosen from the sequence to build the index, we have considered four different dimensions: 4, 6, 8, and 10. The aforementioned parameters, assumptions, and measures are valid in all our experiments with MRI, RMRI, and CMRI. The assumptions and parameters are not required/made for sequential scan – a well-known query processing technique. The experimental results show that precision and number of disk I/Os of our techniques (RMRI and CMRI) are far better than MRI, precision is defined in section 1.2. Precision is the ratio of number of actual matches to query in database to the number of matches returned by index, another definition.

The graph in Fig. 5.1 shows precision for the four techniques, RMRI, CMRI, MRI, and sequential scan at different dimensions on both real data (a) and synthetic data (b). Since RMRI and CMRI have the same precision, their graphs in the figure coincide, even the disk I/O's are same. Because of this, in the rest of this report, we will only consider CMRI in our experimental results and comparisons. As shown in Fig. 5.1, while the precision of MRI ranges from 0.16 to 0.34 on real data and 0.03 to 0.65 on synthetic data, the precision of CMRI is far better, ranging from 0.75 to 0.89 on real data and 0.80 to 0.95 on synthetic data. Sequential scan has the least precision and most number of disk I/Os, as it scans the entire database.

Dim	MRI	CMRI	RMRI	Seq.scan	PrefixMRI	PrefixCMRI	CoMRI
4	14.59	2.03	2.03	201	34	3.68	1.16
6	10.81	1.41	1.4	201	25.2	1.56	1.11
8	12.25	1.7	1.68	201	28.48	1.73	1.21
10	13.12	1.07	1.03	201	20.54	1.22	1.25

Table 5.5: Disk I/Os for Range Queries on Real Data

Dim	MRI	CMRI	RMRI	Seq.scan	PrefixMRI	PrefixCMRI	CoMRI
4	7.88	1.14	1.14	271	10.61	1.25	1.07
6	4.99	1.03	1.03	271	10.12	1.07	1.06
8	5	1.1	1	271	9.19	1.02	1.01
10	3.28	1.01	1	271	7.23	1	1.02

Table 5.6: Disk I/Os for Range Queries on Synthetic Data

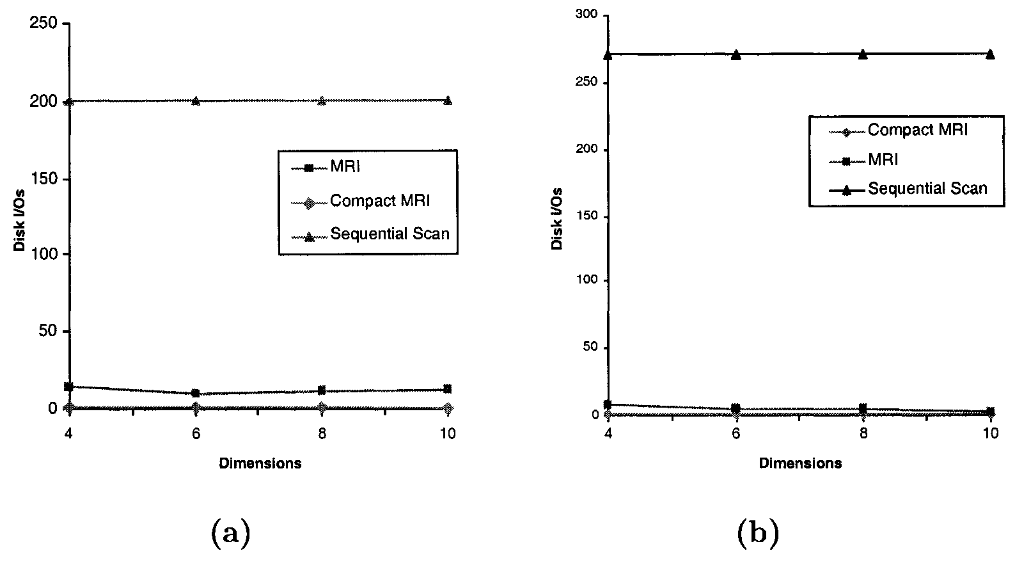


Figure 5.3: Disk I/Os for Range queries on Real data (a) and Synthetic data (b)

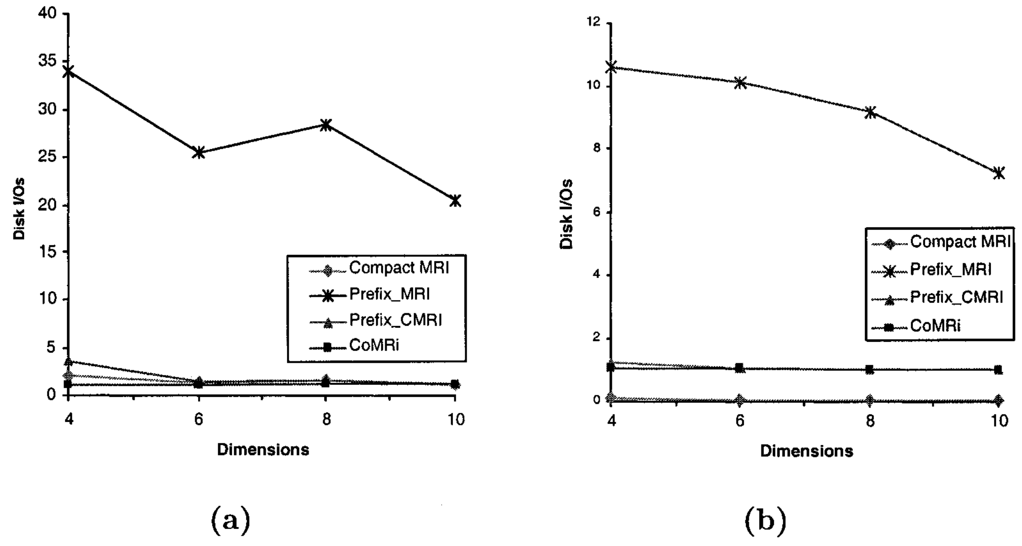
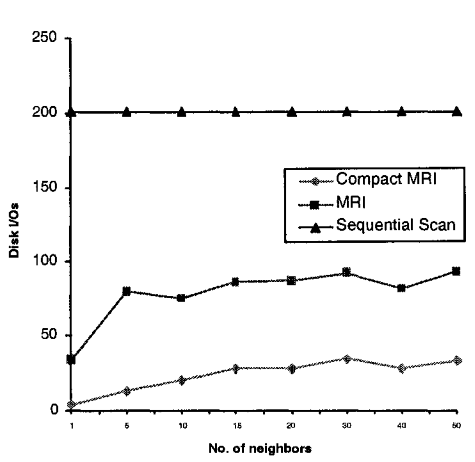
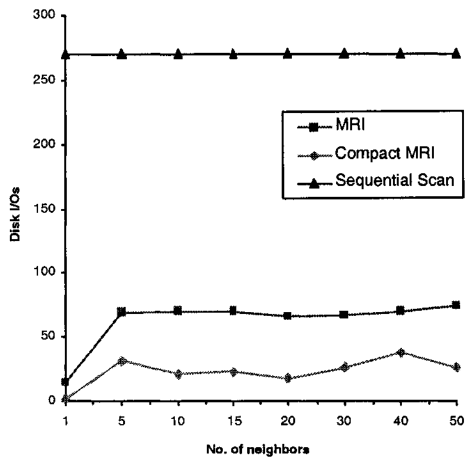


Figure 5.4: Disk I/Os for Range queries on Real data (a) and Synthetic data (b)

The same set of experiments are performed on MRI and CMRI with prefix search and optimized search method from CoMRI. The results are shown in Fig. 5.2. The results for these experiments are shown in different graphs to have a more clear presentation of their behavior. In the figure, prefix search over MRI and CMRI is represented by Prefix_MRI and Prefix_CMRI. Prefix search on MRI yields poor precision, almost close to zero, whereas with CMRI, precision of prefix search is far greater than precision of multi-piece search with MRI. Precision of prefix search on CMRI is close to precision of multi-piece search on CMRI. The CoMRI search algorithm treats multi-piece search in descending order of the length of subqueries and has almost the same performance as MRI's search algorithm. The multi-piece search is done in ascending order of the length of subqueries on our index CMRI.

Figs. 5.3 and 5.4 show the number of disk I/Os performed by multi-piece search and prefix search on CMRI and MRI, the search algorithm by CoMRI, and sequential scan, at different dimensions for the same range queries. As we can see, CMRI outperforms MRI by performing fewer disk accesses both on real data (a) and synthetic data (b). The graphs also indicate that CMRI has noticeably greater pruning power than MRI. In case of real data, the number of disk I/Os for CMRI is almost identical to the number of sequences in the query result. For synthetic data, while the number of disk I/Os for MRI decreases as the number of dimensions increases, CMRI consistently requires very few disk I/Os at every dimension. Since sequential scan reads the entire database, the number of disk I/Os remains high and





(a) (b)

Figure 5.5: Disk I/Os for nearest neighbour queries on Real data (a) and Synthetic data (b)

equals to the number of pages read for the entire database. These figures clearly show that multi-piece search has higher precision than prefix search. The number of disk I/O's for CoMRI, prefix search on CMRI, and multi-piece search on CMRI are similar as their graphs almost coincide. We thus exclude prefix search and CoMRI in our experiment results and comparisons.

K	MRI	CMRI	Seq.scan
1	4.09	33.55	201
5	13.1	79.05	201
10	20.36	75.43	201
15	28.79	85.76	201
20	28.54	86.46	201
30	35.05	92.28	201
40	28.83	81.34	201
50	34.23	92.46	201

Table 5.7: Disk I/Os for Nearest neighbour Queries on Real Data

K	MRI	CMRI	Seq.scan
1	2.06	14.71	271
5	31.37	68.67	271
10	20.36	69.44	271
15	22.71	70.02	271
20	17.78	65.80	271
30	25.80	66.24	271
40	37	69.46	271
50	26.36	74.2	271

Table 5.8: Disk I/Os for Nearest neighbour Queries on Synthetic Data

Dimension	MRI	CMRI
4	621	32
6	743	46
8	890	60
10	1276	73

Table 5.9: Memory occupied by index on stock market data at different dimensions

5.2 Nearest Neighbours Queries

Nearest neighbour queries are performed by varying the number of neighbours. Even though both CMRI and MRI have highest precision at dimension 10, we consider dimension 8 as optimal for search because the increase in performance from dimensions 8 to 10 is not much, considering the effect of the memory utilized and the search speed. Hence we fixed the number of Haar coefficients to be 8, that is, the number of dimensions considered for creating the index is 8. Experiments are performed on the indexes MRI and CMRI at dimensionality 8 for number of neighbours $K = \{1, 5, 10, 15, 20, 30, 40, 50\}$ on real and synthetic data. We also considered sequential scan for the comparison. Number of disk I/Os performed by

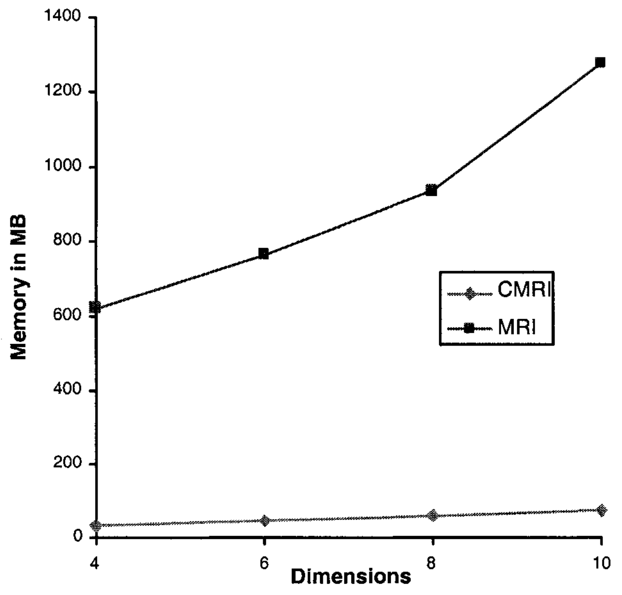
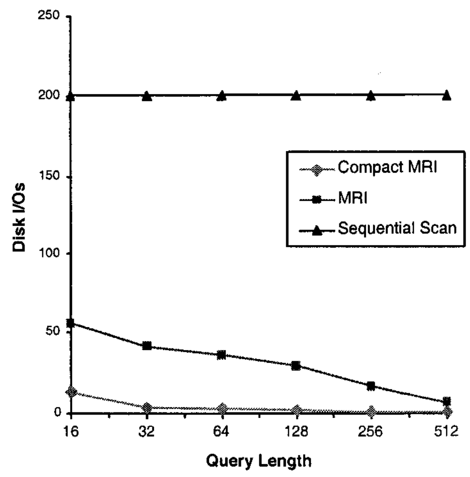
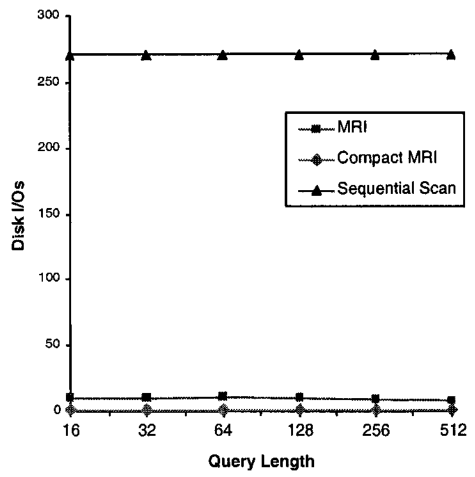


Figure 5.6: Memory occupied by index on stock market data at different dimensions

these techniques is used as a comparison basis. Fig. 5.5 shows the number of disk I/Os by CMRI, MRI, and Sequential scan for nearest neighbour queries on real and synthetic data. While the number of disk I/Os in MRI ranges from 33 to 92 for real data (a), it ranges from 4 to 35 for CMRI. That is, the maximum disk I/Os consumed by CMRI is close to the minimum disk I/Os consumed by MRI. Also for the synthetic data shown in (b), CMRI outperforms MRI significantly. On the synthetic data, the number of disk I/Os for MRI ranges from 14 to 75, while it ranges from 2 to 26 for CMRI. For the sequential scan, the number of disk I/Os required is the same as the number of blocks occupied by real and synthetic data.



(a)



(b)

Figure 5.7: Disk I/Os for Range queries of variable lengths on Real data (a) and Synthetic data (b)

Query Length	MRI	CMRI	Seq.scan
16	55.76	13.04	201
32	42.01	3.64	201
64	36.13	2.66	201
128	28.97	1.7	201
256	16.79	1.15	201
512	6.73	1	201

Table 5.10: Disk I/Os for variable length queries on Real Data

5.3 Memory Consumption

Reducing the memory occupied by the index is one of the goal in our research work. We reduced the index size by avoiding the insertion of redundant sequences. Figure. 5.6 and Table. 5.1 cleary shows the memory consumption of MRI and CMRI for stock market data. In the graph every value is an average of total memory occupied by the 5 indexes (from resolution 4 through 8) created for dimensions

Query Length	MRI	CMRI	Seq. scan
16	10.46	1.11	271
32	10.3	1.03	271
64	10.97	1.1	271
128	10.43	1.01	271
256	8.91	1.01	271
512	8.2	1.01	271

Table 5.11: Disk I/Os for variable length queries on Synthetic Data

Query Length	MRI	CMRI
16	1215.65	504.38
32	1200	344.68
64	1159.14	197.1
128	1134.45	165.18
256	1125.65	59.12
512	104.4	1.76
1024	68.25	1.2

Table 5.12: Disk I/Os for variable length queries on AUSLAN Data

{4,6,8,10}. The graph shows memory occupied by the index in Mega bytes on Y-axis, and the corresponding dimension is shown on X-axis.

5.4 Performance on Variable Length Queries

We have also evaluated the effectiveness of CMRI, MRI, and Sequential scan for variable length queries on real and synthetic data. For this set of experiments, query length is limited to $L = \{16, 32, 64, 128, 256, 512\}$ for all these techniques at dimension 8. Figure 5.7 illustrate the results of these experiments. They show the number of disk I/Os performed for each of these techniques on real and synthetic data. Similar kind of experiments are performed on AUSLAN data and the results

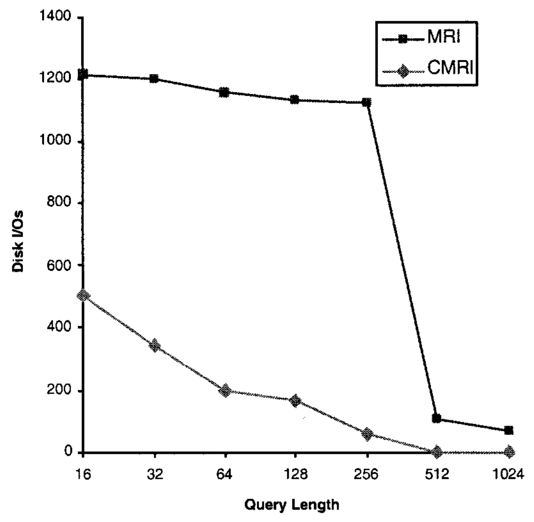


Figure 5.8: Disk I/Os for Range queries of variable lengths on AUSLAN data

are shown in Fig. 5.8. Effect of variable length range queries is studied on this data for queries of length {16,32,64,128,256,512,1024}. As can be seen, CMRI outperforms MRI with at least one order of magnitude, and outperforms Sequential scan with two order of magnitudes on both kinds of data. This indicates that the pruning power of CMRI is significant. While performance of MRI is better on synthetic data, our CMRI is one order of magnitude better than MRI. Moreover, as observed in the above experiments, the performance of CMRI is consistent and similar on both kinds of datasets – real and synthetic.

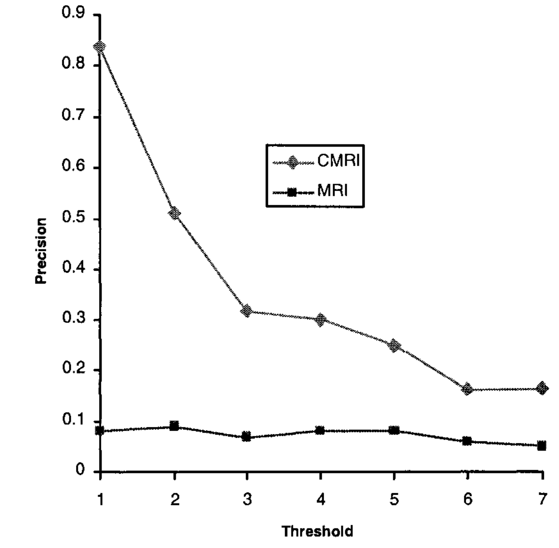


Figure 5.9: Precision for Range queries on Real data for different thresholds

Threshold	CMRI	MRI
1	0.84	0.08
2	0.51	0.09
3	0.32	0.07
4	0.3	0.08
5	0.25	0.08
6	0.16	0.06
7	0.17	0.05

Table 5.13: Effect of threshold on MRI and CMRI for Real Data

5.5 Effect of Threshold

The above experiments clearly showed that CMRI is superior to MRI. We conducted a different set of experiments aimed at studying the behavior of the index at different threshold values for range queries. Fig. 5.9 shows this for the threshold values {1,2,3,4,5,6,7}. Results are obtained on indexes at dimension 8. >From the figure, it is clear that performance of the index degrades as the threshold value increases.

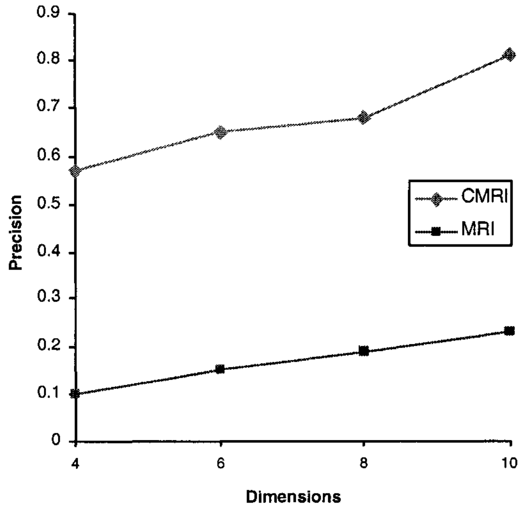


Figure 5.10: Precision for Range queries on mean value Index

Dimensions	MRI	CMRI
4	0.1	0.57
6	0.15	0.65
8	0.19	0.68
10	0.23	0.81

Table 5.14: Effect of mean value on MRI and CMRI for Real Data

Precision is high at threshold 1 and the least at threshold value 7. From the figure we can observe that performance of MRI, compared to CMRI, is poor for every threshold value. These results once again indicate superiority of the index structure, CMRI, proposed in this work.

5.6 Effect of Mean Value Indexing

In addition to the above experiments, extensive experiments are conducted to study the effect of mean value indexing on CMRI and MRI by removing the mean value from each sequence including the query sequence. This strategy is helpful when it is desired to invest in stocks having similar trends but may not have the same values. Through this approach, the index helps to identify matching not only with sequences having the same values but also with sequences having the same trend. This approach is tested on real data. Fig. 5.10 shows the performance of our index at different dimensions after removing the mean values from each sequence. The graph shows the precision of CMRI and MRI at different dimensions {4,6,8,10}. While the precision of CMRI ranges from 0.57 to 0.81, the precision of MRI ranges from 0.1 to 0.23 for range queries. The graph clearly shows that CMRI outperforms MRI.

5.7 Summary

In this chapter we studied and compared the performance of CMRI on variable length queries for range and nearest neighbours queries with MRI. We also compared the performance of CMRI with prefix search and CoMRI search. We also studied effect of varying parameters like dimensions, threshold and query length on CMRI and MRI. We also studied behaviour of CMRI and MRI when the index is created by removing the mean value from the sequences. The graphs clearly show that CMRI performs better than MRI.

Chapter 6

Conclusions and Future Work

This chapter includes concluding remarks and possible future research directories.

6.1 Conclusions

We studied the problem of efficient indexing for variable length queries. Multi-resolution Index (MRI) is proposed for the support of variable length queries on TSDB. However it has a complex grid structure that occupies such a huge memory which needs to be compressed. Even though MRI provides better solution than previous indexing techniques, we identified different opportunities to improve MRI in various aspects, including memory utilization and the index structure, both of which effects its efficiency. The best precision achieved through MRI is 35%, which provided more scope to improve MRI structure.

Considering all these aspects, we first implemented and examined the structure and performance of MRI, and noticed that MRI introduces redundant sequences in its index. It also has the limitation that the number of sequences inserted into the index should be known. The same grid structure could be created by avoiding the insertion of redundant sequences into the index structure. We called the result of this first step of modification of MRI as Reduced MRI (RMRI). For this technique, we also used APCA as local dimensionality reduction technique for the database. Through this step we achieved memory reduction and increased precision. Furthermore, to improve the speed of index, the complex grid structure was replaced by a simple I-adaptive index for every resolution. This resulted in our indexing technique, called Compact MRI (CMRI) as a better solution to support variable length queries for time series data. We used RMRI as a basis for correctness of our CMRI, RMRI (and hence CMRI) retain the essential MRI structure. Using this as a basis, we introduced a direct construction of CMRI with a simple I-adaptive index tree at every resolution as a far better alternative to the complex structure of MRI.

Unlike MRI which keeps track of the number of sequences, or the number of trees (when compressed) in its index, and uses this information during a search, our CMRI index does not record or use this information due to its simple structure. An immediate consequence of this reduced size of CMRI is that it does not require any compression. Dimensionality reduction technique plays an important role in improving the precision of CMRI. Note that the APCA representation is used as a dimensionality reduction technique for CMRI and RMRI, and uses $2M$ dimensions for every M Haar coefficients considered. As a result, the precision of CMRI shown in our experimental results ranges from 0.75 to 0.89 on real data, which is 2 to 3 times better than that of MRI. All these optimizations result in a much less space utilization by CMRI which is 5% of the size as compared to MRI – a significant reduction. This in turn resulted in increased efficiency, reduced number of disk I/Os, and improved precision, as indicated by our experiment results. A summary of advantages of CMRI over MRI is provided as follows.

Advantages of CMRI over MRI

1. CMRI has a simple structure compared to the complex grid structure of MRI.

2. Performance of MRI purely depends on the number of underlying sequences and needs to track of the number of sequences in the database, whereas CMRI does not record or use the number of sequences in the database.

3. The search algorithm in CMRI is faster than MRI as it searches resolution by resolution rather than searching sequence by sequence done in MRI.

4. CMRI occupies much less memory compared to MRI. Size of CMRI is 5% of the size of MRI, as proposed in [Kah04].

5. CMRI does not require compression.

6.2 Future Work

Our research has resolved the drawbacks of MRI. However there are other optimization issues such as speed which could be further improved using more sophisticated buffering schemes. This is useful and essential when handling huge volumes of data. The basic indexing structure used by CMRI is I-adaptive index and improving it might help in increasing the speed. Speed of the index can also be improved with a new distance measure that tightly bounds the distance between query and a bounding region. Such a distance measure would not only help in improving the speed but also helps to improve the precision. Memory occupied by the index could be further reduced by not storing the APCA representation of data points. Implementing CMRI and some of the proposed improvements as components in modern DBMS would revolutionize data management in near future.

Disk I/Os performed in KNN search can be reduced by deploying new searching strategies. This can be achieved through a single phase search rather than with a two phase search.

Bibliography

[Aga93] Agarwal R., Faloutsos C., and Swami A. Efficient similarity search in sequence databases. In *Proc. Int'l Conf. on Foundation of Data Organization and Algorithms*, pages 69–84, October 1993.

[Bec90] Beckman N., Kriegel H.P., Schneider R., and Fu W. The R*-tree: An efficient and robust access method for points and retangles. In *Proc. ACM SIGMOD Int'l Conf. on Mangement of Data*, pages 322–332, 1990.

[Bel61] Bellman R. E. Adaptive control process: A guided tour. *Princeton University Press*, 1961.

[Cha99] Chan K. and Fu W. Efficient time series matching by wavelets. In *Proc. 15th Int'l Conf. on Data Engineering*, pages 126–133, 1999.

[Cha00] Chakrabarti K. and Mehrotra S. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proc. 26th VLDB Conf.*, 2000.

[Den95] Deng K. and Moore A. Multiresolution instance-based learning. In *Proceedings of the Twelfth International Joint Conference on Artificial Intellingence*, pages 1233–1239, San Francisco, 1995. Morgan Kaufmann.

[Fal94] Faloutsos C., Ranganathan M., and Manopolous Y. Fast subsequence matching in time series databases. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, pages 419–429, 1994.

[Fal95] Faloutsos C. and Lin King-Ip (David). Fastmap: A fast algorithm for indexing, data mining and visualization of traditional and multimedia datasets. In *Proc. ACM SIGMOD Conf.*, pages 163–174, May 1995.

[Fal96] Faloutsos C. *Searching Multimedia Databases by Content.* Kluwer Academic Publishers, 1996.

[Gut84] Guttman A. R-trees: An efficient indexing structure for spatial searching. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, pages 47–57, 1984.

[Kah01] Kahveci T. and Singh A. K. Variable length queries for time series data. In *Proc. Int'l Conf. on Data Engineering*, pages 273–282, 2001.

[Kah04] Kahveci T. and Singh A. K. Optimizing similarity search for arbitrary length time series queries. *IEEE Transactions on Knowledge and Data Engieering*, 16(4):418–433, April 2004.

[Kan98] Kanth K.V.R., Agrawal D., and Singh A. Dimensionality reduction for similarity searching in dynamic databases. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, June 1998.

[Keo00] Keogh E., Chakrabarti K., Mehrotra S., and Pazzani M. Dimensionality reduction for fast similarity search in large databases. In *Proc. of Knowledge and Information Systems*, pages 263–286, 2000.

[Keo01] Keogh E., Chakrabarti K., Mehrotra S., and Pazzani M. Locally adaptive dimensionallity reduction for indexing large time series databases. *Journal of the ACM*, May 2001.

[Li 00] Li Q, Lopez I.F.V., and Moon B. Skyline index for time series data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16:669–664, 2000.

[Mol02] Molina H. G., Ullman J. D., and Widom J. *Database Systems: The Complete Book*, pages 697–700. Prentice Hall, 2002.

[Nie81] Nievergelt J., Hinterberger H. and Sevcik K. C. The grid file: An adaptable, symmetric multi-key file structure. In *ECI*, pages 236–251, 1981.

[Nie84] Nievergelt J., Hinterberger H. and Sevcik K. C. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst.*, 9(1):38–71, 1984.

[Opp96] Oppenheim A. V., Willsky A. S., and Nawab S. H. *Signals & systems (2nd ed.)*, chapter 3,4,5. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[Pop02] Popivanov I. and Miller R.J. Similarity search over time series database using wavelets. In *Proc. 18th Int'l Conf. Data Engineering*, pages 212–221, 2002.

[Raf97] Rafiei D. and Mendelzon A.O. Similarity-based queries for time series data. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, pages 13–25, 1997.

[Raf98] Rafiei D. and Mendelzon A.O. Efficient retrieval of similar time sequences. In *Proc. of FODO Conf.*, pages 249–256, 1998.

[She80] Shepard R.N. Multi dimensional scaling. *Princeton University Press*, 1980.

[Smi97] Steven W. Smith. *The scientist and engineer's guide to digital signal processing*, chapter 8, pages 141–160. California Technical Publishing, San Diego, CA, USA, 1997.

[Sri06] Srividya K. and Shiri N. A compact multi-resolution index for variable length queries on time series datasets, 2006. Submitted for evaluation for Journal of Knowledge and Information Systems.

[Sub98] Subrahmanian V. S. . *Principles of Multimedia Database Systems*, pages 88–94. Morgan Kaufmann, 1998.

[Sun03] Sun H., Ozturk O, and Ferhatosmanoglu H. Comri: A compressed multi-resolution index structure for sequence similarity queries. In *IEEE Computer Society Bioinformatics Conference (CSB'03)*, pages 553–558, 2003.

[Tol99] Tolga Bozkaya and Meral Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Trans. Database Syst.*, 24(3):361–404, 1999.

[Wu 00] Wu Y.L., Divyakant A., and Abbadi A.E. A comparision of dft and dwt based similarity search in time series databases. In *Proc. Int'l Conf. on Information and Knowledge Management*, pages 488–495, 2000.

[Yi 00] Yi B. K. and Faloutsos C. Fast time sequence indexing for arbitrary lp norms. In *Proc. 26th Int'l Conf. on Very Large Databases(VLDB)*, 2000.