Multi-Resolution Indexing Method for Time Series

Mei Ma

Submitted in total fulfilment of the requirements of the degree of

Master of Computer Science (by research)(by Thesis only)

Department of Computer Science and Software Engineering
The University of Melbourne

January 2010
Abstract

Time series datasets are useful in a wide range of diverse real world applications. Retrieving or querying from a collection of time series is a fundamental task, with a key example being the similarity query. A similarity query returns all time series from the collection that are similar to a given reference time series. This type of query is particularly useful in prediction and forecasting applications.

A key challenge for similarity queries is efficiency and for large datasets, it is important to develop efficient indexing techniques. Existing approaches in this area are mainly based on the Generic Multimedia Indexing Method (GEMINI), which is a framework that uses spatial indexes such as the R-tree to index reduced time series. For processing a similarity query, the index is first used to prune candidate time series using a lower bounding distance. Then, all remaining time series are compared using the original similarity measure, to derive the query result. Performance within this framework depends on the tightness of the lower bounding distance with respect to the similarity measure. Indeed much work has been focused on representation and dimensionality reduction, in order to provide a tighter lower bounding distance.

Existing work, however, has not used employed dimensionality reduction in a flexible way, requiring all time series to be reduced to have the same dimension. In contrast, in this thesis, we investigate the possibility of allowing a variable dimension reduction. To this end, we develop a new and more flexible tree based indexing structure called the Multi-Resolution Index (MR-Index), which allows dimensionality to vary across different levels of the tree. We provide efficient algorithms for querying, building and maintaining this structure. Through an experimental analysis, we show that the MR-Index can deliver improved query efficiency compared to the traditional R-tree index, using both the Euclidean and dynamic time warping similarity measures.
Declaration

This is to certify that:

(i) the thesis comprises only my original work,

(ii) due acknowledgement has been made in the text to all other material used,

(iii) the thesis is less than 30,000 words in length, inclusive of footnotes, but exclusive of tables, maps, appendices and bibliographies.

Mei Ma, January 2010
Acknowledgements

Firstly, I would like to show my sincere gratitude to my supervisors, Dr. Rui Zhang and Dr. James Bailey. The work presented in this thesis would not have been possible without their support and assistance. I’m deeply influenced and encouraged, not only by their talents and skills in research, but also their care and characteristics. Throughout my two years’ research studies, Rui provided insightful discussions about the research. He improved my research skills and led me to the right research directions. James also gave me a lot of useful advices on my research subject, and tirelessly helped me with the thesis writing.

I would also like to thank all the staff and the students in the Department of Computer Science and Software Engineering. They have really created a nice environment. Working with them gave me a lot of fun.

Finally, I would like to thank my father and mother for their unfailing support. They gave me a lot of encouragement and suggestions whenever I have any difficulties in my life. It is their help and love that enabled me to conduct my research work with confidence and determination.
Contents

1 Introduction 1
   1.1 Time series and their applications .......................... 1
   1.2 Overview of existing methods .................................. 3
      1.2.1 Similarity measures for time series ....................... 3
      1.2.2 Representation methods for time series ...................... 5
      1.2.3 Indexing time series data ................................ 7
   1.3 Motivation ....................................................... 8
   1.4 Contribution and Organization .................................. 9

2 Preliminaries and Related Work 11
   2.1 Terminology ...................................................... 11
   2.2 Lower bounding methods for time series ......................... 12
      2.2.1 Euclidean distance ........................................ 12
      2.2.2 The Dynamic Time Warping (DTW) Distance ................ 20
   2.3 Similarity queries ............................................... 27
      2.3.1 Linear Scan ............................................... 28
      2.3.2 The GEMINI framework for time series data ............... 30
   2.4 Summary ......................................................... 39

3 Multi-Resolution Indexing 41
   3.1 MR-index Structure ............................................. 41
   3.2 Algorithms For Insertion and Search ............................ 48
      3.2.1 Insertion .................................................. 48
      3.2.2 Search .................................................... 51
   3.3 Index Configuration ............................................. 56
   3.4 Framework Extension ............................................ 61
   3.5 Summary ......................................................... 64
## 4 Experimental Evaluation

4.1 Datasets ........................................ 65
4.2 Multiple Optimal Dimensionalities ................. 67
4.3 Evaluation Method ................................ 71
  4.3.1 The Impact of System I/O cache ............... 71
  4.3.2 I/O versus Distance Calculation ............... 73
  4.3.3 Our Evaluation Methods ...................... 74
4.4 Similarity Query Performance Comparison .......... 75
  4.4.1 DTW distance .................................. 75
  4.4.2 Euclidean distance ........................... 80
4.5 Summary ....................................... 83

## 5 Conclusions and Future Directions

5.1 Summary ....................................... 85
5.2 Future Directions ............................... 87
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>An example of the Haar wavelet transformation</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>A SAX lookup table for three symbols</td>
<td>20</td>
</tr>
<tr>
<td>3.1</td>
<td>Different PAA points</td>
<td>43</td>
</tr>
<tr>
<td>4.1</td>
<td>The number of the time series in each group categorized by the optimal dimensionalities measured by the Euclidean distance.</td>
<td>68</td>
</tr>
<tr>
<td>4.2</td>
<td>The number of the time series in each group categorized by the optimal dimensionalities measured by the DTW distance.</td>
<td>69</td>
</tr>
<tr>
<td>4.3</td>
<td>MR-index configurations I (DTW)</td>
<td>75</td>
</tr>
<tr>
<td>4.4</td>
<td>MR-index configurations II (ED)</td>
<td>80</td>
</tr>
</tbody>
</table>
# List of Algorithms

1. RangeNaiveScan($Q, \epsilon$) ........................................... 28
2. $k$NNNaiveScan($Q,k$) ........................................... 29
3. RangeLBScan($Q,\epsilon$) ........................................... 30
4. $k$NNLBScan($Q,k$) ........................................... 31
5. R-Insert($E$) ..................................................... 33
6. R-ChooseLeaf($E,N$) ........................................... 33
7. R-AdjustTree($N,NN$) ........................................... 35
8. RangeGEMINI($Q,\epsilon$) ........................................... 38
9. $k$NNGEMINI($Q,\epsilon$) ........................................... 40
10. Insert($S$) ...................................................... 49
11. ChooseLeaf($S,p$) .............................................. 50
12. UpdatePath($S,p$) .............................................. 51
13. AdjustTree($p,splitNode, S$) ................................ 52
14. RangeQuery($Q,\epsilon,N$) ..................................... 54
15. KNN($Q,k$) ..................................................... 55
16. IndexConfig($D,C$) ............................................. 59
17. GetOptimalDimension($m,C$) ................................ 60
18. Filtering($m, C$) ................................................ 60
19. ConstructIndex($C$) ............................................. 61
List of Figures

1.1 An example of time series data. ........................................... 2
1.2 A summary of similarity measures. ................................. 4

2.1 The Haar wavelet transformation. ................................. 14
2.2 (a) An example of the PAA representation. The time series is reduced to length 5. In order to visually compare the reduced form with the original series, each point in the reduced form is duplicated according to the number of points within each segment. (b) The lower bounding distance between two PAA representations $\tilde{S}_1$ and $\tilde{S}_2$ is the square root of the sum of the product of squared length of the gray lines with the number of points in each segment. ........ 16
2.3 (a) An example of the APCA representation. The time series is reduced to length 5. (b) The lower bounding distance between a time series $S_1$ and an APCA representation $\tilde{S}_2$ is the square root of the sum of the product of the squared length of the gray lines with the length of the segment they falls in. ......................... 17
2.4 An example of the SAX representation. The original time series $S$ is reduced to dimensionality 5 and is further SAX represented as $cbabb$. ................................................................. 19
2.5 A comparison between the Euclidean distance and the DTW distance. Note that the misalignment in time between two sequences can be handled in DTW, since a point in one sequence is allowed to map any point in the other. ................................. 21
2.6 An example of the Sakoe-Chiba band and the Itakura parallelogram. The possible warping paths are restricted to the grey area. ........ 22
2.7 Overlap arrangement in Yi’s lower bounding technique for DTW 23
2.8 Enclose arrangement in Yi’s lower bounding technique for DTW 23
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.9</td>
<td>Disjoint arrangement in Yi’s lower bounding technique for DTW</td>
<td>23</td>
</tr>
<tr>
<td>2.10</td>
<td>A visualization of Kim’s lower bounding technique. $D_{first}$ ($D_{last}$) is the difference between the first (last) two points of $S_1$ and $S_2$. $D_{max}$ ($D_{min}$) is the difference between the two greatest (smallest) of $S_1$ and $S_2$</td>
<td>24</td>
</tr>
<tr>
<td>2.11</td>
<td>The envelope of the original time series in Keogh’s lower bounding technique for DTW. $U$ is the upper bound and $L$ is the lower bound.</td>
<td>25</td>
</tr>
<tr>
<td>2.12</td>
<td>The lower bound distance in Keogh’s lower bounding technique for DTW. The area marked with grey lines contribute to the lower bound distance</td>
<td>26</td>
</tr>
<tr>
<td>3.1</td>
<td>A fraction of an MR-index to show how MBRs at each level are constructed. Note that fan-out information is not shown here.</td>
<td>44</td>
</tr>
<tr>
<td>3.2</td>
<td>A sample MR-index after initialization.</td>
<td>46</td>
</tr>
<tr>
<td>3.3</td>
<td>A sample MR-index after two insertions.</td>
<td>47</td>
</tr>
<tr>
<td>3.4</td>
<td>A sample MR-index after a third insertion and the leaf splits.</td>
<td>47</td>
</tr>
<tr>
<td>3.5</td>
<td>As sample MR-index after thousands of hundreds insertions.</td>
<td>48</td>
</tr>
<tr>
<td>4.1</td>
<td>Examples of synthetic time series with 8 segments.</td>
<td>67</td>
</tr>
<tr>
<td>4.2</td>
<td>Examples of synthetic time series with 16 segments.</td>
<td>67</td>
</tr>
<tr>
<td>4.3</td>
<td>Optimal dimensionalities for $D_{libpa1}$ calculations in the wafer dataset.</td>
<td>70</td>
</tr>
<tr>
<td>4.4</td>
<td>Optimal dimensionalities for $D_{libpa2}$ calculations in the wafer dataset.</td>
<td>70</td>
</tr>
<tr>
<td>4.5</td>
<td>Number of I/O accesses for 50 2NN consecutive queries in $R^{16}$-tree index.</td>
<td>72</td>
</tr>
<tr>
<td>4.6</td>
<td>Real time elapsed for 50 2NN consecutive queries in $R^{16}$-tree index.</td>
<td>73</td>
</tr>
<tr>
<td>4.7</td>
<td>Comparison between I/O cost and DTW cost.</td>
<td>74</td>
</tr>
<tr>
<td>4.8</td>
<td>$IO_{convert}$ for 50 2NN consecutive queries in $R^{16}$-tree index.</td>
<td>75</td>
</tr>
<tr>
<td>4.9</td>
<td>$IO_{convert}$ comparison using different DTW warping window size on indexes with different leaf dimensionalities.</td>
<td>76</td>
</tr>
<tr>
<td>4.10</td>
<td>$IO_{index}$ comparison using different DTW warping window sizes on indexes with the same leaf dimensionality.</td>
<td>77</td>
</tr>
<tr>
<td>4.11</td>
<td>[EW] $IO_{convert}$ comparison using different K for indexes with different leaf dimensionalities measured by the DTW distance.</td>
<td>78</td>
</tr>
<tr>
<td>4.12</td>
<td>[EW] $IO_{index}$ comparison using different K for indexes with the same leaf dimensionality measured by the DTW distance.</td>
<td>78</td>
</tr>
</tbody>
</table>
4.13 [EF] $IO_{convert}$ comparison using different K for indexes with different leaf dimensionality measured by the DTW distance. . . . . 79
4.14 [EF] $IO_{index}$ comparison using different K for indexes with the same leaf dimensionality measured by the DTW distance. . . . . 79
4.15 [SYN] $IO_{convert}$ comparison using different K for indexes with different leaf dimensionalities measured by the DTW distance I. . . . . 80
4.16 [SYN] $IO_{convert}$ comparison using different K for indexes with different leaf dimensionalities measured by the DTW distance II. . . . . 81
4.17 [EW] $IO_{convert}$ comparison using different K for indexes with different leaf dimensionalities measured by the Euclidean distance. . . . . 82
4.18 [EW] $IO_{index}$ comparison using different K for indexes with the same leaf dimensionality measured by the Euclidean distance. . . . . 82
4.19 [ECBF] $IO_{index}$ comparison for 50 2NN queries for indexes with the same leaf dimensionality measured by the Euclidean distance. . . . . 83
4.20 [SYN] $IO_{convert}$ comparison for 50 2NN queries for indexes with different leaf dimensionalities measured by the Euclidean distance. . . . . 84
Chapter 1

Introduction

Time series data is used in many real world applications and is being generated at an increasing rate. Examples include price fluctuations in the stock market, large volumes of product sales records, audio streams, the electrocardiographs (ECGs) for a patient and trajectories of moving objects, collected over time. A similarity query can be used to discover all time series in a dataset that are similar to a given reference time series [18]. It is the principal query type for time series data, due to its wide applications in time series analysis, prediction and forecasting. Since time series applications are usually data intensive, there has been a growing demand to support fast retrieval of time series data, based on similarity. This has drawn much attention from the database community over the last two decades [1][16][63][9][28][47][33]. Many approaches have then been proposed to accelerate similarity query processing for time series data.

In this chapter, we first take a close look at time series data and its popular applications in Section 1.1. Then, in Section 1.2, we provide an overview of the existing methods, including time series data representation, similarity measures and indexing. We analyze the current challenges for indexing time series data and describe the motivation of our research work in Section 1.3. Finally, we present the algorithmic contribution of our work and outline the organization of this thesis in Section 1.4.

1.1 Time series and their applications

Over the last few decades, there has been a significant emergence of applications that involve time series data [22]. A time series (hereafter, we may refer to it interchangeably as a “data sequence”) is a sequence of values that are captured in
monotonically increasing time slots, describing the tendency of the evolving data points during some time period. The points for time series data can be either one dimensional, two dimensional or higher dimensional. We use the term $k$-dimensional time series to indicate that the data points of a time series have a dimensionality $k$. An example of a one-dimensional univariate time series is shown in Figure 1.1. It has length of 128, and each data point is a real number that within the range $(-300,700)$. Compared to a single or isolated data point, a time series directly describes the evolution of the data points. By analyzing time series using similarity comparison and pattern recognition, we can have a better understanding of an object or phenomena which the time series is modeling. In addition, we can potentially forecast the future series, based on the current and past ones [57]. Some typical time series applications are as follows:

- **Financial Data**
  In the stock market, the opening and closing prices of a stock over the last month or year are studied by stock brokers. A stock time series provides information that might help them make purchasing and selling decisions [36]. In another scenario, managers of retail shops may wish to trace the monthly sales records for their products. By analyzing these sales series, they may be able to better control their warehouse storage level and manage their purchasing [15].

- **Sensor Data**
  Many sensors can measure physical quantities and record continuous signals over time. These signals are then read by other instruments and can be treated as a time series. For example, an object tracking device can record the location information about a moving object. The continuously monitored location information forms the trajectory of the object, which is a two-dimensional...
spatial time series [62]. An electrocardiographic device captures and records the electrical activity of the heart over time [13]. The corresponding ECG data is also a one-dimensional time series.

- **Multimedia Data**

A video clip can be viewed as a series of individual images, which is a high-dimensional time series. A so-called “humming-query” system can then allow the user to find a matching song by humming part of the tune [35][65]. Such a tune corresponds to a sequence of (Note, Duration) tuples that can be treated as two dimensional time series data.

Inspired by the boom in time series applications, there has been a dramatic growth of interest in querying and mining such data by the database and data mining community. As a consequence, many approaches have been introduced to 1) represent, 2) measure the similarity between and 3) index time series data. In the remaining part of this thesis, unless otherwise specified, we will assume we are dealing with one-dimensional time series data.

### 1.2 Overview of existing methods

The similarity query is an important query type in time series databases. Give a query, a similarity query returns a set of time series that have a similar behavior to the reference time series. Similarity queries can be found in lots of real world scenarios:

- Discover stocks with similar movements in stock prices to a reference stock.
- Determine the sales opportunity using similarity-based methods [59].
- Find if a tune is similar to one of contained scores in a musical database [41][65].

In this section, first we review the commonly used similarity measures for time series data. Then we give an overview of time series representation and indexing methods.

#### 1.2.1 Similarity measures for time series

Various measures have been introduced to measure the similarity between two time series. Give two times series $S_1$ and $S_2$ and a similarity measure $D$, then the dis-
distance between $S_1$ and $S_2$ that represents their similarity is denoted as $D(S_1, S_2)$. Generally a low distance value indicates a high similarity between two time series, however there are still particular measures, in which a large distance value indicates a high similarity. Based on how the points of a time series can get mapped to the points of another time series, we classify these measures into three categories: One to One, One to Many and One to Many / One to None, as summarized in Figure 1.2. The One to One mapping requires each point of a time series should be mapped to the point at the exact same position of the other time series. The One to Many mapping allows each point of a time series to be mapped to one or more points of another time series. The One to Many / One to None mapping not only allows a point of a time series to be mapped to multiple points of another time series, but also allows a point of a time series to have no mapping at all.

- **One to One**
  - $L_p$-norms
    - $L_1$-norm (Manhattan Distance)
    - $L_2$-norm (Euclidean Distance)
    - $L_p$-norm

- **One to Many**
  - Dynamic Time Warping (DTW)

- **One to Many / One to None**
  - Edit distance for string based measure
    - Longest Common Subsequence (LCS)
    - Edits Sequence on Real Sequence (EDR)
    - Swale
    - Edit Distance with Real Penalty (ERP)

Figure 1.2: A summary of similarity measures.

- **Euclidean distance**
  The Euclidean distance [16], together with its variants that are based on the common $L_p$-norms [63] is a well known measure. The complexity both in time and space for evaluating these metrics is linear. However, there is a key limitation of the Euclidean distance and its variants. First, they require two data sequences to be of the same length. Second, because only the data points at the exact same positions of two sequences can be matched (One to One matching), the Euclidean distance is very sensitive to misalignments in time [24][12].

- **Dynamic time warping distance**
  The dynamic time warping distance (DTW) [6] was introduced to find the best warping path between two data sequences using dynamic programming.
It has been used in the fields such as image searching [3], handwritten text alignment[34][50] and speech recognition [46]. Compared to the Euclidean distance, DTW is a more robust measure for time series data, due to its capability of handling local time shifting and scaling by a non-linear mapping from one sequence to the other [32]. This means any data point of one sequence can be matched to an arbitrary point in the other sequence (one to many matching). The disadvantage is, given two times series of length $m$ and $n$, the time and space complexity for computing DTW is $O(mn)$. The number of possible warping paths grows rapidly as the length of the time series increases.

- Edit distance for strings

DTW is still relatively sensitive to noise. So, in addition to DTW, the Longest Common Subsequence (LCSS) [58], Edit distance with Real Penalty (ERP) [10] and Edit Distance on Real sequences (EDR) [11] measures were proposed to better handle noise in time series data. These distances are all based on the concept of edit distance for strings [14]. LCSS introduces a threshold $\epsilon$, indicating that two data points from two series are considered to match if they are within a distance of $\epsilon$ and are given a reward of 1. Otherwise, if they exceed the $\epsilon$ threshold, no reward is given. EDR is similar to LCSS, but instead of rewarding, penalties are assigned to the gaps between two matched segments, with regard to the length of the gaps. Using a constant value for computing the cost of a gap in time series, the ERP distance tries to combine the strength of DTW and EDR. Recently a new approach called FTSE was proposed in [42]. Its similarity model includes both arbitrary match rewards and gap penalties.

1.2.2 Representation methods for time series

There are many time series representation methods that have been proposed for the purpose of accelerating similarity query processing. Some of them have the property of allowing lower bounding for specific similarity measures. That is to say, given a similarity measure, for example the Euclidean distance, we can define a distance function that calculates the distance between two reduced-size representations that can be guaranteed to be less than or equal to the Euclidean distance between the raw series. It is the lower bounding property that not only provides an alternative
way to speed up the similarity query based on scan, but also allows these representations to index time series data without bringing in any false dismissals [16] as we will discuss in Chapter 2. Next we mention some of the representations that have the lower bounding property.

Using Euclidean distance as the similarity measure, an early approach to represent and reduce a time series was proposed in [1], in which the Discrete Fourier Transform (DFT) is used to transform a series from the time domain into the frequency domain. The first few DFT coefficients are used to represent the corresponding time series, achieving dimensionality reduction. Another approach that exploited the Haar wavelet transform [19] was proposed in [9]. The first few Haar coefficients are used to represent a time series. Since the wavelet transformation has the multi-resolution property, the more coefficients that are used, the closer the representation is to the raw series. Later, Keogh introduced the Piecewise Aggregate Approximation method (PAA) in [29]. It approximates a time series by a number of equal-sized segments, each of which is represented by its mean value. Compared to the Haar wavelet transform, it does not require the sequence length to be an integral power of two. Afterwards, Keogh improved the approximation quality of PAA with a new approach called Adaptive Piecewise Constant Approximation (APCA) [30]. The significant improvement of APCA is that it allows each segment to be of arbitrary length, so that more segments can be placed into the areas with high activity, while only a few segments are needed for the areas with low activity. Recently, a symbolic representation for time series data called SAX was proposed in [38]. The basic idea is to use the PAA representation of a time series data as an input and then discretize the time series into a set of discrete symbols.

When DTW is used as the similarity measure, the first representation method that allowed lower bounding was proposed by Yi, Jagadish, and Faloutsos in [64]. The idea is to use the maximum and minimum points to represent a time series. Then, Kim in [60], extended the 2-tuple vector to 4-tuple vector, where besides the maximum and minimum points, the first and last points are also involved. Later, Keogh showed that their PAA representation of time series data is also applicable to DTW, by introducing a new lower bounding technique [28].

For the other similarity measures we listed in Section 1.2.1, such as LCSS and EDR, there are no particular representation methods or dimensionality reduction techniques for them. However, their authors did provide alternative ways to speed up query processing with the help of the weak triangle inequality [58] [11] and the
1.2 Overview of existing methods

1.2.3 Indexing time series data

Indexes are a key structure for databases to support fast data retrieval, by reducing disk I/O access. Since a time series can be viewed as a multi-dimensional point, spatial indexing methods such as the R-tree [20] and its variants [5][55][25] have been used to index time series data. However, the performance of most spatial indexing methods deteriorates significantly for high dimensionalities and eventually reduces to linear scan. This phenomena is known as the curse of dimensionality [1]. The representation methods we listed in Section 1.2.2 can potentially lessen the dimensionality curse by reducing a time series to a shorter length which is indexable. The lower bounding functions guarantee that no false dismissal will occur for spatial index queries. An approach that allows the use of any dimensionality reduction method for spatial indexing is the GEneric Multimedia INdexIng method (GEMINI) [16], which is the most popular framework for indexing time series data due to its simple implementation and generality. For example, when Euclidean distance is used as the similarity measure, the indexed points can be the DFT coefficients, the Haar wavelet coefficients, the PAA and the APCA points. In the case of DTW, only Kim’s 4-tuple feature vector and PAA points are available for indexing. Given a particular spatial index, the performance of the query will then only depend on the tightness of the lower bounding function.

Following the GEMINI framework, the authors in [37] proposed a skyline index for time series data. The method replaces the traditional minimum bounding region (MBR) of a spatial index with their proposed skyline bounding region (SBR). They showed that their indexing strategy could help alleviate the curse of dimensionality for time series data. In [38], the authors used the Vector Approximation (VA) file [17] to index a time series dataset instead of spatial indexes, in a context where SAX is applied for time series representation, because the SAX representation can potentially have a high dimensionality, making traditional spatial indexes unsuitable. Recently, a B-tree like structure called the TS-tree was developed to index time series data [2]. Before indexing, all the time series are dimensionality reduced and quantised into symbols. These symbol based time series are then sorted into lexicographic order in a node that makes all the tree nodes overlap-free.
1.3 Motivation

Dimensionality reduction is a necessary process for time series indexing. The query performance depends on how tight the associated lower bounding function is compared to the original distance function. We define the term “tightness” as the ratio of the lower bounding distance over the true distance between two sequences. We observe that even with the same dimensionality reduction and lower bounding methods, the tightness of the lower bounds between different sequences in the same dataset can significantly vary, even under the same dimensionality. We demonstrate this phenomenon with an example below:

Example 1. Give four sequences $s_1$, $s_2$, $s_3$ and $s_4$ in a time series dataset, it is possible that the tightness of the lower bounds between $s_1$ and $s_2$ is 0.9 using the dimensionality 8 which makes it a good dimensionality for lower bounding $s_1$ and $s_2$, since the lower bounding distance is already very close to the original distance. On the other hand, it is also possible that the tightness of the lower bound between $s_3$ and $s_4$ is only 0.1 using the dimensionality 8 and this increases to 0.9 when the dimensionality is 32.

This phenomenon reveals the main drawback of the GEMINI framework, which is that only a single dimensionality can be used to construct a spatial index and the dimensionality that is chosen may not be optimal for all time series that will be indexed. With a large dimensionality, not only will the fan-out of a spatial index shrink and the overlap region increase, but also it may be unnecessary to use that kind of large dimensionality distinguishing a sequence from others at all. With a small dimensionality, the tightness of the lower bounds will be reduced, resulting in more data I/O accesses and true distance calculations in the refinement process.

This drawback of the GEMINI framework has inspired us to directly develop an index structure that can apply multiple dimensionalities to accelerate similarity queries for time series data. In our proposed index structure, the time series data should be able to be reduced to different dimensionalities, such that most dissimilar time series can be distinguished with only few dimensionalities (low resolution) while more dimensionalities (high resolution) can be used for comparing relatively similar time series.
1.4 Contribution and Organization

In the thesis, we propose the Multi-Resolution Index (MR-Index), an index which is suitable for efficient similarity queries on large time series databases. It assigns different dimensionalities to index nodes at different levels. Intuitively, smaller dimensionalities in the upper levels can be used to prune dissimilar sequences, without traversing down the index further. Larger dimensionalities in the lower levels distinguish relatively similar sequences and guarantee that the number of data I/O accesses and true distance calculations are minimized. We also develop a method that automatically computes the optimal dimensionalities from a given time series dataset and assigns them to different index levels when pre-constructing an MR-index. We show that our MR-Index and dimensionality retrieving method can be integrated with any dimensionality reduction and lower bounding techniques. Our contributions include:

- a novel index structure that allows multiple dimensionalities.
- an automatic dimensionality retrieval method that helps construct the MR-Index.
- an extendable framework that allows any dimensionality reduction methods for indexing to be plugged in.

The organization of the rest of thesis is as follows:

- Chapter 2 is a literature review of relevant work.
- Chapter 3 presents our proposed index structure.
- Chapter 4 provides experimental results with analysis and discussions.
- Chapter 5 gives a summary of the thesis and discusses future research directions.

A subject index and list of references is provided at the end of the thesis.
Chapter 2
Preliminaries and Related Work

In this chapter, first we review different similarity measures for time series data, focusing on representation methods and lower bounding techniques. Then we review works that utilize these methods to accelerate similarity queries for time series databases.

2.1 Terminology

Firstly, we give some basic definitions and relevant symbols that will be used in the remaining part of this thesis.

Definition 1. A one-dimensional time series is a sequence of one-dimensional values that are captured in increasing time slots. We use the symbol $S$ to refer to a time series. The symbol $S[i]$ indicates the $i^{th}$ point of the time series $S$. The symbol $|S|$ represents the length of the time series $S$.

Example 2. A time series $S = \{10, 20, 30, 40, 50\}$ is of length 5. And $S[1] = 10$ is the first point of $S$. $|S| = 5$.

Definition 2. A time series database is a set of time series, which we will refer to such a database as $DS$.

Example 3. A time series database $DS = \{S_1, S_2, S_3\}$ has three time series which are $S_1$, $S_2$ and $S_3$ respectively.

Definition 3. A similarity measure is a function that calculates the distance between two time series, which is generally denoted as $D$. We use particular symbols for particular similarity measures such as $ED$ referring to the Euclidean distance and $DTW$ referring to the dynamic time warping distance.
Example 4. \( D(S_1, S_2) = \epsilon \) indicates that the distance between \( S_1 \) and \( S_2 \) is \( \epsilon \).

Definition 4. **Dimensionality reduction** is the process that reduces the length of a time series, which is generally denoted as DR.

Example 5. \( \bar{S} = DR(S, d) \) and \( |\bar{S}| < |S| \), where \( d \) is the dimensionality (or the length) to which \( S \) is reduced to.

Definition 5. A **lower bounding function** is a function that calculates the lower bound distance between two dimensionality reduced time series. The lower bound distance is less than or equal to the original distance between the two time series. We use the symbol \( D_{lb} \) to refer to a lower bounding function.

It is always true that \( D_{lb}(DR(S_1, d), DR(S_2, d)) \leq D(S_1, S_2) \) where \( d \) is an integer that is less than the lengths of \( S_1 \) and \( S_2 \).

2.2 Lower bounding methods for time series

Over the last decade, many similarity measures and representation methods for dimensionality reduction have been introduced for time series data. In this section, we will review the most common similarity measures for time series data. For each measure, we explain its dimensionality reduction method and lower bounding strategies.

2.2.1 Euclidean distance

Euclidean distance is well known for measuring similarity in early studies, due to its simplicity in implementation and efficiency in running time [1][16][9][29][30][8][61][45]. Recently, Keogh has assessed that Euclidean distance is surprising competitive with other more complex approaches, especially for relatively large data sets [38]. Given two time series \( S_1 \) and \( S_2 \) of the same length \( n \), their Euclidean distance \( ED(S_1, S_2) \) is the square root of the sum of squared differences, which is calculated as:

\[
ED(S_1, S_2) = \sqrt{\sum_{i=0}^{n-1} (S_1[i] - S_2[i])^2}
\]  

(2.1)

There have been many methods proposed to represent the time series and lower bound the Euclidean distance.
The Discrete Fourier Transform

The discrete Fourier transform [56] converts a signal from the time domain into the frequency domain. It is a primary method for dimensionality reduction of time series data. Suppose \( S_t \) is a time series in the time domain, then according to the \( n \)-point discrete Fourier transform, the series in the frequency domain \( S_f \) can be calculated by the following function:

\[
S_f = \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} s_t e^{-j \frac{2\pi ft}{n}}, f = 0, 1, ..., n - 1
\]  

(2.2)

where \( f \) indicates the axis in the \( n \)-dimensional frequency space and \( j \) is the imaginary unit \( j = \sqrt{-1} \). The first few DFT coefficients can be used to represent the raw series \( S_t \). We denote the representation as \( S_k = < S_0, S_1, ..., S_{k-1} > \), where \( k \) is less than \( n - 1 \). The energy of a sequence \( S \) is defined as the sum of energies at every point of the sequence:

\[
E(S) = \sum_{t=0}^{n-1} |S_t|^2
\]  

(2.3)

According to the Parseval’s theorem, the energy in the time domain is the same as that in the frequency domain, which is:

\[
\sum_{t=0}^{n-1} |S_t|^2 = \sum_{f=0}^{n-1} |S_f|^2
\]  

(2.4)

It can be further observed that the Euclidean distance between two series in the time domain is the same as that in the frequency domain, which is:

\[
ED(S_{1t}, S_{2t}) = ED(S_{1f}, S_{2f})
\]  

(2.5)

Since the representations \( S_{1k} \) and \( S_{2k} \) have less coefficients than \( S_{1f} \) and \( S_{2f} \), we have

\[
ED(S_{1k}, S_{2k}) \leq ED(S_{1f}, S_{2f}) = ED(S_{1t}, S_{2t})
\]  

(2.6)

So the Euclidean distance for the DFT represented time series can lower bound the Euclidean distance between the two raw time series.
The Haar Wavelet Transform

Another approach that was proposed in [19], used the Haar wavelet for dimensionality reduction of time series data. The Haar wavelet transforms a time series into different levels with different resolutions. Given a time series \( S \) of length \( n \) which must be a power of 2, the Haar wavelet transforms \( S \) into \((\log_2 n + 1)\) levels (see Figure 2.1). At the \( i^{th} \) level, \( 2^i \) is the resolution of that level, indicating the number of averages (or coefficients) that are generated from the original time series \( S \). The average \( S_{i,j} \) and the coefficient \( d_{2^i+j} \) in the \( i^{th} \) level are calculated as:

\[
S_{i,j} = \frac{1}{2}(S_{i+1,2j} + S_{i+1,2j+1}) \quad (2.7)
\]

\[
d_{2^i+j} = \frac{1}{2}(S_{i+1,2j} - S_{i+1,2j+1}) \quad (2.8)
\]

where \( \frac{1}{2} \) is known as the scaling factor. Finally, the time series \( S \) is represented by its mean value \( S_{0,0} \) and all the coefficients in each level from bottom to top, which are \( \{ S_{0,0}, d_1, d_2, ... , d_{n-1} \} \).

We next give an example to illustrate the Haar wavelet transform. Suppose \( S = \{2, 4, 6, 8\} \) is a time series, the wavelet transformation is shown in Table 2.1. We start from a resolution of 4 that is the same as the length of \( S \). In the first row, the

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Averages</th>
<th>Coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>{2, 4, 6, 8}</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>{3, 7}</td>
<td>{-1, -1}</td>
</tr>
<tr>
<td>1</td>
<td>{5}</td>
<td>{-2}</td>
</tr>
</tbody>
</table>
resolution is 4 and the averages are 2, 4, 6 and 8 which can be view as the original time series $S$. Then, in the second row, we get the resolution 2 by dividing the resolution 4 of the first row by 2. The averages 3 and 7 are obtained by calculating the average of $\{2, 4\}$ and $\{6, 8\}$ in the first row respectively. The coefficients -1 and -1 are obtained by taking the difference of $\{2, 4\}$ and $\{6, 8\}$ in the first row, further multiplied by a scaling factor of $\frac{1}{2}$. Then, in the third row, we get the resolution 1 by further dividing the resolution 2 of the second row. Correspondingly, the average 5 is obtained by calculating the average of $\{3, 7\}$ in the second row and the coefficient -2 is obtained by taking the difference of $\{3, 7\}$ in the second row further multiplied by $\frac{1}{2}$. Finally, the average value of the original time series $S$ which is 5, together with the coefficients from bottom to top $\{-2, -1, -1\}$ constructs the Haar coefficients for the time series data, which are $\{5, -2, -1, -1\}$.

It has been proved [19] that if a scaling factor $\frac{1}{\sqrt{2}}$ instead of $\frac{1}{2}$ is applied during the transformation in Equation 2.7 and Equation 2.8, the Euclidean distance can be preserved in the Haar domain as well, which is $ED(S_1, S_2) = ED(\bar{S}_1, \bar{S}_2)$ where $\bar{S}_1$ and $\bar{S}_2$ are the wavelet representations of $S_1$ and $S_2$ using the scaling factor $\frac{1}{\sqrt{2}}$. Then, just like the DFT transform, the first few Haar coefficients can be used to represent a time series, and the Euclidean distance for the reduced representations can lower bound the Euclidean distance for the raw data, according to $ED(First(\bar{S}_1), First(\bar{S}_2)) \leq ED(\bar{S}_1, \bar{S}_2) = ED(S_1, S_2)$.

**Piecewise Aggregate Approximation**

The piecewise aggregate approximation method (PAA) [29] approximates a time series by a number of equal-sized segments, each of which is represented by its mean value. Suppose $S = \{S_1, ..., S_n\}$ is the original time series, and $\bar{S} = \{\bar{S}_1, ..., \bar{S}_N\}$ is the reduced representation, then the $i^{th}$ element of $\bar{S}$ represents the average of a segment as calculated by the following equation:

$$\bar{S}_i = \frac{N}{n} \sum_{j=\frac{iN}{n}(i-1)+1}^{\frac{iN}{n}i} S_j$$ (2.9)

An example is given in Figure 2.2(a), in which a time series is reduced to 5 dimensions. Compared to the Haar wavelet transform, it does not have the restriction that the sequence length of the time series should be an integral power of two.

Given two sequences $S_1$ and $S_2$, with their PAA representations $\bar{S}_1$ and $\bar{S}_2$, re-
2.2 Lower bounding methods for time series  

Preliminaries and Related Work

Figure 2.2: (a) An example of the PAA representation. The time series is reduced to length 5. In order to visually compare the reduced form with the original series, each point in the reduced form is duplicated according to the number of points within each segment. (b) The lower bounding distance between two PAA representations \( \bar{S}_1 \) and \( \bar{S}_2 \) is the square root of the sum of the product of squared length of the gray lines with the number of points in each segment.

spectively, the lower bounding function \( D_{lbpa1} \) is defined as follows:

\[
D_{lbpa1}(\bar{S}_1, \bar{S}_2) = \sqrt{\frac{n}{N} \sum_{i=1}^{N} (\bar{S}_1[i] - \bar{S}_2[i])^2},
\]

which in Figure 2.2(b) corresponds to the square root of the sum of the product of squared length of the gray lines with the number of points in each segment. The lower bounding property can be stated as \( D_{lbpa1}(\bar{S}_1, \bar{S}_2) \leq ED(S_1, S_2) \). A detailed proof can be found in Appendix A of [29].

Adaptive Piecewise Constant Approximation

The approximation of time series data was further improved in [30], in which a new representation method called Adaptive Piecewise Constant Approximation (APCA) was proposed. It again divides the whole series into several segments, and uses the mean value to represent each segment. However, compared to PAA, its advantage is that it allows each segment to be of arbitrary length, so that more segments can be placed into areas with high activity, while only a few segments are needed for the areas with lower activity.

Given a time series \( S = \{S_1, S_2, ..., S_n\} \), its APCA representation is described as

\[
\bar{S} = \{ \langle v_1, r_1 \rangle, ..., \langle v_M, r_M \rangle \}, r_0 = 0,
\]

where \( M \) is the reduced dimensionality, which represents the number of variable-
length segments, \( v_i \) is the mean value of the data points that fall within the \( i^{th} \) segment, and \( r_i \) is the rightmost point of the \( i^{th} \) segment. From the representation, we can see that if reducing to the same dimensionality, APCA takes twice as much storage as PAA does, due to the involvement of the additional endpoint position information. An example of the APCA representation is given in Figure 2.3(a), in which the time series is divided into three segments.

![APCA representation](image)

Figure 2.3: (a) An example of the APCA representation. The time series is reduced to length 5. (b) The lower bounding distance between a time series \( S_1 \) and an APCA representation \( \tilde{S}_2 \) is the square root of the sum of the product of the squared length of the gray lines with the length of the segment they falls in.

In order to make the APCA representation indexable using spatial indexes, two lower bounding functions were introduced to calculate the lower bounding distance between a query sequence and a data sequence and the distance between a query sequence and the Minimum Bounding Region (MBR) of a group of data sequences. Let us denote these as \( D_{lb} \) and \( MINDIST \) respectively.

Given a time series \( S_1 \), its APCA representation \( \tilde{S}_1 = \{ < v_1, r_1 >, ..., < v_M, r_M > \} \) and another series \( S_2 \), \( D_{lb} \) calculates their lower bound distance by first segmenting \( S_2 \) into \( \tilde{S}_2 = \{ < v'_1, r'_2 >, ..., < v'_M, r'_M > \} \) using the same segment boundaries of \( \tilde{S}_1 \) and then aggregating the length differences of the mean values between the segments of the two APCA representations \( \tilde{S}_1 \) and \( \tilde{S}_2 \). The formula is shown in Equation 2.12 and visualized in Figure 2.3(b).

\[
D_{lbapca}(\tilde{S}_1, \tilde{S}_2) = \sqrt{\sum_{i=1}^{M}(r_i - r_{i-1})(v_i - v'_i)^2}
\]  

(2.12)

A new definition of MBR was introduced so that spatial indexes could be used to do the MINDIST calculation. Each MBR \( R \) in a \( 2M \)-dimensional space represents \( M \) regions, \( R = \{ R_1, R_2, ..., R_M \} \). Each Region \( R_i \) is the minimum bounding region.
that contains the $i^{th}$ segments of all the time series that are indexed below an index node. For an index node $N$, each $R_i \in R$ is defined as:

\[
R_i[1] = \min_{\bar{S}\text{under}N}(\bar{S}.\min_i)
\]
\[
R_i[2] = \min_{\bar{S}\text{under}N}(\bar{S}.r_{i-1} + 1)
\]
\[
R_i[3] = \max_{\bar{S}\text{under}N}(\bar{S}.\max_i)
\]
\[
R_i[4] = \max_{\bar{S}\text{under}N}(\bar{S}.r_{i})
\]

where $R_i[1]$ and $R_i[2]$ define the bottom left corner of $R_i$ and $R_i[3]$ and $R_i[4]$ define the upper right corner of $R_i$. Then the MINDIST is calculated as regarding to the region $R$ which is constructed from a set of two-dimensional rectangles. The detailed function can be found in [30] and it was proved that the MINDIST lower bounds the Euclidean distance between the query sequence and all the sequences indexed below the index node. Suppose $Q$ is a query sequence, and $R$ is the region of an index node as described above, then for any sequence $S$ under $R$ we have $\text{MINDIST}(Q,R) \leq \text{ED}(Q,S)$.

### Symbolic Aggregate approXimation (SAX)

Recently, a symbolic representation for time series data called SAX was proposed in [38]. It takes the PAA representation of a time series as input and discretise it into a set of pre-defined symbols. The discretization process is carried out as follows: First the y-axis, which indicates the domain of each data point is divided into $m$ regions, where $m$ is the cardinality of the symbol set. Then, each region is uniquely assigned a symbol from the set. Finally, any PAA value is assigned the same symbol as the region which it falls in.

An example of the SAX representation is given in Figure 2.4. In the example, the solid line represents the original time series and the dotted line represents its PAA representation, just like in Figure 2.2(a). Then the symbol set $\{a, b, c\}$ divides the y-axis into three regions and each PAA value is mapped into these symbols. Finally the time series is represented as $cbabb$.

The breakpoints on the y-axis can be arbitrary in the SAX representation. However, based on the assumption that normalized time series have a Gaussian distribution, in order to ensure almost equiprobable symbols in a SAX word, the authors used a sorted list of numbers $B = \{\beta_1, ..., \beta_{\alpha-1}\}$ as the breakpoints, such that they
follow the $N(0, 1)$ Gaussian distribution from $\beta_i$ to $\beta_{i+1} = \frac{1}{\alpha}$ ($\beta_0$ and $\beta_\alpha$ are defined as $-\infty$ and $\infty$, respectively).

![Diagram](image)

**Figure 2.4**: An example of the SAX representation. The original time series $S$ is reduced to dimensionality 5 and is further SAX represented as $cbabb$.

Since SAX is a symbolic representation of PAA, its lower bounding function for the Euclidean distance is almost the same as that of PAA, as shown in Equation 2.13. The only difference is that instead of the distance calculation between the actual values in PAA’s lower bound function, the calculation in SAX’s lower bound function can be achieved with a pre-computed look up table, since the distance between symbols can be pre-calculated without knowledge of the actual time series.

$$D_{lbsax}(\hat{S}_1, \hat{S}_2) = \sqrt{n \sum_{i=1}^{N} (\hat{S}_1[i] - \hat{S}_2[i])^2}, \quad (2.13)$$

where $\hat{S}_1$ and $\hat{S}_2$ are the symbolic representations of $S_1$ and $S_2$. For example, given two SAX representations $\hat{S}_1 = \{aabcc\}$ and $\hat{S}_2 = \{bacca\}$ and the lookup table for these three symbols in Table 2.2, also suppose the original length of $\hat{S}_1$ and $\hat{S}_2$ is 40, their lower bound distance is calculated as:

$$D_{lbsax}(\hat{S}_1, \hat{S}_2) = \sqrt{\frac{40}{5} \sum_{i=1}^{5} (\text{dist}(\hat{S}_1[i], \hat{S}_2[i]))^2} \quad (2.14)$$

From the lookup table, we can quickly obtain the following distance value between
2.2 Lower bounding methods for time series  

Preliminaries and Related Work

symbols:

\[
\text{dist}(\hat{S}_1[1], \hat{S}_2[1]) = \text{dist}(a, b) = 0.34 \\
\text{dist}(\hat{S}_1[2], \hat{S}_2[2]) = \text{dist}(a, a) = 0 \\
\text{dist}(\hat{S}_1[3], \hat{S}_2[3]) = \text{dist}(b, c) = 0.34 \\
\text{dist}(\hat{S}_1[4], \hat{S}_2[4]) = \text{dist}(c, c) = 0 \\
\text{dist}(\hat{S}_1[5], \hat{S}_2[5]) = \text{dist}(c, a) = 0.68
\]

(2.15)  
(2.16)  
(2.17)  
(2.18)  
(2.19)

So finally we obtain

\[
D_{lbsax}(\hat{S}_1, \hat{S}_2) = \sqrt{\frac{40}{5} \sqrt{0.34^2 + 0^2 + 0.34^2 + 0^2 + 0.68^2}}
\]

(2.20)

With the help of the pre-calculated symbol distance lookup table, the lower bound calculation for SAX is much faster than PAA lower bounding distance calculation. It can be proved that \(D_{lbsax}\) lower bounds \(D_{lbpaa}\), so that \(D_{lbsax}\) lower bounds the Euclidean distance, since \(D_{lbpaa}\) lower bounds the Euclidean distance already. For example, suppose two time series \(S_1\) and \(S_2\) have the Euclidean distance as \(ED(S_1, S_2)\), and their PAA representations are \(\tilde{S}_1\) and \(\tilde{S}_2\) and SAX representations are \(\hat{S}_1\) and \(\hat{S}_2\) respectively, then we have \(D_{lbsax}(\hat{S}_1, \hat{S}_2) \leq D_{lbpaa}(\tilde{S}_1, \tilde{S}_2) \leq ED(S_1, S_2)\).

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0.43</td>
<td>0.86</td>
</tr>
<tr>
<td>b</td>
<td>0.43</td>
<td>0</td>
<td>0.43</td>
</tr>
<tr>
<td>c</td>
<td>0.86</td>
<td>0.43</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.2: A SAX lookup table for three symbols.

2.2.2 The Dynamic Time Warping (DTW) Distance

The dynamic time warping distance [6] is a more robust similarity measure than the Euclidean distance due to its capability of handling local time shifting and scaling. In DTW, a non-linear mapping from one sequence to another is applied, so that a data point in one sequence can be matched to any other point in the other sequence. A comparison between the Euclidean distance and DTW is shown in Figure 2.5.
Given two sequences $S_1$ and $S_2$, we calculate $\text{DTW}(S_1, S_2)$ using dynamic programming. First we construct a $n \times m$ matrix to align them, where $n$ and $m$ are the lengths of $S_1$ and $S_2$ respectively. The $(i, j)$ element of the matrix represents the distance between the two data points $S_1[i]$ and $S_2[j]$. Then, a warping path $P$, from cell $(1, 1)$ to $(n, m)$, is a set of matrix elements that corresponds to a particular mapping from $S_1$ to $S_2$. Let us denote $P$ as:

$$P = p_1, p_2, ..., p_K, \max(n, m) \leq K \leq n + m - 1$$ (2.21)

The constraints on the warping path $P$ are:

- Monotonicity: Given two consecutive points $p_k = (i, j)$ and $p_{k-1} = (\hat{i}, \hat{j})$ in the path, $i - \hat{i} \geq 0$ and $j - \hat{j} \geq 0$.
- Continuity: Given two consecutive points $p_k = (i, j)$ and $p_{k-1} = (\hat{i}, \hat{j})$ in the path, $i - \hat{i} \leq 1$ and $j - \hat{j} \leq 1$.

The distance of a warping path is the square root of the sum of the distance of matrix elements along the path. After all the possible paths between two sequences are listed and their distances calculated, the DTW distance is the one with the minimum
2.2 Lower bounding methods for time series  Preliminaries and Related Work

distance value.

![Sakoe-Chiba Band and Itakura Parallelogram](image)

Figure 2.6: An example of the Sakoe-Chiba band and the Itakura parallelogram. The possible warping paths are restricted to the grey area.

From the above algorithm for computing the DTW distance, we can see that its time and space complexity is $O(mn)$. The number of possible warping paths grows rapidly as the length of the time series increases. In order to speed up the process of computing DTW, global constraints which limit the scope of the warping path (the warping window) can be introduced to eliminate pathological warping paths, so that a comparatively small section of one sequence is not allowed to map to a comparatively large section of another sequence. Typically used global constrains are the Sakoe-Chiba band and the Itakura parallelogram respectively [53][46]. The former uses a warping window of fixed length, while the latter accomplishes a similar effect by exploiting a warping window with dynamic size, that is relative to the position in the diagonal of the computing matrix. An example of these two constraints is given in Figure 2.10. Experiments showed that the DTW distance with constraints could still give accurate results in similarity queries and classifications [27][49].

**Two Tuple Vector**

The first lower bounding technique for DTW was proposed in [64], in which a time series $S$ is represented as a two tuple vector $<\text{max, min}>$, where $\text{max}$ ($\text{min}$) is the maximum (minimum) value of $S$. These two values construct the upper bound and lower bound for $S$ respectively. Based on the relative positions of the maximum and minimum values of two sequences, the following three different arrangements can be used to calculate their lower bound distance:

22
• $S_1$ and $S_2$ overlap: Suppose $S_1,\max > S_2,\max$, then all the points in $S_1$ that are larger than $S_2,\max$ and all the points in $S_2$ that are smaller than $S_1,\min$ can contribute to the lower bound distance, as shown in Figure 2.7.

• $S_1$ encloses $S_2$: All the points in $S_1$ that are larger than $S_2,\max$ or smaller than $S_2,\min$ can contribute to the lower bound distance, as shown in Figure 2.8.

• $S_1$ and $S_2$ are disjoint: Either all the points in $S_1$ or all the points in $S_2$ are to contribute to the lower bound distance. As shown in Figure 2.9, if the area marked with vertical lines is larger than that marked with slashes, all the points in $S_1$ are picked, otherwise, all the points in $S_2$ are picked.
Combing the above three arrangements, the lower bound distance $D_{lbly}$ between two time series $S_1$ and $S_2$ is defined as:

$$D_{lbly}(S_1, S_2) = \begin{cases} \sum s_1[i] \geq s_2.max |s_1[i] - s_2.max| + \sum s_2[i] \leq s_1.min |s_2[i] - s_1.min| & \text{overlap} \\ \sum s_1[i] \geq s_2.max |s_1[i] - s_2.max| + \sum s_1[i] \leq s_2.min |s_1[i] - s_2.min| & \text{enclose} \\ \max(\sum_{i=1}^n |s_1[i] - s_2.max|, \sum_{i=1}^m |s_2[i] - s_1.min|) & \text{disjoint} \end{cases}$$

(2.22)

### Four Tuple Vector

Kim in [60] introduced a 4-tuple feature vector $< \text{first}, \text{last}, \text{max}, \text{min} >$ to represent a time series. The symbols $\text{first}$ and $\text{last}$ represent the first and the last data points of the series, and $\text{max}$ and $\text{min}$ represent the greatest and the smallest points of the series respectively. Given two time series $S_1$ and $S_2$, their lower bounding distance $D_{lbkim}$ for the DTW distance is the maximum difference of their corresponding four features as defined in Equation 2.23. A visualization of Kim’s lower bounding technique is shown in Figure 2.2.2.

$$D_{lbkim} = \max \begin{cases} |s_1.first - s_2.first| \\ |s_1.last - s_2.last| \\ |s_1.max - s_2.min| \\ |s_1.min - s_2.min| \end{cases}$$

(2.23)

Figure 2.10: A visualization of Kim’s lower bounding technique. $D_{first}$ ($D_{last}$) is the difference between the first (last) two points of $S_1$ and $S_2$. $D_{max}$ ($D_{min}$) is the difference between the two greatest (smallest) of $S_1$ and $S_2$
Piecwise Adaptive Approximation

Keogh showed that the PAA representation of time series data is also applicable to the DTW distance, by introducing a new lower bounding technique [28]. As we have already introduced the PAA representation in Section 2.2.1, we will directly explain this lower bounding technique for the DTW distance next.

Firstly, given two time series $S_1$ and $S_2$, a bounding envelope $<U, L>$ is constructed for $S_1$, where $U$ and $L$ represent the envelope’s upper bound and lower bound respectively. Suppose $r$ is the warping window size that we discussed above, then $U$ and $L$ are defined as:

$$
U_i = \max (S_1[i - r] : S_1[i + r]) \\
L_i = \min (S_1[i - r] : S_1[i + r])
$$

(2.24)

We visualize the envelope together with the original series in Figure 2.11. Then the lower bound distance between $S_1$ and $S_2$ is the squared sum of the distances from every point of $S_2$ that is not within the bounding envelope of $S_1$, to the nearest orthogonal edge of the bounding envelope, which can be defined as:

$$
D_{lbkesselh}(S_1, S_2) = \sum_{i=1}^{n} \begin{cases} 
(S_2[i] - U_i)^2 & \text{if } S_2[i] > U_i \\
(S_2[i] - L_i)^2 & \text{if } S_2[i] < L_i \\
0 & \text{otherwise}
\end{cases}
$$

(2.25)

We visualize the lower bound distance $D_{lbkesselh}$ between two original time series in Figure 2.12. Then, after $S_1$ and $S_2$ with length $n$ are PAA reduced to $\bar{S}_1$ and $\bar{S}_2$ with length $N$, $S_1$’s bounding envelope is transformed accordingly, with $\hat{U}$ and $\hat{L}$
2.2 Lower bounding methods for time series

Preliminaries and Related Work

Figure 2.12: The lower bound distance in Keogh’s lower bounding technique for DTW. The area marked with grey lines contribute to the lower bound distance.

respectively, which are defined as:

\[ \hat{U}_i = \max(U_{\frac{n}{N}(i-1)+1}, \ldots, U_{\frac{n}{N}(i)}) \]

\[ \hat{L}_i = \min(L_{\frac{n}{N}(i-1)+1}, \ldots, L_{\frac{n}{N}(i)}) \] (2.26)

Once we have the transformed envelope, then the squared sum of the distances from every PAA point of \( \bar{S}_2 \) that falls outside the transformed envelope of \( S_1 \), to the nearest orthogonal edge of the bounding envelope, multiplying the length of each segment is returned as the lower bound distance, which can be seen in Equation 2.27.

\[
D_{lbpaad2}(S_1, \bar{S}_2) = \sqrt{\frac{N}{n} \sum_{i=1}^{n} \left\{ \begin{array}{ll}
(S_2[i] - \hat{U}_i)^2 & \text{if } S_2[i] > \hat{U}_i \\
(S_2[i] - \hat{L}_i)^2 & \text{if } S_2[i] < \hat{L}_i \\
0 & \text{otherwise}
\end{array} \right\}}
\] (2.27)

Recently, the bounding envelope (we will refer to it as Shasha’s envelopes afterwards) for calculating the lower bounding distance between a time series and a PAA point was further improved by a linear transformation in [65]. The definition of the new bounding envelope is shown in Equation 2.29, in which the new upper bound \( \bar{U} \) and lower bound \( \bar{L} \) are themselves PAA transformed from the original envelope \( \hat{U} \) and \( \hat{L} \).

\[
\bar{U}_i = \frac{N}{n} \sum_{j=\frac{n}{N}(i-1)+1}^{n} \hat{U}_j
\]

\[
\bar{L}_i = \frac{N}{n} \sum_{j=\frac{n}{N}(i-1)+1}^{n} \hat{L}_j
\] (2.28)
Compared to the old bounding envelope Keogh proposed, the new one does not even totally enclose the original query sequence, which gives a higher chance that each value of another PAA point is likely to fall outside the envelope in order to contribute to a more tight lower bound. It was proved that with such an envelope, the lower bounding function $D_{lbshasha}$ (Equation 2.29) still lower bounds the DTW distance.

$$D_{lbshasha}(S_1, \bar{S}_2) = \sqrt{\frac{N}{n} \sum_{i=1}^{N} \left\{ \begin{array}{ll} (\bar{S}_2[i] - \bar{U}_i)^2 & \text{if } \bar{S}_2[i] > \bar{U}_i \\ (\bar{S}_2[i] - \bar{L}_i)^2 & \text{if } \bar{S}_2[i] < \bar{L}_i \\ 0 & \text{otherwise} \end{array} \right.} \quad (2.29)$$

### 2.3 Similarity queries

Similarity queries are widely used in the real world applications. Based on how a time series is compared to another, similarity queries are usually classified into two categories. One is whole matching, in which the query and compared data sequences share similar or identical lengths [1][28][30][48]. The other is subsequence matching, where we look for a consecutive subsequence within each data sequence, that best matches the query sequence using a much smaller length [39][40][23][52]. For the thesis, we will only consider whole matching.

Given a time series, there are usually two similarity query types that can be used to find its similar patterns in a database, which are the range query (Definition 6) and the kNN (Definition 7) query, respectively.

**Definition 6. Range query**

*Given a time series dataset $DS$, a query series $Q$, a distance metric $D$, and a threshold $\epsilon$, find all the time series $S \in DS$ such that $D(S, Q) \leq \epsilon$.***

From the definition of range query, we can see that a user defined threshold $\epsilon$ is required for query processing. As the threshold is usually difficult to determine without sufficient domain knowledge, the range query is not used as frequently as the kNN query for the time series databases.

**Definition 7. kNN query**

*Given a time series dataset $DS$, a query series $Q$, a distance metric $D$, and a number $k$, find a subset $R$ of $DS$ that consists of $k$ time series such that for any time series $S_R \in R$ and $S \in \{DS - R\}$, $D(Q, S_R) \leq D(Q, S)$.***
2.3 Similarity queries

kNN queries can be processed without domain knowledge and are widely used for the classification of time series data. For example, the label of a time series can be predicted by the label of its nearest neighbor.

2.3.1 Linear Scan

The most straightforward way to do a similarity query is by using linear scan.

Naive Linear Scan

For a range query (see Algorithm 1), each sequence $S$ is retrieved from the dataset $DS$ and its distance to the query sequence $Q$ is calculated. If the distance value is less than or equal to the threshold $\epsilon$, the sequence will be moved to the result set, otherwise the scan continues with the next candidate sequence.

Algorithm 1: RangeNaiveScan($Q, \epsilon$)

```plaintext
/* This function returns a set of time series, whose distance to $Q$ is equal to or less
than $\epsilon$ */
/* $Q$ is the query time series. */
/* $\epsilon$ is the query range. */
/* $R$ is the result set. */

begin
  for each $S \in DS$ do
      if $D(Q,S) \leq \epsilon$ then
          add $S$ to $R$;
    end
  end
return $R$;
end
```

For the $kNN$ query (see in Algorithm 2), a priority queue that has a maximum capacity of $k$ sequences is needed to maintain the result set. The sequences in the queue are sorted by their distances to the query sequence. The sequence at the top of the queue has the largest distance value. As the scan proceeds, sequences from the dataset are continuously inserted into the queue if their distance is less than the largest distance in the queue. In the meantime, the sequence at the top of the queue is moved out when there is a new one coming in. After the scan is finished, the sequences in the priority queue are the $k$ nearest neighbors of the query sequence.
Algorithm 2: kNNAiveScan(Q,k)

/* This function returns the k nearest neighbors of the query sequence Q. */
/* Q is the query sequence. */
/* k is an integer. */
/* R is the result set. */

begin

Let R be a priority queue; // maintain the top k sequences

for each S \in DS do

if R is not full then

insert S into R ; // insert the first k sequences

else

\[ d = D(Q,S) ; \]

if \[ d < R.largestDist \] // find a candidate

R.removeTop();

insert S into R;

end

end

return R;

end

For the two naive scan algorithms, we need to read every sequence from the data file and calculate its distance to the query sequence, which yields time complexity of O(n).

Linear Scan With Lower Bounding

In Algorithm 1 and Algorithm 2, the distance can be any similarity measure, as we discussed in Section 1.2.1. For similarity measures that have a lower bounding distance function, it is in fact not necessary to calculate the actual distance between all the candidate sequences and the query sequence.

In a range query, only candidate sequence whose lower bounding distance to the query sequence is larger than \( \epsilon \), can be pruned without calculating the actual distance because:

\[ D(Q,S) \geq D_{lb}(Q,S) > \epsilon \]  \hspace{1cm} (2.30)

An improved range query is shown in Algorithm 3, where Line 3 and Line 4 are the pruning steps.

For a kNN query, a candidate sequence whose lower bounding distance to the
2.3 Similarity queries

Algorithm 3: RangeLBScan(Q, ϵ)

/* This function returns a set of time series, whose distance to Q is equal to or less than ϵ. The lower bounding strategy is applied compared to Algorithm 1. */
/* Q is the query time series. */
/* ϵ is the query range. */
/* R is the result set. */
begin
for each $S \in DS$ do
  if $D_{lb}(Q,S) \geq ϵ$ then
    continue ; // pruned using lower bounding distance
  end
  if $D(Q,S) \leq ϵ$ then
    add $S$ to $R$ ; // find a result
  end
end
return $R$;

query sequence is larger than the largest actual distance in the queue can be pruned without calculating the actual distance because:

$$D(Q,S) \geq D_{lb}(Q,S) > D(Q,C_k) \geq D(Q,C_{k-1}) \geq ... \geq D(Q,C_1) \quad (2.31)$$

An improved $k$NN query is shown in Algorithm 4, where Line 7 and Line 8 are the pruning steps.

For Algorithm 3 and Algorithm 4, suppose $m$ sequences can be pruned, then we need $n$ time to read the sequences and $(n - m)$ time to calculate the original distance, so the total time is $(2n - m)$. Therefore the time complexity is still $O(n)$.

The advantages can be summarized as: Firstly, I/O access is sequential, and continuous sequential I/O accesses is much faster than random I/O on the index [21][51][54]; Second, the number of I/O accesses and actual distance calculation is fixed, consequently the query time can be estimated accurately; Third, there are no additional storage requirements, needed for the index file.

2.3.2 The GEMINI framework for time series data

Despite the above advantages, scan is not suitable for extremely large time series data sets, in which hundreds of thousands pages need to be read from the disk. In-
Algorithm 4: kNNLBScan(Q,k)

/* This functions returns the k nearest neighbors of the query sequence Q. The lower bounding strategy is applied, in contrast to Algorithm 2. */
/* Q is the query sequence. */
/* k is an integer. */
/* R is the result set. */

begin

// maintain the first k sequences
Let R be a priority queue;

for each S ∈ DS do

if R is not full then

insert S into R ; // insert the first k sequences

else

if D_{lb}(Q,S) > R.largestDist then

continue ; // prune using lower bound distance

end

if D(Q,S) < R.largestDist then // find a candidate

R.removeTop();

insert S into R;

end

end

end
stead, indexes can be used to speed up similarity query processing by reducing the number of I/O accesses. In [16] the authors proposed the GEneric Multimedia INDEXing method (GEMINI) which is a framework that can utilize any dimensionality reduction method for indexing. In this framework, all time series are first reduced to a shorter length using some dimensionality reduction method, and then these reduced time series are inserted into a spatial index. The associated lower bounding function that calculates the distance in the index space guarantees that there will be no false dismissals when queries are evaluated using the index. Next we will review the R-tree index, since it is the most commonly used spatial index in the GEMINI framework. Then, we will explain how lower bounding is applied. Finally we review algorithms for range and kNN queries under the GEMINI framework.

**R-tree index**

An R-tree [20] is a height-balanced tree data structure that is similar to B-tree [4], which is used to index multi-dimensional points. A non-leaf node contains a list of entries of the form $< p, MBR >$, where $p$ is the address of its child node and $MBR$ is the minimum bounding rectangle that includes all the $MBRs$ in its child node. A leaf node contains a list of entries of the form $< \hat{p}, R >$, where $\hat{p}$ refers to a record in the database and $R$ is the smallest rectangle that spatially contains the object being indexed.

Let $M$ be the maximum number of entries will fit in one node and let $m \leq \frac{M}{2}$ be a parameter specifying the minimum number of entries in a node, An R-tree satisfies the following properties[20]:

- The root has at least two children unless it is a leaf.
- Every leaf node contains between $m$ and $M$ entries, unless it’s a root.
- Every non-leaf node contains between $m$ and $M$ entries, unless it’s a root.
- All the leaf nodes appear on the same level.
- For each entry in a non-leaf node, its $MBR$ is the smallest rectangle that spatially contains the $MBRs$ in its child node.
- For each entry in a leaf node, its $R$ is the smallest rectangle that spatially contains the object being indexed.

Next we briefly review the insertion and search in R-tree.

The **R-Insert** algorithm (Algorithm 5) in an R-tree uses the $MBRs$ in each non-leaf node to ensure that spatial objects that are close to each other are placed in
Preliminaries and Related Work

2.3 Similarity queries

the same leaf node. Firstly, it invokes the R-ChooseLeaf function to allocate the
leaf node \( l \) for the new entry \( E \). If \( l \) still has room, \( E \) will be inserted into its next
available slot. Otherwise, a split occurs and a new leaf node \( ll \) is generated. Finally
the R-AdjustTree procedure is called to update the tree and propagate the split up
the tree.

Algorithm 5: R-Insert(\( E \))
/* This procedure inserts a new entry into an R-tree. */
/* \( E \) of the form \(< p, R >\) is the new entry to be inserted. */
begin 1
\( l = \text{R-ChooseLeaf}(E,\text{root}); \) // find the leaf node
2 \( ll \leftarrow \text{null}; \) // indicate no split
3 if \( l \) has room then
4 \( l.\text{insert}(E); \)
5 else
6 \( ll \leftarrow \text{R-SplitNode}(l,E); \) // leaf splits
7 end
8 \( \text{R-AdjustTree}(l, ll); \) // update the tree
9 end
10

The R-ChooseLeaf algorithm (Algorithm 6) picks up an appropriate leaf node
for storing the new entry \( E \). During each recursion, if the node \( N \) being visited is
already a leaf node, the algorithm terminates. Otherwise, it looks into its each entry,
and find the entry \( e \) that needs the least enlargement to contain \( E.MBR \). Then the
recursion continues with the child node of \( e \).

Algorithm 6: R-ChooseLeaf(\( E,N \))
/* This function returns the leaf node for the new entry \( E \). */
/* \( E \) is the new entry to be inserted. */
/* \( N \) is the node that is currently visited. */
begin 1
2 if \( N \text{ is a leaf node} \) then
3 \( \text{return } N; \)
4 else
5 Find the entry \( e \) in \( N \), whose MBR needs the least enlargement to
spatially contain \( E.MBR \);
6 \( \text{return } \text{R-ChooseLeaf}(E, e.\text{child}); \)
7 end
8 end
9

The R-AdjustTree algorithm (Algorithm 7) updates the MBRs of the required
2.3 Similarity queries

Preliminaries and Related Work

entries after an insertion and propagates the splits up the tree if necessary. Firstly
if the node \( N \) visited in last recursion is already the root node and the second pa-
rameter \( N \) is not null, which indicates the original root has splitted, a new root will
be created with two entries pointing to the original root and the split node. Conse-
quently the tree will grow taller by one level. Otherwise if the original root doesn’t
split, the algorithm terminates. If the node \( N \) is a non-leaf node, the MBRs of \( E \)
which is the entry in \( N \)’s parent node \( P \) that points to \( N \) is updated to enclose all
the MBRs in \( N \). Then if the second parameter \( NN \) is not null, which indicates that
\( N \) splits in last recursion, a new entry \( EE \) will be created for \( NN \), and its MBR will
be updated accordingly. Next if the parent node \( P \) still has room for \( EE \), \( EE \) will
be inserted into the next available slot of \( P \). Otherwise, another split occurs and
generates a new node \( PP \). Finally R-AdjustTree is called recursively to propagate
the update up the tree.

The \textbf{SplitNode} function is called when a new entry is to be inserted into a node
\( N \) that already contains \( M \) entries. It generates another node \( NN \) and then all
the \((M + 1)\) entries are partitioned into these two nodes \( N \) and \( NN \). In order to
make the search on an R-tree efficient, the total area that \( N \) and \( NN \) covers after
splitting should be minimized. Based on how these entries are partitioned, there are
following three \textbf{SplitNode} algorithms for an R-tree:

- **Exhaustive** It lists all the possible groupings, and choose the best group that
covers the minimum area. For \((M + 1)\) entries, since the number of possible
groupings is \( O(2^M) \), this algorithm is a time consuming process which is
difficult to use in practise.

- **Quadratic** In this algorithm, firstly two entries will be chosen to be the first
elements of the two nodes \( N \) and \( NN \) respectively. These two entries form
the pair that would waste the largest area if they were put into the same node.
Then the \textbf{PickNext} function is called to choose an entry \( E \) from the remaining
entries. Next \( E \) is inserted into the node that needs the least enlargement to
enclose it. For each remaining entry \( \hat{E} \), the \textbf{PickNext} algorithm firstly tries
to insert it into \( N \), and calculate the area increase \( d_1 \) required for covering
\( \hat{E} \). Then it tries to insert it into \( NN \), and again calculate the area increase
\( d_2 \) required for covering \( \hat{E} \). Finally it returns the entry with the maximum
difference between \( d_1 \) and \( d_2 \). The complexity for this algorithm is \( O(M^2) \).

- **Linear** This algorithm picks the first two entries in the following way. Firstly
for each dimension, it discovers the entry whose MBR has the highest low
Algorithm 7: R-AdjustTree(N, NN)

/* This procedure updates the MBRs of the involved entries after an insertion and propagates the slips up the tree. */
/* N is the node being visited in last recursion. */
/* NN is the resulting second node in last recursion if splits occurs, a null value indicates no splitting. */

begin
  if N is the root then
    if NN is not null then // old root splits
      Create a new root with two entries for N and NN;
    end
    return;
  else
    Let P be the parent node of N;
    Let E be the entry for N in P;
    Update E.MBR so that it encloses all the MBRs in N;
    PP ← null; // indicate no split
    if NN is not null then // splits occured in last recursion
      Create a new entry EE for NN; // update EE’s MBR accordingly
      if P has room then
        P.insert(EE);
      else
        PP ← R-SplitNode(P, EE);
      end
    end
    R-AdjustTree(P, PP);
  end
end
2.3 Similarity queries

The Search algorithm in an R-tree is straightforward. Given a query rectangle $Q$, the algorithm starts from the root node, and recursively examines the subtree of an entry $E$ as long as $E$’s MBR intersects $Q$, until the record is found in the leaf node.

When an R-tree is used in the GEMINI framework, the insertion for the reduced time series data is the same as described above. However, the search algorithm needs to be changed, which we will explain in the remaining sections.

**Lower Bounding in the GEMINI framework**

Under the GEMINI framework, all the reduced time series are stored in the leaf nodes of the spatial index, with pointers pointing to the actual time series in the data file. In order to avoid false dismissals during querying, lower bounding functions are given together with the dimensionality reduction methods, to calculate the distance in the index space. For each dimensionality reduction method, there should be two lower bounding functions $D_{lb}$ and $MINDIST$. $D_{lb}$ calculates the distance between the query sequence, either the original one or the reduced one, and the reduced candidate sequence when a leaf node is reached, such that:

$$D_{lb}(Q, \bar{S}) \leq D(Q, S)$$

or

$$D_{lb}(\bar{Q}, \bar{S}) \leq D(Q, S)$$ \hspace{1cm} (2.32)

$MINDIST$ calculates the lower bound distance between the query sequence, either the original one or the reduced one, and the bound $B$ of a group of reduced time series when an index node $N$ is reached, such that:

$$MINDIST(Q, \bar{S}) \leq \min(\{D(Q, S_i) | S_i \in N\})$$

or

$$MINDIST(\bar{Q}, \bar{S}) \leq \min(\{D(Q, S_i) | S_i \in N\})$$ \hspace{1cm} (2.33)
where \( S_i \in N \) represents the sequence \( S_i \) is stored under the index node \( N \). For simplicity, we will assume that the lower bound distance is calculated between the original query sequence and a reduced candidate sequence. By default, the bound \( B \) is the minimum bounding region that spatially contains the reduced time series in the leaf nodes as defined by the spatial indexes. However, particular bounding methods and MINDIST definitions have been introduced to give a tighter lower bounding distance for some representation methods [30][37].

**Querying under the GEMINI framework**

**Range Query** A typical range query algorithm in the GEMINI framework is evaluated recursively (see in Algorithm 8). The input is the query sequence \( Q \), the range \( \epsilon \) and a tree node \( N \). The query starts with the query sequence \( Q \), the range \( \epsilon \) and the root node of the index. If \( N \) is an index node, each entry \( e \) together with its bounding region \( B \) is retrieved from \( N \). We then calculate the distance MINDIST between the query sequence and these bounding regions. If any lower bound distance is greater than the \( \epsilon \), the entire subtree is pruned, otherwise we continue the query at its child node. On the other hand, if \( N \) is a leaf node, all the reduced time series in that node are retrieved and their lower bound distances to the query sequence are calculated using \( D_{LB} \). If any lower bound distance is already larger than \( \epsilon \), that candidate sequence can be pruned without calculating the actual distance. Otherwise, the actual data sequence is retrieved from the data file and the true distance to the query sequence is calculated. If the true distance is within the range of \( \epsilon \), the corresponding candidate sequence is added to the result set.

**kNN Query** A typical \( k \)NN query algorithm in the GEMINI framework is also recursive (see in Algorithm 9). First, we create a priority queue \( pq \) to store the temporary nodes and the candidate sequences. In the queue, all the nodes and the sequences are sorted in a descending order according to their distance values, with the top having the largest value. Then we push the root node using a zero distance into the queue and enter the loop. Each iteration during the loop, we remove the top element from the queue. If the top element is a data sequence, it is added to the result set directly. Otherwise, there are three possibilities. First, if the top element is the offset that points to the actual data sequence, the data sequence will be retrieved from the disk and its actual distance to the query sequence is calculated and pushed into the queue. Second, if the top element is a leaf node, then all the
Algorithm 8: RangeGEMINI(Q, ϵ)

/* This function returns a set of time series, whose distance to Q is equal to or less
     than ϵ */
/* Q is the query time series. */
/* ϵ is the query range. */
/* R is the result set. */
begin
    if N is an index node then
        for each Entry e ∈ N do
            if MINDIST(Q, e.B) > ϵ then
                return ; // prune the subtree
            else
                // continue the recursion with the subtree
                RangeQuery(Q, ϵ, e.Child);
            end
        end
    else
        for each S ∈ N do
            if Dlb(Q, S) > ϵ then
                return ; // prune the sequence S
            end
            Read S from the disk;
            if D(Q, S) < ϵ then
                Add S to R ; // find a result
            end
        end
    end
end
reduced time series are obtained from the node, their lower bound distances are calculated, and these reduced time series are pushed into the queue with their lower bound distances. Finally, if the top element is an index node, all the entries with their bounding regions are retrieved, and their child nodes are pushed into the queue with the lower bound distances calculated between their bounding regions and the query sequence. The algorithm terminates when there are already $k$ candidates in the result set.

For the range and $k$NN queries in the GEMINI framework, the worst case is that all the index nodes and the data sequences need to be read and the distances between each sequence and the query sequence need to be calculated, which takes $(3n-1)$ time to complete. So the time complexity in the worst case is still $O(n)$. In the best case where only one node needs to be read at each index level and a constant number of sequences need to be read from the data file and are involved in distance calculation, the time complexity is $O(\log n)$.

### 2.4 Summary

In this section, we have reviewed some popular similarity measures for time series data. For each similarity measure, we reviewed specific time series dimensionality reduction methods, together with the lower bounding techniques that can help make the reduced time series spatially indexable. Then, we explained both the naive scan methods for doing range query and $k$NN query in a time series database and the improved scan methods with lower bounding. Finally, after describing the R-tree index, we explained how traditional spatial indexes can index the time series data and how the range query and $k$NN query are conducted using these spatial indexes.
Algorithm 9: $k$NNGEMINI($Q,e$)

/* This function returns the $k$ nearest neighbors of the query sequence $Q$. */
/* $Q$ is the query sequence. */
/* $k$ is an integer. */
/* $R$ is the result set. */

begin
1 Create a priority queue $pq$ to keep temporary internal nodes and candidate sequences.;
2 $pq$.push(0,root);
3 while $pq$ is not empty do
4     top = $pq$.pop();
5     if top is a sequence then
6         add top to the result set;
7         if there are $k$ sequences in the result set then
8             return; // find $k$NN already
9         end
10     else
11         if top is a sequence offset then
12             Retrieve the full sequence $S$ from the data file;
13             $pq$.push(D($Q,S$),$S$); // push the data sequence
14         end
15     end
16     if top is a leaf node then
17         for each reduced series $\bar{S} \in$ top do
18             // push the PAA point
19             $pq$.push(Dlb($Q,\bar{S}$),$\bar{S}$.sequenceOffset);  
20         end
21     else
22         for each entry $e \in$ top do
23             // push the MBR of an index entry
24             $pq$.push(MINDIST($Q,e.B$),$e.child$);
25         end
26     end
27 end
Chapter 3
Multi-Resolution Indexing

In this chapter, we explain our multi-resolution index (MR-index) to support fast data retrieval in a time series database. First of all, we explain the index structure and its associated algorithms. Next, we present our configuration method for automatically computing the optimal dimensionalities for pre-constructing an MR-index. To make our ideas clearer, we use the DTW distance as the similarity measure and PAA as the representation method. Then, in the final section, we explain how our index can in fact be applied using any similarity measures and any dimensionality reduction methods.

3.1 MR-index Structure

In this section, we present our multi-resolution index method for time series databases using constraint DTW distance as the similarity measurement and PAA as the dimensionality reduction method.

An MR-index is a height-balanced tree, similar to an R-tree. Each non-leaf node (internal node) of the index consists of a set of non-leaf entries (internal entries), each of which has a minimum bounding region (MBR) and a pointer that stores the address of the child node. Each leaf node of the index is made up of a set of leaf entries, each of which stores a reduced sequence (PAA points) and a offset that indicates the position of the original sequence in the data file. Before the construction of the MR-index, a parameter $d$ is assigned to each level to indicate either the dimensionality of the MBR in an internal node or the length of the PAA point in a leaf node. The assignment process is called MR-index configuration, which we

1A PAA point is the time series after PAA reduction. The dimensionality of a PAA point is the length of the reduced time series.
3.1 MR-index Structure

Multi-Resolution Indexing

will discuss in Section 3.3. Here we overview the concept briefly to show that unlike R-tree and its variants, the dimensionalities of our MR-index at different levels may be different. In order to simplify the description in the remainder, we use the following symbols to represent the MR-index components.

- \( l \). The tree level, leaf nodes are at level 1.
- \( d \). Dimensionality of an MBR or a PAA point.
- \( NI = \{ EI_1, ..., EI_m \}, d, l \). An internal node with \( m \) entries at level \( l \) with the dimensionality \( d \).
- \( EI_i = R^d, p \). The \( i \)th entry of an internal node with \( p \) as the pointer and \( R^d \) as the MBR whose dimensionality is defined by its node.
- \( R^d = \{ H^d, L^d \} \). A MBR with dimensionality \( d \). \( H^d \) is the higher endpoint of the diagonal of the MBR and \( L^d \) is the lower endpoint of diagonal of the MBR.
- \( NL = \{ EL_1, ..., EL_m \}, d, l \). A leaf node with \( m \) entries at level \( l \) with the dimensionality \( d \).
- \( EL_i = \{ \bar{S}^d_i, p \} \). The \( i \)th entry of a leaf node with \( p \) as the point and \( \bar{S}^d_i \) as the PAA point whose dimensionality is defined by its node.

Once the dimensionality of an entry of an internal node is specified, its MBR is constructed using all the data sequences under this entry instead of its children’s MBRs due to the variance in dimensionality. First, all the data sequences are PAA reduced to the length corresponding to the dimensionality of the target MBR. Then the maximum points at each dimension form the higher endpoint of its major diagonal and the minimum ones make the lower endpoint. Suppose \( k \) sequences of length \( n \) are stored under an entry \( EI_i \) of an internal node \( NI_i \) with a dimensionality \( d \), \( \bar{S}^d_i \) refers to each PAA point transformed from the original sequence \( S_i \), then the higher and lower endpoints \( H^d \) and \( L^d \) of \( EI_i R^d \) are defined as:

\[
H = \{ h_1, h_2, ..., h_d \}, L = \{ l_1, l_2, ..., l_d \}
\]

(3.1)

where for \( 1 \leq j \leq d \),

\[
h_j = \max\{ \bar{S}^d_1[j], ..., \bar{S}^d_d[j] \}, l_j = \min\{ \bar{S}^d_1[j], ..., \bar{S}^d_d[j] \}
\]

(3.2)

Example We give the following example to show how each MBR is constructed. Figure 3.1 gives a fraction of an MR-index, in which the leaf level is assigned a dimensionality of 8, and from bottom to top, the other two levels use dimensionalities
Table 3.1: Different PAA points

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\bar{S}_8^1)</td>
<td>{1,3,5,7,9,11,13,15}</td>
</tr>
<tr>
<td>(\bar{S}_8^2)</td>
<td>{2,4,6,8,10,12,14,16}</td>
</tr>
<tr>
<td>(\bar{S}_8^3)</td>
<td>{5,7,9,11,13,15,17,19}</td>
</tr>
<tr>
<td>(\bar{S}_8^4)</td>
<td>{10,12,14,16,18,20,22,24}</td>
</tr>
<tr>
<td>(\bar{S}_4^1)</td>
<td>{2,6,10,14}</td>
</tr>
<tr>
<td>(\bar{S}_4^2)</td>
<td>{3,7,11,15}</td>
</tr>
<tr>
<td>(\bar{S}_4^3)</td>
<td>{6,10,14,18}</td>
</tr>
<tr>
<td>(\bar{S}_4^4)</td>
<td>{11,15,19,23}</td>
</tr>
<tr>
<td>(\bar{S}_2^1)</td>
<td>{4,12}</td>
</tr>
<tr>
<td>(\bar{S}_2^2)</td>
<td>{5,13}</td>
</tr>
<tr>
<td>(\bar{S}_2^3)</td>
<td>{8,16}</td>
</tr>
<tr>
<td>(\bar{S}_2^4)</td>
<td>{13,21}</td>
</tr>
</tbody>
</table>

4 and 2 respectively. \(S_1, S_2, S_3\) and \(S_4\) are four sequences with length 16. Their corresponding PAA points under different dimensionalities can be found in Table 3.1. First, they are PAA reduced to length 8, and stored in the two leaf nodes \(NL_8^1\) and \(NL_8^2\) respectively. \(NI_4^1\) is the parent node of \(NL_8^1\) and \(NL_8^2\) with dimensionality 4. It has two entries \(EI_4^1\) and \(EI_4^2\) pointing to \(NL_8^1\) and \(NL_8^2\) respectively. Then, the MBR \(R_4^1\) of the entry \(EI_4^1\) is constructed by \(\bar{S}_4^1\) and \(\bar{S}_4^2\), which are the PAA points of \(S_1\) and \(S_2\) under dimensionality 4. The maximum (minimum) points in each of the 4 dimensions form \(R_4^1.H\) (\(R_4^1.L\)) (See Equation 3.3).

\[
R_4^1.H = \{\max(2,3), \max(6,7), \max(10,11), \max(14,15)\} = \{3,7,11,15\} \\
R_4^1.L = \{\min(2,3), \min(6,7), \min(10,11), \min(14,15)\} = \{2,6,10,14\} \quad (3.3)
\]

Correspondingly the MBR \(R_4^2\) of the entry \(EI_4^2\) is constructed by \(\bar{S}_4^3\) and \(\bar{S}_4^4\) in the same way. (See in 3.4).

\[
R_4^2.H = \{\max(6,11), \max(10,15), \max(14,19), \max(18,23)\} = \{11,15,19,23\} \\
R_4^2.L = \{\min(6,11), \min(10,15), \min(14,19), \min(18,23)\} = \{6,10,14,18\} \quad (3.4)
\]

Finally, the four sequences are all under the entry \(EI_2^1\) of the node \(NI_2^1\) with the
3.1 MR-index Structure  

Multi-Resolution Indexing

dimensionality 2. Then, the corresponding MBR $R^2_1$ is constructed by $S^2_1, S^2_2, S^2_3$ and $S^2_4$, the four PAA points under dimensionality 2 (See in Equation 3.5).

$$R^2_1.H = \{ \max(4,5,8,13), \max(12,13,16,21) \}$$

$$= \{13,21\}$$

$$R^2_1.L = \{ \min(4,5,8,13), \min(12,13,16,21) \}$$

$$= \{4,12\}$$ (3.5)

$$F_n = \frac{PAGESIZE - a}{2 * d * 8 + b}$$ (3.6)

where the constant $PAGESIZE$ is normally 4096 bytes and the parameters $a$ and $b$ represent the pre-allocated space size for storing the additional information for a node and an entry respectively in the implementation stage. Note that the constant 8 in the denominator indicates that each float point of an MBR is double precision requiring 8 bytes. For the leaf nodes, the maximum fan-out $F_l$ is defined as:

$$F_l = \frac{PAGESIZE - a}{d * 8 + b}$$ (3.7)

The constant 2 in the denominator is removed in Equation 3.7 because PAA points don’t have two boundaries as MBRs.

Figure 3.1: A fraction of an MR-index to show how MBRs at each level are constructed. Note that fan-out information is not shown here.
From Equation 3.6 and Equation 3.7, we can see that $F_l$ is a fixed value, since there is only one leaf level and it will be confirmed after the leaf is assigned a dimensionality in the configuration stage. However $F_n$ is a variable that depends on the dimensionality that is associated with each level: A small dimensionality leads to a large $F_n$ while a large dimensionality produces a small $F_n$. Although in our MR-index, different levels can be assigned either the same or different dimensionalities, we require that the dimensionality at a lower level should be greater than or equal to that at a upper level (Rule 1). Accordingly the leaf node has the largest dimensionality in an MR-index. The incremental dimensionality from root to leaf reveals the main advantage of our MR-index which is nodes at upper levels with large fan-outs can be used to prune most dissimilar sequences with rough MBRs of small dimensionalities while nodes at lower levels with small fan-outs can be used to prune comparatively similar sequences with more precise MBRs of high dimensionalities.

**Rule 1.** (Incremental Dimensionality) Given an MR-index internal node $NI_i$, for any other internal node (or leaf node) $NI_j$ such that $NI_i.l \leq NI_j.l$, it is required that $NI_i.d \geq NI_j.d$ and for any other internal node $NI_j'$ such that $NI_j'.l \leq NI_i.l$, it is required that be $NI_j'.d \geq NI_i.d$.

**Example** To give a clear overview of the MR-index, we use the following example to show the entire construction of an MR-index. We suppose that the following $<$ dimensionality, level $>$ list has been produced in the index configuration stage:

$$config = \{< 3, 4 >, < 2, 8 >, < 1, 16 >\}. \tag{3.8}$$

It means that the root (level 3), its children (level 2) and the leaves (level 1) are assign to the dimensionalities 4,8 and 16 respectively.

- **Initialization** Unlike the R-tree which begins with an empty root node, an MR-index starts by constructing a root node, some internal nodes and a leaf node with one node for each level. Additionally, a link has also been set up so that the leaf can be accessed from the root. In this example, we can see in Figure 3.2 that three nodes $NI^4_1$, $NI^8_1$ and $NL^1_{16}$ are created from top to bottom. The dimensionalities assigned to them correspond to those in $config$. Two links are also created in the two entries $EI^4_1$ and $EI^8_1$ such that $NI^4_1$ is the parent of $NI^8_1$ which is the parent of $NL^1_{16}$ in turn. The MBRs for these two entries are initialized to contain all zero values for each dimension. There
is no entry in the leaf node $NL_1^{16}$, since no sequence has been inserted. We notice that root $NI_4^4$ has the largest fan-out which is 8 while the leaf $NL_1^{16}$ has the smallest one which is 2. For simplicity, we don’t use Equation 3.6 and Equation 3.7 to calculate the actual fan-out here, but just assume that the leaf has a fan-out of 2 and the fan-out is doubled when the dimensionality is halved.

![Diagram](image)

Figure 3.2: A sample MR-index after initialization.

- **After two insertions** Now, we add two sequences $S_1$ and $S_2$ into the index. After the insertions, two entries will be created in the leaf node $NL_1^{16}$ that store the corresponding two PAA points $\bar{S}_1^{16}$ and $\bar{S}_2^{16}$, with pointers to their positions in the data file (See in Figure 3.3). Then, the MBR $R_8^8$ of the entry $EI_8^8$ in the parent node is constructed to be the smallest rectangle that contains $\bar{S}_8^1$ and $\bar{S}_8^2$ as we have already discussed in the previous example. In the same way $\bar{S}_4^1$ and $\bar{S}_4^2$ are used to construct the MBR $R_4^1$ of the entry $EI_4^1$ in the root node.

- **Another insertion and split** Then we add another sequence $S_3$. As there is no room in the original leaf $NL_1^{16}$, it splits and generates a new leaf node $NL_2^{16}$. Suppose $\bar{S}_1^{16}$ and $\bar{S}_3^{16}$ are kept in the old leaf, while $\bar{S}_2^{16}$ is switched into the new leaf, then the MBR $R_1^8$ of the entry $EI_1^8$ is re-constructed with $\bar{S}_1^8$ and $\bar{S}_3^8$, since it on longer needs to cover $\bar{S}_2^8$. A new entry $EI_2^8$ is created in the parent node $NI_1^8$ for the new leaf node. Its MBR $R_2^8$ is directly calculated from $\bar{S}_2^8$. As for the MBR $R_4^1$ in the root node, re-construction isn’t necessary because it still has to contain $\bar{S}_4^1$ and $\bar{S}_4^2$. However it needs an update to contain the new PAA point $\bar{S}_3^4$. The index after inserting $S_3$ is shown in Figure 3.4.

- **Insertion continues** As the insertion continues, the splits of the leaf nodes will cause the internal nodes to split and consequently will finally cause the root node to split. As we can not assign an dimensionality to the potential
Figure 3.3: A sample MR-index after two insertions.

Figure 3.4: A sample MR-index after a third insertion and the leaf splits.
new roots in the configuration stage, we will directly use the dimensionality of the old root instead. Therefore, the original root together with its sibling and the new root are required to use the same dimensionality. After some thousands of insertions, the final MR-index is shown in Figure 3.5. Intuitively, based on the dimensionality we can divide it into two parts. Part I which is due to the splits of the original root can be considered as an R-tree with the dimensionality 4. The dimensionalities of nodes in Part II increase from top to bottom.

Figure 3.5: As sample MR-index after thousands of hundreds insertions.

3.2 Algorithms For Insertion and Search

In this section, we describe the algorithms to construct an MR-index and to do range and $k$NN queries using an MR-index. We still use the DTW as the similarity measure and PAA as the dimensionality reduction method to present our ideas.

3.2.1 Insertion

Insert

The insert algorithm (Algorithm 10) for a new data sequence $S$ in the MR-index is straightforward. First, we create a stack $p$ to store all the selected entries so that we
Algorithm 10: Insert($S$)

```plaintext
/* This procedure inserts a time series into an MR-index. */
/* $S$ is the time series to be inserted. */
begin
create a new stack $p$; // store the visited entries
$NL ← ChooseLeaf(S, root, p);$ // allocate the leaf node
if $NL$ has room for $S$ then
    $S^d ← PAA(S, NL, d);$;
    $NL$.add($S^d$);
    UpdatePath($S, p$); // update the MBRs of the visited entries
else
    $NL2 ← NL$.split($S$);
    AdjustTree($NL2, S, p$); // propagate the split up the tree
end
end
```
can easily update or re-construct the involved MBRs after the insertion by popping up each entry from the stack (Line 2). Then the ChooseLeaf method is called to allocate a leaf node $NL$ for the new sequence $S$ (Line 3). If $NL$ is not full yet, $S$ is PAA reduced to the assigned dimensionality of $NL$ and added to its next available slot. Then all the MBRs of the entries along the path $p$ will be updated to contain the corresponding PAA points from $S$ (from Line 4 to Line 7). Otherwise, splits are involved and propagated up the tree (from Line 9 to Line 10).

ChooseLeaf

The ChooseLeaf algorithm is a recursive method that selects an existing leaf node $NL$ for placing the PAA point $S^d$ generated from the new sequence $S$. The first two parameters $S$ and $N$ represent the new data sequence and the node being visited at current recursion respectively. The third parameter $p$, as described in Algorithm 10, pushes each visited entry of an internal node into it. If $N$ is a leaf node already, the method terminates (from Line 2 to Line 3). Otherwise, $S$ is PAA reduced to $S^d$ with regarding to the dimensionality of $N$ first (Line 5). Then we conduct a search in the node $N$ and find the entry $EI$ with the MBR $R^d$ that needs the least enlargement to contain $S^d$ (Line 6). Next, we push this entry $EI$ into the stack $p$ to indicate that it needs update after insertion (Line 7). Finally, we continue the recursion with the child node of $EI$ (Line 8).
3.2 Algorithms For Insertion and Search  
Multi-Resolution Indexing

Algorithm 11: ChooseLeaf(S,p)

/* This function allocates a leaf node to store S. */
/* S = the time series to be inserted. */
/* p = the stack stores all the visited entries */

begin
if N is leaf then
   return N;  // find the leaf
end

SD ← PAA(S,N,d);  // dimensionality reduction
For all entries in N, find the entry EI with the MBR that needs the least enlargement to hold SD;
p.push(EI);
return ChooseLeaf(S, EI → child, p);  // continue the recursion with the subtree
end

UpdatePath

The UpdatePath algorithm recursively updates the MBR of each entry along the path p to contain different PAA points generated from the same new data sequence S. First, we check whether there is still any entry in the stack p. If the stack is empty, which means all the involved entries have already been processed, the method ends (from Line 2 to Line 3). Otherwise, we pop up the entry EI at the top from the stack, PAA reduce the original sequence S according to the dimensionality of the node in which EI resides, and then update EI’s MBR R^d to satisfy Equation 3.2 (from Line 5 to Line 7). Next, we continue the recursion with the remaining stack p (Line 8).

AdjustTree

The AdjustTree algorithm recursively propagates node splits from a leaf to the root, making further splits and re-constructing MBRs as necessary (Algorithm 13). First we check whether there is still any entry in the stack p. If the stack is empty already, which indicates that the old root is popped in the last recursion, we will create a new root for the index with two entries EI_1 and EI_2 pointing to the old root and its sibling generated in the root split respectively (from Line 2 to Line 5). Then we PAA reduce all the data sequences that can be fetched from EI_1 and construct its MBR with Equation 3.2. We construct the MBR of EI_2 in the same way (Line 6). Otherwise,
we will pop the top entry \( EI \) from the stack \( p \), which points to the child node that just split. Then we need to re-construct its MBR, since entries in its child node are changed due to splitting, which further change the data sequences that are under this entry \( EI \) (from Line 8 to Line 10). Next, a new entry \( EI_3 \) is created with the pointer to the split node \( \text{splitNode} \) and its MBR is constructed from the data sequences that can be fetched from the split node (from Line 11 to Line 13). If the node \( NI \) that contains \( EI \) still has room for the new entry \( EI_3 \), \( EI_3 \) will be added to the next available slot of \( NI \). Then, the UpdatePath method is called to update the MBRs of the remaining entries along the stack \( p \) (from Line 15 to Line 17). Otherwise, \( NI \) will split and this AdjustTree method is called recursively to manage the new split (from Line 19 to Line 20).

### Split

We use the R-tree’s quadratic algorithm [20] to implement a split that occurs in our index structure. However, other split algorithms such as \( R^* \)-tree could also be applied easily.

#### 3.2.2 Search

We implement the most two common query types for similarity search for time series: range query and \( k \)NN query. Since DTW is the similarity measure and
Algorithm 13: AdjustTree\((p,splitNode, S)\)

/* This procedure propagates the split up the tree and make further splits. */
/* p is the stack that stores the visited entries. */
/* splitNode is the newly created node in last recursion. */
/* S is the newly inserted time series. */
begin
  if \(p\) is empty then
    Create a new root with two entries \(E_1\) and \(E_2\). ;
    \(E_1\).child \(<--\) oldRoot ;
    \(E_2\).child \(<--\) splitNode ;
    Construct the MRBs of \(E_1\) and \(E_2\) ;
  else
    \(E\) \(<--\) \(p\).top() ; // \(E\)'s child splits in last recursion
    Re-construct the MBR of \(E\) ;
    \(p\).pop() ;
    Create a new entry \(E_3\) ;
    \(E_3\).child \(<--\) splitNode ; // create a new entry for the \(splitNode\)
    Construct the MBR of \(E\) ;
    \(N_1\) \(<--\) \(E\).getNode() ;
    if \(N_1\) has room for \(E_3\) then
      currentNode.addEntry\((E_3)\) ; // insert the entry in the current node
      UpdatePath\((S,p)\) ; // update the remaining visited entries and terminates
    else
      \(newSplitNode\) \(<--\) \(N_1\).split\((E_3)\) ; // another split
      AdjustTree\((p,newSplitNode,S)\) ; // propagate up the split
    end
  end
end
PAA is chosen for dimensionality reduction, the two lower bounding functions from Equation 2.27 in Section 2.2.2 will be used here.

**Range Query**

Given a query sequence $Q$, the range query finds all the sequences in the database that are similar within a distance $\epsilon$ (Algorithm 14). The algorithm recursively accesses the tree nodes from the root to the leaves. For each node $N$ that is visited, if it is an internal node, the query sequence $Q$ is PAA reduced according to the dimensionality of $N$ (Line 3). Then we evaluate each entry $EI$ in $N$ with the lower bounding lemma (from Line 4 to Line 6). A subtree of an entry in a non-leaf node is eliminated if the MINDIST between the MBR of the entry and the reduced query sequence $\bar{Q}^d$ is greater than $\epsilon$. However, if the MINDIST is still less than or equal to $\epsilon$, the method continues the recursion with the child node of $EI$ (from Line 5 to Line 6). If the current visited node $N$ is a leaf node, we still reduce the query sequence $Q$ first according to the dimensionality of $N$ (Line 10). Then we evaluate each entry $EL$ in the leaf node $N$. If the lower bound distance $LB_{PAA}$ between the PAA point $\bar{S}^N.d$ in the entry $EL$ and the reduce query sequence $\bar{Q}$ is less than or equal to the threshold $\epsilon$, we need to read the actual data $S$ sequence from the data file and calculate the DTW (from Line 11 to Line 13). The sequence $S$ is added to the result if and only if the DTW is still less than or equal to $\epsilon$ (Line 15). For those PAA points whose lower bound distance is already larger than $\epsilon$, the recursion ends.

**kNN Query**

The $kNN$ query algorithm finds the top $k$ sequences in the database that are close to a query sequence $Q$ (Algorithm 15). We begin by creating a priority queue $pq$ and pushing the root node of the index into it (Line 2 and Line 3). All the elements in the priority queue have been assigned a value that is either a lower bound distance or an actual DTW distance. These elements are sorted in the ascending order, the top element of the queue has the smallest distance value. Then we traverse the index by popping an element out of the queue at each step (Line 5). If the popped element is a sequence, we will directly add it to the result set because any sequence that is retrieved later here can not be more similar to the query sequence than the current one (Line 6 and Line 7). If there are already $k$ sequences, the method terminates.
Algorithm 14: RangeQuery(Q, ε, N)

/* This function returns a set of time series, whose distance to Q is equal to or less */
/* Q is the query time series. */
/* ε is the query range. */
/* N is the node that is currently visited. */
/* R is the result set. */

begin
  if N is an internal node then
    \[ \bar{Q}_d \leftarrow \text{PAA}(Q, N.d); \] // dimensionality reduction
    for each Entry EI in N do
      if \[ \text{MINDIST}(\bar{Q}, EI.MBR) \leq \varepsilon \] then
        RangeQuery(Q, ε, N.child, R); // continue the search in the
        sub-tree
      end
    end
  else
    \[ \bar{Q}_d \leftarrow \text{PAA}(Q, N.d); \] // dimensionality reduction
    for each Entry EL in N do
      if \[ D_{lbpa2}(\bar{Q}, EL.S^N.d) \leq \varepsilon \] then
        /* calculate the true distance for those cannot be pruned using the */
        /* lower bounding distance */
        Retrieve the sequence S from the data file with EL.child;
        if \[ DTW(Q, S) \leq \varepsilon \] then
          Add S to R;
        end
      end
    end
  end
end
Algorithm 15: KNN(Q,k)

/* This functions returns the k nearest neighbours of the query sequence Q. */
/* Q is the query sequence. */
/* k is an integer. */
/* R is the result set. */

begin
1 Create a priority queue pq to keep temporary internal nodes and
candidate sequences.;
pq.push(0,root); // start the loop with the root node
2 while pq is not empty do
3 top = pq.pop();
4 if top is a sequence then
5 add top to R;
6 if there are k sequences in the result set then
7 return R;
8 end
9 else
10 if top is a sequence offset then
11 Retrieve the full sequence S from the data file;
pq.push(DTW(Q,S),S); // calculate the true distance
12 end
13 if top is a leaf node then
14 for each entry EL in top do
15 Q ← PAA(Q,EL.d); // dimensionality reduction
16 pq.push(D_{lpaa2}2(Q,EL.PAAPoint),EL.child) ;
17 end
18 else
19 for each entry E1 in top do
20 Q ← PAA(Q,E1.d); // dimensionality reduction
21 pq.push(MINDIST(Q,E1.R),E1.child); 
22 end
23 end
24 end
25 end
26 end
27 end
28 end
29 end
3.3 Index Configuration

Otherwise if the popped item is an offset that points to the data sequence, we will read the sequence from the data file and calculate the DTW distance and push it back into the queue (from Line 12 to Line 14). If it is either an internal node or a leaf node, each entry $E_I$ will be evaluated by calculating the lower bound distance between the reduced query sequence $\bar{Q}$ and the MBR in the internal entry or the PAA point $\bar{S}$ in the leaf entry. For different levels, the lower bound distance calculation is based on the specific dimensionality that associates with the level. Then the child nodes or the sequence offsets together with the lower bound distances are pushed back into the queue respectively.

3.3 Index Configuration

To build up an MR-Index, we need a list of dimensions to be assigned to each index level. In this section, we present the process to obtain the required dimension list. First of all, we give a formal definition of our MR-Index configuration problem:

**Problem Definition 1.** Given a dataset $D$ which contains $m$ sequences $S_1,...,S_m$, find a set $R$ of $<d,l>$ pairs to construct an MR-index.

Here, in each pair, the first integer $d$ represents the dimension that is picked from a dimension candidate set $C$. In the scope of this thesis, the dimension candidate set $C$ is $\{2,4,8,16,32,64\}$. The second integer $l$ that starts from 1 indicates the index level respectively. The leaf level is ranked as 1.

The MR-index configuration process first computes a set of dimensions that are suitable for lower bound distance calculations between sequences in the dataset, then it sorts the dimensions in a descending order and assigns them to the index levels from bottom to top. We begin by introducing two definitions that will be used later.

**Definition 8.** Given two sequence $S_i$, $S_j$ and a dimension $d$, the relative tightness of lower bound distances (defined as $rt$) is the ratio of the lower bound distance after sequences dimensionality reduction over the original lower bound distance, which can be calculated as follows:

$$rt(S_i, S_j, d) = \frac{D_{lbpa2}(S_i, PAA(S_j, d))}{D_{lbkeogh}(S_i, S_j)}$$  \hspace{1cm} (3.9)
rt is in the range [0,1], with the larger the better. We can reduce \( S_j \) to \( d \) while calculating the lower bound distance with \( S_i \), if and only if \( rt(S_i, S_j, d) \) is greater than a threshold \( \alpha \).

**Definition 9.** Given a dataset \( D \) of \( m \) sequences, a sequence \( S \), a dimension candidate set \( C \) and a threshold \( \alpha \), the optimal dimension (defined as \( \text{optimaldim} \)) for \( S \) doing lower bound distance calculations in \( D \) is the smallest dimension in \( C \) that satisfies Equation 3.10

\[
\frac{1}{m} \sum_{i=1}^{m} rt(S, S_i, d) \geq \alpha, S_i \in D \tag{3.10}
\]

Equation 3.10 calculates the average of the relative tightness between \( S \) and the other sequences \( S_i \) using the candidate dimension \( d \), and checks if the average is greater than or equal to the threshold \( \alpha \).

Algorithm 16 shows the MR-Index configuration process. In this algorithm, we use two mappings \( m_1 \) and \( m_2 \) respectively. \( m_1 \) is an integer to integer mapping. We categorize the sequences based on their optimal dimensions and put the sequence number of each category into \( m_1 \). Next \( m_2 \) is an integer to queue mapping, which is used to store the relative tightness of lower bound distances between \( s_i \) and the other sequences under each dimension. The inputs to the algorithm are the dataset \( D \) to be indexed and the candidate dimension set \( C \) from which the final dimensions are picked. The output \( R \) is a set of \(<d, l>\) pairs representing the structure of the final MR-Index. As the configuration process does pre-calculation on the data sequences, we cannot afford to use the whole dataset. So at the first step, we create a sample set \( S \) from the original dataset \( D \) using a sampling method, see Line 2. Then we initialize the first mapping \( m_1 \) (Line 3). For each sequence \( s_i \) in the sample set \( S \), we create a second mapping \( m_2 \) for it (Line 5 to Line 9) and calculate the original lower bound distance \( olb \) between \( s_i \) and the other sequences \( s_j \) in \( S \) (Line 11). Next we transform \( s_j \) into each dimension \( d \) in the candidate set \( C \) with PAA (Line 13). For each reduced sequence \( \bar{s}_j \), its lower bound distance \( rlb \) with \( s_i \) is calculated and the relative tightness of the lower bound distance \( rt \) under this dimension can be obtained by dividing \( rlb \) by the original lower bound distance \( olb \) (Line 14 to Line 15). Then we allocate the queue in \( m_2 \) with the dimension being used and insert \( rt \) into it (Line 16 to Line 17). After we complete the insertion of \( m_2 \) with every \(<s_i, s_j>\) pair in every dimension \( d \), the GetOptimalDimension function is called which gives the optimal dimension \( \text{optimaldim} \) to \( s_i \) for lower
3.3 Index Configuration

bound distance calculation in $S$ (Line 20). We increase the corresponding count in $m_1$ by 1 (Line 21). Next a subset $\hat{C}$ is generated from the dimension candidate set $C$ with a filtering algorithm to indicate the total optimal dimensions occurred in the sample dataset $S$ (Line 23). We call the $ConstructIndex$ function with $\hat{C}$ to build up an initial MR-Index $R$ first. If $R$ is able to index the whole dataset $D$, the algorithm terminates. Otherwise, we need to insert tree levels to increase the maximum number of sequences that can be indexed. During each insertion, we will pick up the dimension that has a largest count in $m_1$, and minus it by the smallest count (Line 27 to Line 28). In this way, we can ensure the number of every dimension in the final MR-Index is approximately proportional to how many sequences can use that dimension for lower bound distance calculation. Then we continuously update $\hat{C}$ and re-construct $R$ until it is suitable (Line 29 to Line 30).

Algorithm 17 finds a sequence’s optimal dimension for lower bound distance calculation in a dataset according to Definition 9. The inputs tp the algorithm are the integer to queue mapping $m$ as described in Algorithm 16 and the dimension candidate set $C$ respectively. First, the biggest dimension in the candidate set is picked (Line 2). Then we look into the mapping $m$ and compute the mean value of the relative tightness $rt$ in each dimension. If any dimension $d$ is less than the current optimal dimension $optimaldim$ and its $\bar{rt}$ is greater than or equal to the threshold $\alpha$ defined in Definition 9, we change the optimal dimension to $d$.

The $Filtering$ function is straightforward. Sequences are categorized according to their optimal dimensions and the size of each category is inserted into $m_1$ in Algorithm 16. When Line 23 is reached, $m_1$ holds a sequence number for each candidate dimension. Then in $Filtering$ function, only the dimensions with a number that is greater than or equal to a threshold are chosen for constructing the MR-Index.

The $ConstructIndex$ algorithm returns a set of $<d,l>$ pairs representing the structure of an MR-Index according to the input dimension set $C$. It begins by sorting the dimension set in a descending order. Then it builds up the index from bottom to top, assigning larger dimensions to the lower levels and smaller dimensions to the upper levels.

It is worth mentioning that it’s possible that all the sequences in a dataset share the same optimal dimension. In that case, our MR-Index will be exactly the same as the R-tree index.
Algorithm 16: IndexConfig(D,C)
/* This functions returns a set of <d,l> pairs representing the dimensioanity for each index level. */
/* D is the original time series dataset. */
/* C is the dimensionality candidate set. */
/* R is the result set. */
begin
  S ← Sampling(D); // obtain a sample set
  let m1 be a mapping from an integer to an integer;
  for si ∈ S do
    let m2 be a mapping from an integer to a queue ;
    for d ∈ C do
      create a queue q_d for the candidate dimension d;
      m2.insert(d,q_d);
    end
    for sj ∈ S and sj ≠ si do
      olb ← D_bkσgh(si,sj); // lower bound distance without reduction
      /* get the relative tightness value with the other sequences */
      for d ∈ C do
        sj ← PAA(sj,d);
        rlb ← D_lbpaa(si,sj); // lower bound distance with reduction
        rt ← rlb ÷ olb;
        q_d ← m2[d];
        q_d.insert(rt);
      end
    end
    optimaldim = GetOptimalDimension(m2, C);
    m1[optimaldim]++;
  end
  Ĉ ← Filtering(m1,C);
  R ← ConstructIndex(Ĉ);
  mincount ← min({m1[d]|d ∈ C}); /* continue contructing the index until it can index all the sequences in the original dataset */
  while GetMaxIndexableSeqNum(R) ≤ |D| do
    Find the dimension d ∈ Ĉ with the maximum m1[d];
    m1[d]←mincount;
    Ĉ.insert(d);
    R ← ConstructIndex(Ĉ);
  end
  return R
end
3.3 Index Configuration

Multi-Resolution Indexing

**Algorithm 17: GetOptimalDimension(m, C)**

/* This function returns the optimal dimensionality for a sequence. */
/* m is a <int,queue> mapping. */
/* C is the dimensionality candidate set. */
/* optimaldim is the optimal dimension for a sequence */

1 begin
2   /* get the maximum dimensionality at first */
3   optimaldim ← max(C);
4   for d ∈ C do
5     queue ← m[d];
6     rt ← mean(queue);
7     /* find the smallest dimensionality that meet the threshold requirement */
8     if rt ≥ α and d ≤ optimaldim then
9       optimaldim ← d;
10    end
11 end
12 return optimaldim;
13

**Algorithm 18: Filtering(m, C)**

/* This function returns a set of qualified dimensionalities for index construction. */
/* m is a <int,int> mapping. */
/* C is the dimensionality candidate set. */
/* Ĉ is the qualified dimensionality set. */

1 begin
2   Ĉ ← { };
3   for d ∈ C do
4     /* only pick the dimensionality whose sequence number is equal to or */
5     /* greater than the threshold */
6     if m[d] ≥ (β * |C|) then
7       Ĉ.insert(d);
8    end
9 end
10 return Ĉ
11
Algorithm 19: ConstructIndex(C)

/* This function assigns the dimensionalities from the input into the index levels. */
/* C is a set of dimensionalities. */
/* R is the result set of \(< d, l >\) pairs. */

begin
  R ←− \{
  sort C in the descending order;
  level ←− 1; // the leaf
  /* dimensionality increases from leaf to root */
  for i = 0; i < |C|; i ++ do
    R.insert(\(< C[i], level ++ >\));
  end
  return R;
end

3.4 Framework Extension

We have demonstrated the general idea of our MR-Index in the previous sections with DTW as the similarity measure and PAA as the dimensionality reduction method. Actually, any metrics and dimensionality reduction techniques can be applied in the MR-index as long as they satisfy the following two conditions.

The first condition is to guarantee a necessary construction property for the MR-Index.

**Theorem 1.** Given a dimensionality reduction function dr, we can build up an MR-index if and only if it is able to reduce the dimensionality of a data sequence into multiple smaller dimensionalities.

**Proof.** From the index structure, we know that the dimensionalities of the MBR at different levels may be different for the purpose of multi-resolution. Each MBR is constructed by the data sequences stored below. Therefore, we can build up an index properly if and only if each data sequence can be reduced into different lengths with regarding to those of the MBRs.

**Example 6.** The Euclidean distance, the DFT, Haar wavelet and PAA transformations can be applied in our MR-index, since we can choose the number of corresponding coefficients to represent the time series data. For the DTW distance, the PAA transformation can still be applied for the same reason. However, the two tuple vector and the four tuple vector representations can not be used here, because only a fixed value, two or four, is allowed in their transformations.
3.4 Framework Extension

Multi-Resolution Indexing

The second condition requires two lower bounding functions to be defined to make sure no false dismissal will occur in any range query or \( k \)NN query.

**Theorem 2.** No false dismissal occurs in range queries if and only if there are two lower bound functions \( \text{lbdist} \) and \( \text{mindist} \) such that

\[
\text{lbdist}(\text{dr}(S_1,d),\text{dr}(S_2,d)) \leq d(S_1,S_2)
\]

and

\[
\text{mindist}(\text{dr}(S_1,d),R) \leq \text{lbdist}(\text{dr}(S_1,d),\text{dr}(S_i,d))
\]

where \( R \) is the smallest rectangle that contains the reduced points \( \text{dr}(S_i,d) \).

**Proof.** Let \( q \) be the query sequence, \( c \) be a data sequence, and \( \epsilon \) be the tolerance. We want to prove that if \( d(q,c) \leq \epsilon \), then it will be retrieved when we conduct a range query with the index. That is, we want to prove that the condition in Line 5 of Algorithm 14 is always true if there is any qualified data sequence stored below \( e \). Notice for each data sequence \( c \) stored below an entry \( EI \), there exists

\[
\text{mindist}(\text{dr}(q,\text{EI}.d),\text{EI}.R) \leq \text{lbdist}(\text{dr}(q,\text{EI}.d),\text{dr}(c,\text{EI}.d))
\]

and

\[
\text{lbdist}(\text{dr}(q,\text{EI}.d),\text{dr}(c,\text{EI}.d)) \leq d(q,c) \leq \epsilon
\]

Then we have

\[
\text{mindist}(\text{dr}(q,\text{EI}.d),\text{EI}.R) \leq \epsilon
\]

As long as the conditions in Line 5 and Line 12 are true, we can traverse down the index until we retrieve the data sequence from the data file and compare it with the query sequence using the true distance function (Line 14 of Algorithm 14). Otherwise, suppose

\[
\text{lbdist}(\text{dr}(q,\text{EI}.d),\text{dr}(c,\text{EI}.d)) > \epsilon \geq d(q,c)
\]

Then we cannot further retrieve the data sequence from the disk and calculate the actual distance even if it is a qualified one, due to the false condition in Line 12. This is the same case if

\[
\text{mindist}(\text{dr}(q,\text{EI}.d),\text{EI}.R) \leq \text{lbdist}(\text{dr}(q,\text{EI}.d),\text{dr}(c,\text{EI}.d))
\]
cannot be guaranteed.

**Theorem 3.** No false dismissal occurs in kNN queries if and only if there are two lower bound functions \( \text{lbdist} \) and \( \text{mindist} \) such that

\[
\text{lbdist}(\text{dr}(c_1,d), \text{dr}(c_2,d)) \leq d(c_1, c_2)
\]

and

\[
\text{mindist}(\text{dr}(c_1,d), R) \leq \text{lbdist}(\text{dr}(c_1,d), \text{dr}(c_i,d))
\]

where \( R \) is the smallest rectangle that contains the reduced points \( \text{dr}(c_i,d) \).

**Proof.** Let \( q \) be the query sequence and \( G \) be the dataset, we want to prove that when we issue a kNN query with the index, we can retrieve a result set \( H \) that contains \( k \) sequences such that

\[
d(q,c) \leq d(q,s) \text{ for all } c \in H, s \in G - H
\]

The above inequity obviously holds if we can prove every time what we add to the result set is the nearest neighbor of the query sequence in the remaining dataset. Suppose a data sequence \( c \) is at the top of the priority queue, due to the ascending order, it is obvious that

\[
d(q,c) \leq d(q,s) \text{ for all } s \text{ that is after } c
\]

Next we need to prove

\[
d(q,c) \leq d(q,s) \text{ for all } s \text{ stored below an entry EI that is after } c
\]

Choose an arbitrary entry \( EI \) that is after \( c \), due to the ascending order, we have

\[
d(q,c) \leq \text{mindist}(\text{dr}(q,EI.d), EI.R)
\]

Because we have

\[
\text{mindist}(\text{dr}(q,EI.d), EI.R) \leq \text{lbdist}(\text{dr}(q,EI.d), dr(s,EI.d))
\]

for all \( s \) stored under \( EI \) If

\[
\text{lbdist}(\text{dr}(q,EI.d), dr(s,EI.d)) \leq d(q,s)
\]
3.5 Summary

Then we reach the goal that
\[ d(q, c) \leq d(q, s) \]
Otherwise, if sometimes
\[ lbdist(dr(q, EI.d), dr(s, EI.d)) > d(q, s) \]
Suppose \( k \) nearest neighbors have already been retrieved, and one of them is denoted as \( a \). It is possible that there is a sequence \( b \) that hasn’t been picked up, but
\[ d(q, a) > d(q, b) \]
because
\[ lbdist(dr(q, EI.d), r(b, EI.d)) > d(q, a) > d(q, b) \]
resulting that in the priority queue, we reach \( a \) before we can reach the entry \( e \) that is above \( b \). Then false dismissal occurs.

3.5 Summary

In this chapter, we first gave an overview of our MR-index including the tree structure and the construction of the entry MBR. Then we presented detailed algorithms for how to construct an MR-index and how to do range and \( kNN \) queries in an MR-index. Next, a method based on computation of statistics is given to show the index configuration process. Finally, we showed that any similarity measure and dimensionality reduction methods can be applied in our index, as long as certain conditions are met. We also proved that no false dismissal can occur for MR-index queries.
Chapter 4
Experimental Evaluation

In this chapter, we evaluate the performance of the MR-index that we described in Chapter 3. We use the PAA representation for time series in our MR-index and implement two common similarity measures, the Euclidean distance and DTW distance. For each similarity measure, we incorporate the corresponding lower bounding strategies with PAA into our MR-index. The R-tree index is chosen as the base line for comparison. All the experiments were conducted on a Windows XP desktop with the 2.66GHz Intel Core 2 Duo CPU and 3.25G memory.

Firstly, we briefly describe the data sets that are used in the experiments. Then we explain about the motivation of the proposed index structure, which is there could be multiple optimal dimensionalities in a time series dataset that could be used for lower bounding distance calculations. Next, we explain the method used to quantize the data results. Finally we compare the quantized range query and kNN query results between MR-index and R-tree index, in different data sets using different similarity measures.

4.1 Datasets

The time series data sets we used in our experiments are from both real world applications and commonly used synthetic data generators. Since we require extremely large data sets to evaluate the performance of the index structure, the original data sets are extended. Given a time series \( S = \{S[1], S[2], ..., S[n]\} \) from a real dataset, we generate another time series \( S' = \{S'[1], S'[2], ..., S'[n]\} \) by adding a small amount of noise to each data point \( S[i] \) of \( S \) (Equation 4.1).

\[
S'[i] = S[i] + S[i] \times p \times q, \quad (4.1)
\]
where \( p \) is a value that follows the standard normal distribution and \( q \) is an adjusting ratio, which we assigned to 0.05. In this way, a real time series dataset can be easily extended to any size without changing its original distribution. The details of each dataset used is given as follows:

- **Wafer.** A collection of in-line process-control measurements recorded from various sensors during the processing of silicon wafers for semiconductor fabrication constitute the wafer database; each dataset in the wafer database contains the measurements recorded by one sensor during the processing of one wafer by one tool [43]. The original wafer dataset contains 1000 overlapping time series of length 153. Then we extended the dataset to contain 500,000 time series. We refer to the original wafer dataset as \( W \) and the extended one as \( EW \).

- **Fortune.** This is a collection of stock prices from Fortune 500 companies. The original dataset contains 500 time series of length 252. Then we extended the dataset to contain 500,000 time series. We refer to the extended dataset as \( EF \).

- **CBF.** This is the popular Cylinder-Bell-Funnel (CBF) dataset of [31]. The original CBF dataset we obtained contains 900 time series of length 128. Then we still extended the dataset to contain 500,000 time series. We refer to the extended dataset as \( ECBF \).

- **Synthetic.** This is the synthetic dataset specialized for the PAA transformation, which also contains 500,000 time series. Each time series has a length of 128. All the time series are categorized into two groups. Each time series in the first group has 8 segments, with 16 data points in each segment. For each segment, we assign a value as the average of the 16 data points. Different average values vary a lot. Then the value of the data points falling within one segment is generated by adding a small amount noise to the segment average. Each time series in the second group has 16 segments, with 8 data points in each segment. The value of each data point is generated in the same way. In this dataset, for a time series with 8 segments, we only need to reduce its dimensionality to 8 or even smaller, since its PAA reduction form with a larger dimensionality does not make any big difference. For example, if we reduce it to dimensionality 16, since there are only 8 segments, there will be almost identical points in the reduced form. Typical examples for the time series from the two groups are shown in Figure 4.1 and Figure 4.2. We refer to this...
4.2 Multiple Optimal Dimensionalities

We begin by showing that different time series in a dataset may have different optimal dimensionalities for doing lower bound distance calculation in query processing. This is a key motivation behind our proposal for the MR-index, since an R-tree index with a single dimensionality cannot accommodate such a requirement. The
first experiment involves the original wafer dataset $W$ and evaluates the two lower bounding functions $D_{lbpa1}$ and $D_{lbpa2}$ that we discussed in Chapter 2. $D_{lbpa1}$ is used to lower bound the Euclidean distance while $D_{lbpa2}$ is to lower bound the dynamic time warping distance. In this experiment, for each time series in the dataset, we calculate its relative tightness of the lower bounding distance under each candidate dimensionality using the other time series in the dataset. According to Definition 9, we use $\alpha = 0.6$ to determine their optimal dimensionalities. Then we categorize all these time series into several groups, such that the time series in the same group share the same optimal dimensionality. Finally we count the number of the time series in each group and summarise it in Table 4.1 and Table 4.2. From Table 4.1, where the Euclidean distance is used as the similarity measure, we can observe that 827 of 1000 time series use 8 as the optimal dimensionality, which means if this dataset is indexed using the R-tree index with the dimensionality 8, the performance could be reasonable. However, there are still 142 time series that require the dimensionality 16 for lower bounding distance calculations. Additionally the group II only has 31 time series while the group I has none, since the dimensionality 2 and 4 are comparatively too small for most time series in this dataset. Although the dimensionalities 32 and 64 are large, the corresponding groups still have no time series because the relative tightness of each time series has already passed the threshold when the dimensionality reaches 16. When DTW distance is selected as the similarity measure, from Table 4.2, we can see that 624 time series use 16 as the optimal dimensionality and 260 time series use 8 as the optimal dimensionality, which almost takes 90% of the whole dataset. So it consequently requires the index to involve at least two dimensionalities. There are still some time series in other groups, but only a small amount.

Table 4.1: The number of the time series in each group categorized by the optimal dimensionalities measured by the Euclidean distance.

<table>
<thead>
<tr>
<th>Group</th>
<th>Dimensionality</th>
<th>Number of Time Series</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>II</td>
<td>4</td>
<td>31</td>
</tr>
<tr>
<td>III</td>
<td>8</td>
<td>827</td>
</tr>
<tr>
<td>IV</td>
<td>16</td>
<td>142</td>
</tr>
<tr>
<td>V</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>VI</td>
<td>64</td>
<td>0</td>
</tr>
</tbody>
</table>

Next, in order to give an idea about how the relative tightness value changes as the dimensionality increases, for each similarity measure, we pick three time
Table 4.2: The number of the time series in each group categorized by the optimal dimensionalities measured by the DTW distance.

<table>
<thead>
<tr>
<th>Group</th>
<th>Dimensionality</th>
<th>Number of Time Series</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>II</td>
<td>4</td>
<td>48</td>
</tr>
<tr>
<td>III</td>
<td>8</td>
<td>260</td>
</tr>
<tr>
<td>IV</td>
<td>16</td>
<td>624</td>
</tr>
<tr>
<td>V</td>
<td>32</td>
<td>55</td>
</tr>
<tr>
<td>VI</td>
<td>64</td>
<td>5</td>
</tr>
</tbody>
</table>

series $S_1$, $S_2$ and $S_3$ from particular groups. The relative tightness values under each dimensionality for each time series are shown in Figure 4.3 and Figure 4.4. The optimal dimensionalities for $S_1$, $S_2$ and $S_3$ are then marked as $A$, $B$, $C$ in each graph.

For the experiment using the Euclidean distance, $S_1$, $S_2$ and $S_3$ are from the group III, II and IV respectively. As shown in Figure 4.3, the relative tightness values of $S_1$ and $S_3$ share a similar trend, increasing dramatically from dimensionality 2 to dimensionality 16 and slowly from the dimensionality 16 to the dimensionality 64. However, they have different values in magnitude, which directly leads to different optimal dimensionalities for them. For $S_1$, its relative tightness starts from 0.2 in the dimensionality 2 and after two increments it reaches approximate 0.62, passing the threshold $\alpha = 0.6$, which makes $A = 8$ as its optimal dimensionality. As for $S_3$, its relative tightness value in each dimensionality is smaller than that of $S_1$. The first dimensionality that makes its relative tightness greater than the threshold $\alpha$ is $B = 16$. Therefore $S_3$’s optimal dimensionality is 16. The evolution of $S_2$’ relative tightness is different from the other two. It starts from a relatively greater value which is 0.3 in the dimensionality 2 and reaches approximate 0.7 in the dimensionality 4. So the optimal dimensionality of $S_2$ is 4. However, it decreases from the dimensionality 16 to 32 and almost remains the same from the dimensionality 16 to 64. This phenomenon reveals the fact that in the function $D_{lbpa1}$, a larger dimensionality does not always make a better lower bound distance. So it is important for us to appropriately choose the dimensionalities before constructing the index.

Figure 4.4 shows the relative tightness values of another three time series in each dimensionality for the DTW lower bound function $D_{lbpa2}$. $S_1$ and $S_3$ are from group IV while $S_2$ is from group III. As we can see from the graph, the evolvement of the three series of the relative tightness from lower dimensionalities to higher ones follows a random pattern. The $rt$ value of $S_1$ begins from 0, and remains 0
4.2 Multiple Optimal Dimensionalities

Experimental Evaluation

Figure 4.3: Optimal dimensionalities for $D_{lbpa1}$ calculations in the wafer dataset.

Even when the dimensionality is increased to 4, which means if we use $S_1$ as a query sequence, and conduct a $k$NN query in a $R^2$-tree index or $R^4$-tree index, not a single candidate time series in the data file can be pruned with the index. Then it starts to increase and passes the threshold for the first time when the dimensionality is 16. For the other two time series, their $rt$ values in each candidate dimensionality and their optimal dimensionalities can be directly seen from the graph, so we do not describe them in detail.

Figure 4.4: Optimal dimensionalities for $D_{lbpa2}$ calculations in the wafer dataset.

In conclusion, the main reason that there may be multiple optimal dimension-
alities in a dataset for time series data to do lower bound distance calculation in
index query processing, is that the similarities between different time series can be
different. For two significantly dissimilar time series, a smaller dimensionality can
make the lower bound distance close to their true distance. For two comparatively
similar time series, a larger dimensionality is necessary to make the lower bounding
distance sufficiently tight.

4.3 Evaluation Method

There are two traditional methods that are used to evaluate the performance of an
index structure. One method is to use the actual processing time. Before a query
starts, the performance evaluator records the start time as \(start\) and when the query
terminates, the performance evaluator records the finishing time as \(end\). Then the
difference between \(end\) and \(start\) is the processing time and it can be compared
among different index structures. The other method is to use the number of I/O
accesses, because I/O is the most time consuming process in an index query and
compared to the I/O access time, the time for the other operations can be neglected.
In this section, firstly we show that the above two methods are not applicable in
our scenario. Then we describe our method to report the query results in the R-tree
index and the MR-index.

4.3.1 The Impact of System I/O cache

In an ideal situation, we expect more time to be required when there are more I/O
accesses in an index query. Therefore, we can use the first method to evaluate a
query result accurately. However in modern operating systems, various caching
strategies are implemented to accelerate I/O access [26][7]. It leads to different be-
haviors in accessing cached I/O and uncached I/O. For cached I/O, since it is stored
in the main memory, it only takes a short time to access. For the uncached I/O,
the operating system not only needs to fetch data from the device, but also needs to
cache it in the main memory, which takes much longer to process.

In our query experiments, we need to perform a group of consecutive queries on
an index in order to measure an average performance. Therefore, for the first few
queries, all I/O accesses could be uncached. Then as the index file is read again and
again, more and more disk pages will be cached in the main memory, so the I/O accesses of the next few queries could be either cached or uncached. Also, there is a chance that the index file is all cached in the main memory, after that all the I/O access of the queries would be cached. Suppose the total number of I/O accesses of the above consecutive queries are the same for each, then the real time calculated by the first method would become less and almost remains the same for the last few queries, which cannot evaluate the actual performance accurately.

We demonstrate the above phenomenon with the following experiment using the EW dataset. In this experiment, 50 2NN queries are processed consecutively in a $R^{16}$-tree index and the DTW distance is used as the similarity measure. The number of I/O accesses for each query is shown in Figure 4.5 and the real time cost for each query is shown in Figure 4.6. We can see that the real time cost does not correlate well with the actual number I/O accesses, especially for the later queries, due to the impact of caching. It does not always go up when there are more I/Os and go down when there are less I/Os. For example, the number of I/O accesses of the 30th query is larger than that of the 1st query, but its real time cost is only approximately 25% of that of the 1st query. Therefore we can not use the real time cost to evaluate our query efficiently.

![Figure 4.5: Number of I/O accesses for 50 2NN consecutive queries in $R^{16}$-tree index.](image-url)
4.3.2 I/O versus Distance Calculation

The second method uses the number of I/O accesses to evaluate the query results, basing on the assumption that compared to other CPU computations, the I/O operation is the most time consuming process in an index query. This assumption holds for the Euclidean distance, because its time complexity is $O(n)$. However it does not always hold for the DTW distance, especially when the calculation is performed for longer time series, because its time complexity is $O(mn)$.

To compare the time cost between the I/O accesses and DTW calculation, firstly we generate a 1G data file and record the time $t_1$ for randomly accessing $n_1 = 10000$ data pages within the file. Now we assume that all the above I/O accesses are uncached. Then the time $t_{io}$ for a random uncached I/O operation is $t_1 / n_1$. On our experimental desktop, $t_{io} = 7$. Next, we generate individual pairs of random walk time series with different length. For each pair, we calculate their DTW distance for $n_2 = 10000$ times and record the time $t_2$. So the time $t_{dtw}$ for a single DTW calculation at a particular length is $t_2 / n_2$. We show the ratio $p = t_{dtw} / t_{io}$ in each length in Figure 4.7. From the graph, we can see that $p$ increases dramatically as the length of the random walk series increases. The position where $p = 1$ indicates that the DTW calculation at that length takes equal time to an uncached I/O operation.

Therefore for a time consuming similarity measures like the DTW distance, we cannot only use the I/O access number to evaluate the query results, but also need to consider the time that is used to calculate the distance function.
4.3 Evaluation Method

4.3.3 Our Evaluation Methods

We use two different methods to compare the query results between a R-tree index and an MR-index. When the leaf dimensionalities of the R-tree index and MR-index are the same, only the number of index I/O accesses will be used to compare their performance, because the number of data I/O accesses and distance calculations would be the same. However, when the leaf dimensionalities of the R-tree index and MR-index are different, it leads to different number of data I/O accesses and distance calculations, so we use Equation 4.2 to quantize a query result:

\[
IO_{convert} = p + q + q \times \frac{t_{dtw}}{t_{io}},
\]

where \( p \) is the number of index I/O accesses, \( q \) is the number of data I/O accesses, \( t_{dtw} \) is the time that takes to finish a DTW distance calculation and \( t_{io} \) is the time to access an uncached I/O. The values of \( t_{dtw} \) and \( t_{io} \) are obtained using the method that we discussed in Section 4.3.2 before the experiments. The term \( q \times \frac{t_{dtw}}{t_{io}} \) is used to convert the number of distance calculations to the number of I/O accesses. We use the symbol \( IO_{convert} \) to refer to the quantized result.

Additionally, we also use our second method to demonstrate the experimental results in Section 4.3.1 (Figure 4.8). As can be seen, the entire graph is very close to Figure 4.5, because the only difference in each point is \( q \times \frac{t_{dtw}}{t_{io}} \). And for the EW dataset, as the length of the time series is only 153, the value of \( t_{dtw} \) is comparatively very small as can be estimated from Figure 4.7.
4.4 Similarity Query Performance Comparison

In this section, we compare the query performance between the MR-index and the R-tree index. The DTW distance is chosen as the similarity measure for the first set of experiments and the Euclidean distance is used for the second set of experiments.

4.4.1 DTW distance

Firstly, we provide the MR-index configurations for each dataset in Table 4.3. As base lines for comparison, we choose the following three R-tree indexes $R^8$, $R^{16}$ and $R^{32}$ which respectively have dimensionality 8, 16 and 32.

The first experiment evaluates how the DTW warping window size influences the query performance. We use the EW dataset in this experiment. For each warping window size, we execute 50 2NN queries and calculate the average numbers of $IO_{convert}$ or index I/O as we discussed in Section 4.3.3. The comparison among $R^8$, $R^{16}$ and the MR-index is shown in Figure 4.9. As the size of DTW warping win-
Similarity Query Performance Comparison

 Experimental Evaluation

As the window increases, many more time series have to be refined, resulting in an increase of the number of data I/O accesses and DTW calculations. Since our MR-index has a larger leaf dimensionality, the number of the time series that need to be refined is less than that in R8 and R16. Thus the overall performance of the MR-index is superior to the two R-tree indexes for each warping window size. Additionally, for the same reason, the performance of R16 is better than that of R8. The comparison between R32 and the MR-index is shown in Figure 4.10. Here, as mentioned in Section 4.3.3, the index I/O number is to be compared because these two indexes have the same leaf dimensionality. Increasing the DTW warping window size causes an increase in the number of data I/O accesses, and further increases the number of index I/O accesses. However, as we can see from the graph, for each warping window size, the index I/O number of our MR-index is less than that of the R-tree index by a fixed percentage. This is because lots of dissimilar time series can be pruned in the upper levels of the MR-index without further traversing down the index, resulting in less index I/O accesses.

Figure 4.9: IOconvert comparison using different DTW warping window size on indexes with different leaf dimensionalities.

Our second experiment evaluates the kNN query performance for varying k. We use the EW, EF and SYN data sets in this experiment. For each dataset and each k, we execute 50 kNN queries and calculate the averages of IOconvert or IOindex. The results of the EW dataset are shown in Figure 4.11 and Figure 4.12. As we can see from the graph, as k increases, the number of IOconvert or IOindex does not increase.
Figure 4.10: $IO_{\text{index}}$ comparison using different DTW warping window sizes on indexes with the same leaf dimensionality.

dramatically. This is mainly because similar reduced time series data are stored in the same leaf node, and so we do not need to fetch much more leaf nodes or time series from the data file when extracting more nearest neighbors for the query time series. However, for each $k$, the number of $IO_{\text{convert}}$ and $IO_{\text{index}}$ of our MR-index is still less than the R-tree index by a fixed percentage, due to the same reasons we discussed above.

Results for the EF dataset are shown in Figure 4.13 and Figure 4.14. In Figure 4.13, our MR-index is compared with $R^{8}$ and $R^{32}$. The reason that our MR-index can beat $R^{8}$ is that we have a large dimensionality in the leaf, which leads to less number of data I/O accesses and distance calculations. Notice that our MR-index can still beat $R^{32}$, which has a larger leaf dimensionality. This is because when moving from 16 to 32, although the dimensionality doubles, the performance of the lower bound distance does not improve at an equal scale. Then the larger dimensionality only brings a dramatic increase in the number of index I/O, but does not reduce the number of data I/O accesses and distance calculations. It also shows that a larger dimensionality does not always make a better lower bound distance than a smaller one. In Figure 4.14, the index I/O number of our MR-index is less than that of $R^{16}$ with the same leaf dimensionality, since dissimilar time series can be pruned at the upper level of the MR-index, which has a large fan-out.
4.4 Similarity Query Performance Comparison

Experimental Evaluation

Figure 4.11: [EW] $IO_{convert}$ comparison using different $K$ for indexes with different leaf dimensionalities measured by the DTW distance.

Figure 4.12: [EW] $IO_{index}$ comparison using different $K$ for indexes with the same leaf dimensionality measured by the DTW distance.
Figure 4.13: [EF] $IO_{\text{convert}}$ comparison using different K for indexes with different leaf dimensionalities measured by the DTW distance.

Figure 4.14: [EF] $IO_{\text{index}}$ comparison using different K for indexes with the same leaf dimensionality measured by the DTW distance.
Results for the SYN dataset are shown in Figure 4.15 and Figure 4.16, in which our MR-index is compared with $R^8$ and $R^{32}$.

Figure 4.15: [SYN] $IO_{convert}$ comparison using different K for indexes with different leaf dimensionalities measured by the DTW distance I.

### 4.4.2 Euclidean distance

We also implemented the MR-index using the Euclidean distance. In this set of the experiments, we use the EW, ECBF and SYN data sets. The corresponding MR-index configurations are show in Table 4.4. Here we notice that the configuration for the same dataset EW is changed for the Euclidean distance. This is because the we use a different lower bounding function in the MR-index configuration stage. Additionally, as the value of $t_{cd}$ is very small compared to the value of $t_{io}$, we will ignore the last term $q = \frac{t_{cd}}{t_{io}}$ in Equation 4.2 when reporting the experiment results.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>MR-index Config</th>
</tr>
</thead>
<tbody>
<tr>
<td>EW</td>
<td>${ &lt; 5, 8 &gt;, &lt; 4, 8 &gt;, &lt; 3, 16 &gt;, &lt; 2, 16 &gt;, &lt; 1, 16 &gt; }$</td>
</tr>
<tr>
<td>ECBF</td>
<td>${ &lt; 5, 8 &gt;, &lt; 4, 16 &gt;, &lt; 3, 16 &gt;, &lt; 2, 16 &gt;, &lt; 1, 32 &gt; }$</td>
</tr>
<tr>
<td>SYN</td>
<td>${ &lt; 4, 4 &gt;, &lt; 3, 4 &gt;, &lt; 2, 8 &gt;, &lt; 1, 8 &gt; }$</td>
</tr>
</tbody>
</table>
Figure 4.16: [SYN] $\text{IO}_{\text{convert}}$ comparison using different K for indexes with different leaf dimensionalities measured by the DTW distance II.

For the EW dataset, we still evaluate the performance with $k$NN queries in different $k$. The results are shown in Figure 4.17 and Figure 4.18. From the graph, we can see that, similar to the experiments for the DTW distance, the performance of our MR-index is better than the R-tree index with a smaller leaf dimensionality, due to less number of data I/O accesses, and better than the R-tree index with a larger leaf dimensionality, due to a fewer number of index I/O accesses and a small increase in the number of data I/O accesses, and again better than the R-tree index with the same leaf dimensionality, due to fewer index I/Os.

In the ECBF dataset, we compare the index I/O of each 2NN queries between the MR-index and $R^{32}$ (Figure 4.19). From the graph, we can see that the number of index I/O of the two indexes follows the same trend for the 50 2NN queries. This is because they have to access to the same number of data I/O due to their equality in the leaf dimensionality. However, our MR-index still beats the R-tree index in each query for the same reason we discussed before.

Finally in the SYN dataset, we compare the $\text{IO}_{\text{convert}}$ of each 2NN queries between the MR-index and $R^{16}$, $R^{32}$ (Figure 4.20). Since it is the dataset we generated specialized for the PAA transformation, the performance of our MR-index is much better than the R-tree indexes. The reason is that the R-tree index with an unnecessary larger dimensionality only leads to more index I/O accesses, but does not
4.4 Similarity Query Performance Comparison

Experimental Evaluation

Figure 4.17: [EW] $IO_{convert}$ comparison using different $K$ for indexes with different leaf dimensionalities measured by the Euclidean distance.

Figure 4.18: [EW] $IO_{index}$ comparison using different $K$ for indexes with the same leaf dimensionality measured by the Euclidean distance.
4.5 Summary

In this section, after describing the datasets, we carried out experiments to show our motivation behind the MR-index, which is that there could exist multiple optimal dimensionalities for lower bound distance calculation in a time series dataset. Then we explained how we evaluated our experiment results. Finally, we implemented our MR-index both for the DTW distance and the Euclidean distance, and compared $k$NN query performance with various R-tree indexes with different dimensionalities. We showed that our well-configured MR-index is better.
Figure 4.20: [SYN] $IO_{convert}$ comparison for 50 2NN queries for indexes with different leaf dimensionalities measured by the Euclidean distance.
Chapter 5
Conclusions and Future Directions

The purpose of this dissertation was to develop an index structure for one dimensional time series databases. The motivation behind this is that the most commonly used GEMINI framework could only support one dimensionality in the spatial index structure, which may not be optimal for time series data when doing the lower bound distance calculation in query processing. Motivated by this, we have made the following four significant contributions:

- We have developed a multi-resolution index structure for one-dimensional time series databases that can use multiple dimensionalities, rather than the single dimensionality currently used in the GEMINI framework.
- We have developed a method that can automatically obtain the optimal dimensionalities from a time series database and assign them to different index levels for index pre-construction.
- We also showed that our index structure can be applied to any similarity measures and dimensionality reduction strategies, and proved that no false dismissal would occur in index queries.
- Experimental results showed that well configured MR-index could not only beat the R-tree indexes with larger leaf dimensionalities, but also could beat the R-tree indexes with the same or smaller dimensionalities.

A summary of our work is presented in Section 5.1. In Section 5.2, we discuss future directions.

5.1 Summary

In Chapter 1, we briefly discussed the field of time series, including its definition and real world applications, various similarity measures and representation meth-
5.1 Summary

Conclusions and Future Directions

ods. We also presented our motivation for developing a new index structure using a detailed example, which is that the single dimensionality applied in the GEMINI framework may not be the optimal one for all the time series within a data set to do the lower bounding distance calculations in index queries.

In Chapter 2, we described existing works on lower bounding strategies for the commonly used Euclidean and DTW distance under different representation methods. Some of these approaches can be extended in our index structure as well, so long as they meet the conditions in Section 3.4, for example, the DFT and the Haar Wavelet transform for the Euclidean distance and the PAA transform for the DTW distance. Then, after giving the definitions of the range and \( k \)NN query in a time series database, we described the two naive linear scan algorithms for these two queries at first. In these algorithms, all the time series need to be fetched from the disk and are required for distance calculations, which is a time consuming process. Then we explained how lower bounding strategies can be applied for the naive linear scan algorithms, which we referred to as the linear scan with lower bounding. Although in the linear scan with lower bounding algorithms, we still need to read all the time series from the disk, we do not need to do all the distance calculations because of the pruning of the lower bounding function. Next we briefly reviewed the R-tree, including its basic properties, insertion algorithms with three types of splitting and search algorithms, since it is the most commonly used spatial index in the GEMINI framework. Finally we explained how lower bounding is applied in the GEMINI framework and described its range query and \( k \)NN query algorithms.

In Chapter 3, we presented the structure of our MR-index, including the node structure, how the fan-out of each node is determined and how the MBR of each entry is constructed. We also illustrated our MR-index with an example. The major difference between MR-index and R-tree index can be summarized as: 1) The dimensionality at each level in a MR-index can be different, resulting in different fan-outs in different levels, while in an R-tree index the dimensionality is a fixed value. 2) The MBR of each entry in a MR-index is directly constructed from all the reduced time series that are under this entry, while in an R-tree index it is constructed by all the MBRs in its child node. Then we explained the algorithms for inserting a new time series into a MR-index. These algorithms guarantee that our MR-index is a balanced tree and no false dismissal will occur in an index query. Next, we described how the range query and \( k \)NN query are conducted in our MR-index. Afterwards we gave the definition of the relative tightness for a lower bounding strategy,
Conclusions and Future Directions

5.2 Future Directions

which is the key to determine the optimal dimensionality for a time series to do the lower bounding distance calculation in an index query. Based on that, we explained a method that can automatically retrieve the optimal dimensionalities from a time series data set and assign them into different tree levels for pre-constructing a MR-index. Finally we provided the conditions that should be met for our MR-index to be extended to a particular dimensionality reduction method and proved that index queries would not bring any false dismissals.

In Chapter 4, after briefly describing the data sets we used, we conducted an experiment to show our research motivation again, which is that even in the same time series data set, the optimal dimensionality for different time series data to do the lower bounding distance calculations in index queries can be different, therefore we need to apply multiple dimensionalities in the index structure. Then we described our own method to evaluate query results, by converting the number of distance calculation into the number of disk I/O access. Finally, we presented the results of our query experiments conducted on real world and synthetic data sets. From the experiments, we can observe that our MR-index could beat the R-tree index with the same and a larger leaf dimensionality, due to fewer number of internal nodes access. It could also beat R-tree with a smaller leaf dimensionality, due to fewer number of data I/O access and distance calculation.

5.2 Future Directions

In the insertion algorithm, when a split occurs, we need to re-construct two MBRs in the parent node, one for the old child and one for the new child respectively. Then we have to retrieve all the time series data that are under these two entries from the data file due to the possible difference in dimensionality for different levels. This is a time consuming process, which can be a bottle neck for the insertion algorithm. We have already optimized it by applying a main memory LRU [44] buffer to cache the used time series data. However, we still would like to find alternative ways to address this issue.

Secondly, we can observe that the proposed automatic optimal dimensionality retrieval method is based on the time series data set to be indexed, which is assumed to be static. Therefore the dimensionalities in an existing MR-index only reflect the information for the original data set. For any inserted new time series, other than the ones from the original data set, we can not guarantee that the optimal
5.2 Future Directions

Dimensionality is within the current MR-index. So a future step is how to maintain the optimal dimensionality information for time series data that are indexed with an MR-index, including both the original data set and time series that are newly inserted. In addition, we also need strategies to adjust the MR-index according to changes in the optimal dimensionality. For example, due to the insertion of some time series, we may find a new optimal dimensionality that should be added to the existing MR-index.

Finally, the similarity queries that can be used in our MR-index are based on whole matching. Another direction is to extend the MR-index so that we can do subsequence matching as well.
5.2 Future Directions


5.2 Future Directions

Conclusions and Future Directions


Conclusions and Future Directions

5.2 Future Directions


[27] E. Keogh and A. Ratanamahatana. Everything you know about dynamic time warping is wrong. 3rd Workshop on Mining Temporal and Sequential Data, in conjunction with 10th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD-2004), Seattle, WA, 2004.
5.2 Future Directions

Conclusions and Future Directions


5.2 Future Directions


5.2 Future Directions

Conclusions and Future Directions


Conclusions and Future Directions

5.2 Future Directions


5.2 Future Directions

Minerva Access is the Institutional Repository of The University of Melbourne

Author/s: Ma, Mei

Title: Multi-resolution indexing method for time series

Date: 2010


Persistent Link: http://hdl.handle.net/11343/35340

File Description: Multi-resolution indexing method for time series

Terms and Conditions: Copyright in works deposited in Minerva Access is retained by the copyright owner. The work may not be altered without permission from the copyright owner. Readers may only download, print and save electronic copies of whole works for their own personal non-commercial use. Any use that exceeds these limits requires permission from the copyright owner. Attribution is essential when quoting or paraphrasing from these works.